Neural Software Abstractions

Michael Chang

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-172
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-172.html

May 12, 2023

Acknowledgement

I want to thank my committee members, Alison Gopnik, Jonathan Ragan-Kelley, Sergey Levine, and Tom Griffiths for their support in my critical Ph.D. milestones. I thank my advisor Sergey Levine for challenging me to think critically and clearly. I thank my advisor Tom Griffiths for his steadfast support and guidance in my intellectual and professional development. I want to thank all my collaborators for many exciting research projects. I want to thank all the deep relationships I have formed over this journey. I would like to thank my family, for all the sacrifices they had made to help me on my journey and their love and presence no matter what happens.

Neural Software Abstractions

By

Michael Chang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sergey Levine, Co-chair
Professor Thomas L. Griffiths, Co-chair
Professor Alison Gopnik
Professor Jonathan Ragan-Kelley

Spring 2023

Neural Software Abstractions

Abstract

Neural Software Abstractions

by

Michael Chang

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Sergey Levine, Co-chair

Professor Thomas L. Griffiths, Co-chair

The desire to efficiently solve problems has driven humans to create tools to accomplish more with less. To be useful in a variety of contexts, a tool must encode knowledge of how to solve a general problem, knowledge that models the system that the tool manipulates. For most of human history, tools enabled humans to better manipulate only physical systems, such as using a lever for lifting heavy objects. These tools implicitly modeled the physical system via their specialized design. The computer is significant because it was the first universal tool for modeling and manipulating any system.

Unfortunately, this universality has historically been restricted to systems that only humans can manually model and manipulate, via code. Humans have long acted as the interface between computers and the physical world, but we will increasingly become the bottleneck to progress as computers become more powerful and the world becomes more complex. If we could build machines that automatically model and manipulate systems on their own, then we would solve more problems with less effort: we would need only specify *what* the problem is rather than bother with *how* to solve it.

The problem of building machines that automatically model and manipulate systems is not new and arguably encompasses the entire field of artificial intelligence (AI). Solving such a problem implies two things: first, that the machine can represent system interactions and second, that the machine can learn such representations automatically. What it means to represent system interactions is to represent the entities in the environment, the transformations that change the state of these entities, and choices the agent makes to apply these transformations. What it means to learn representations automatically is for these representations to be learned functions of the machine's raw sensorimotor stream. For such representations to be effective for automatically modeling and manipulating systems, they need to generalize over the

combinatorial space of possible combinations of entities, of transformations, and of choices, and criterion that I call **combinatorial generalization**.

Neither of the two paradigms that have dominated AI since the mid-1900s have yet offered a complete solution to both desiderata. The symbolic paradigm offers solutions for how to represent system interactions but not for how to learn representations. Conversely, the connectionist paradigm offers solutions for how to learn representations, but generally such representations do not directly expose the entities, transformations, and choices of the underlying system interaction in question. In the last half century these two paradigms have grown into the modern disciplines of software programming and deep learning, largely retaining their original complementary strengths and weaknesses. How can we achieve the strengths of both?

One prominent class of approaches for combining both paradigms is to use neural networks for processing symbolic data or searching over symbolic code. These methods have achieved great success in natural language processing, code generation, and symbolic search, but they all assume a human-defined abstraction of the system to begin with. To actually address the problem of automatically modeling and manipulating systems, we need the machine to create these abstractions from its own sensorimotor experience. We need to combine both paradigms in a different way. What we would want instead are AI methods that can learn directly from raw data as deep learning algorithms do, with learned representations that generalize over the combinatorial space of system interactions as software does.

My central thesis is that there is a deep similarity between electronic circuits and neural networks, and that adapting the methods we invented almost a century ago for creating modular software programs on top of analog circuits can enable neural networks to exhibit similar generalization properties as software does. I argue that the principle of **separation of concerns** was the key design principle that enabled representations in software to generalize and that **contextual refinement** was the key technique that enabled us to implement the principle of separation of concerns at every level of the computing stack. This thesis presents various ways for how to instantiate contextual refinement in neural networks and shows the gains in combinatorial generalization that this technique brings.

To my family.

Contents

Contents **ii**

List of Figures **vi**

List of Tables **xx**

1 Introduction **1**
1.1 Contextual refinement 5
1.2 Neural software abstractions 7
1.3 Outline and Contributions 10

I Entities **11**

2 Representing Static Entities **12**
2.1 Introduction 12
2.2 Related Work 14
2.3 Background 15
2.4 Implicit Iterative Refinement 16
2.5 Experiments 18
2.6 Discussion 25

3 Representing Dynamic Entities **27**
3.1 Introduction 27
3.2 Related Work 29
3.3 Problem Formulation 31
3.4 Object-Centric Perception, Prediction, and Planning (OP3) 32
3.5 Experiments 37
3.6 Discussion 41

II Transformations **42**

4 Representing Physical Transformations **43**
4.1 Introduction . . . 43
4.2 Related Work . . . 45
4.3 Goal-Conditioned Reinforcement Learning with Entities . . . 46
4.4 Neural Constraint Satisfaction . . . 48
4.5 Experiments . . . 54
4.6 Discussion . . . 57

5 Representing Virtual Transformations **58**
5.1 Introduction . . . 58
5.2 Compositional Generalization . . . 59
5.3 A Learner That Programs Itself . . . 61
5.4 Experiments . . . 64
5.5 Related Work . . . 67
5.6 Discussion . . . 69

III Choices **73**

6 Representing Policies as Games **74**
6.1 Introduction . . . 74
6.2 Related Work . . . 76
6.3 Preliminaries . . . 77
6.4 Societal Decision-Making . . . 78
6.5 Mechanism Design for the Society . . . 79
6.6 From Equilibria to Learning Objectives . . . 82
6.7 Experiments . . . 83
6.8 Discussion . . . 91

7 Local Credit Assignment **92**
7.1 Introduction . . . 92
7.2 Related Work . . . 94
7.3 Background . . . 95
7.4 Dynamic Modularity in Learning Systems . . . 97
7.5 An Algorithmic Causal Model of Learning . . . 100
7.6 Modularity in Reinforcement Learning . . . 101
7.7 Simple Experiments . . . 103
7.8 Discussion . . . 108

IV Conclusion **110**

Bibliography **113**

A Representing Static Entities **136**
A.1 Future Work . . . 136
A.2 Further Experiments . . . 139

B Representing Dynamic Entities **142**
B.1 Observation Model . . . 142
B.2 Evidence Lower Bound . . . 143
B.3 Posterior Predictive Distribution . . . 144
B.4 Interactive Inference . . . 145
B.5 Cost Function . . . 146
B.6 Architecture and Hyperparameter Details . . . 147
B.7 Experiment Details . . . 149
B.8 Ablations . . . 152
B.9 Interpretability . . . 153

C Representing Physical Transformations **154**
C.1 Implementation Details . . . 154
C.2 Baseline Implementation Details . . . 157
C.3 Environment Details . . . 159
C.4 Additional Results . . . 159
C.5 Combinatorial Space . . . 161
C.6 Limitations and future work. . . . 162
C.7 Why the name "Neural Constraint Satisfaction?" . . . 162

D Representing Virtual Transformations **163**
D.1 Data . . . 163
D.2 Learner Details . . . 164
D.3 Experiment Details . . . 165
D.4 Additional Experiments . . . 167

E Representing Policies as Games **172**
E.1 Game Theory . . . 172
E.2 Societal Decision-Making Framework . . . 173
E.3 The Cloned Vickrey Society as a Solution to Societal Decision-Making . . . 173
E.4 Decentralized RL Algorithms for the Cloned Vickrey Society . . . 175
E.5 Implementations of an On-Policy Decentralized RL Algorithm . . . 175
E.6 Experimental Details . . . 177

F Local Credit Assignment **182**

F.1 Background 182
F.2 Assumptions 186
F.3 Additional Theoretical Results 187
F.4 Proofs 188
F.5 Simulation Details 193

List of Figures

2.1 **Overview.** We address efficient training of iterative refinement methods for learning representations of latent sets, such as the slot attention model in (a), whose illustration is adapted from Locatello et al. [208]. Vanilla slot attention backpropagates gradients through the unrolled iterative refinement procedure, which leads to training instabilities as shown by the growing Jacobian spectral norm (b). Implicit slot attention uses the first-order Neumann approximation of the implicit gradient, which simply truncates the backpropagation, leading to substantially more effective training, as shown in (c). . . 13

2.2 **Code.** The first order Neumann approximation to the implicit gradient adds only one additional line of Pytorch code [243] to the original forward function of slot attention, but yields substantial improvement of optimization. `attn` and `slots` correspond to ϕ and θ in the text respectively. 18

2.3 **Can slot attention by trained as a fixed point operation with implicit differentiation?** (2.3a) The relative residual of the forward iteration of slot attention is close to zero, which motivates its treatment as a fixed point operation. (2.3b) The forward and backward computations of vanilla slot attention can be swapped out with different solvers, most of which result in the same improved optimization performance. Here we compare with *IB*, *BB*, *IN*, and *BN* variants of implicit differentiation across 4 seeds. Table 2.2 defines these acronyms. 19

2.4 **Qualitative results.** Across three datasets, optimizing SLATE with implicit differentiation leads to improved image reconstructions through the slot bottleneck. Black borders indicate the ground truth image, blue border indicate our method, and red borders indicate vanilla SLATE. The rest of the panels visualize attention masks. In the CLEVR-Mirrors dataset, whereas vanilla SLATE misses objects, changes their size, or changes their color (indicated by the circles), implicit SLATE reconstructs the ground truth more faithfully. 21

2.5 **Does implicit differentiation remove the need for various optimization tricks?** We ablate three heuristically-motivated optimization tricks from both vanilla SLATE and our method, and show that for two out of the three, removing the optimization trick quantitatively hurts the vanilla model but not the implicit model. Whereas removing gradient clipping and learning rate warmup causes vanilla SLATE's training to become unstable, as indicated by the growth of the Jacobian norm of the slot attention cell, our method trains significantly more stably and can take advantage of the larger gradient steps. 23
2.6 (a) Without gradient clipping, our implicit differentiation technique keeps gradients small while backpropagating through the unrolled iterations causes gradients to explode. (b) Training with implicit differentiation also is not sensitive to the number of iterations with which to iterate the slot attention cell. (c) Using one iteration for vanilla slot attention trains as stably, but reconstructs more poorly, than implicit slate. 24
2.7 We outperform vanilla slot attention on object property prediction. 24
2.8 Implicit differentiation preserves the quality of predicted segmentations from Locatello et al. [208]. 25

3.1 **OP3**. **(a)** OP3 can infer a set of entity variables $H_{1:K}^{(T)}$ from a series of interactions (interactive entity grounding) or a single image (entity grounding). OP3 rollouts predict the future entity states $H_{1:K}^{(T+d)}$ given a sequence of actions $a^{(T:T+d)}$. We evaluate these rollouts during planning by scoring these predictions against inferred goal entity-states $H_k^{(G)}$. **(b)** OP3 enforces the **entity abstraction**, factorizing the latent state into *local* entity states, each of which are symmetrically processed with the same function that takes in a *generic entity* as an argument. In contrast, prior work either **(c)** process a global latent state [145] or **(d)** assume a fixed set of entities processed in a permutation-sensitive manner [94, 184, 340, 326]. **(e-g)** Enforcing the entity-abstraction on modeling the **(f)** dynamics and **(g)** observation distributions of a POMDP, and on the **(e) interactive inference** procedure for grounding the entity variables in raw visual observations. Actions are not shown to reduce clutter. 28

3.2 **Comparison with other methods.** Unlike other methods, OP3 is a fully probabilistic factorized dynamic latent variable model, giving it several desirable properties. First, OP3 is naturally suited for combinatorial generalization [28] because it enforces that local properties are invariant to changes in global structure. Because every learnable component of the OP3 operates symmetrically on each entity, including the mechanism that disambiguates entities itself (c.f. COBRA, which uses a learned autoregressive network to disambiguates entities, and Transporter and C-SWMs, which use a forward pass of a convolutional encoder for the global scene, rather than each entity), the weights of OP3 are invariant to changes in the number of instances of an entity, as well as the number of entities in the scene. Second, OP3's recurrent structure makes it straightforward to enforce spatiotemporal consistency, object permanence, and refine the grounding of its entity representations over time with new information. In contrast, COBRA, Transporter, and C-SWMs all model single-step dynamics and do not contain mechanisms for establishing a correspondence between the entity representations predicted from the previous timestep with the entity representations inferred at the current timestep. 30

3.3 **(a)** The observation model $\mathcal{G}$ models an observation image as a composition of sub-images weighted by segmentation masks. The shades of gray in the masks indicate the depth δ from the camera of the object that the sub-image depicts. **(b)** The graphical model of the generative model of observations, where k indexes the entity, and i, j indexes the pixel. Z is the indicator variable that signifies whether an object's depth at a pixel is the closest to the camera. 33

3.4 The **dynamics model** $\mathcal{D}$ models the time evolution of every object by symmetrically applying the function ℓ to each object. For a given object, d models the individual dynamics of that object (d_o), embeds the action vector (d_a), computes the action's effect on that object (d_{ao}), computes each of the other objects' effect on that object (d_{oo}), and aggregates these effects together (d_{comb}). 34

3.5 **Amortized interactive inference** alternates between refinement (pink) and dynamics (orange) steps, iteratively updating the belief of $\lambda_{1:K}$ over time. $\hat{\lambda}$ corresponds to the output of the dynamics network, which serves as the initial estimate of λ that is subsequently refined by $f_{\mathcal{G}}$ and $f_{\mathcal{Q}}$. ∇ denotes the feedback used in the refinement process, which includes gradient information and auxiliary inputs (Appdx. B.4). . . . 36

3.6 **(a)** In the block stacking task from [164] with single-step greedy planning, OP3 generalizes better than both O2P2, an oracle model with access to image segmentations, and SAVP, which does not enforce entity abstraction. **(b)** OP3 exhibits better multi-step planning with objects already present in the scene. By planning with MPC using random pick locations (SAVP and OP3 (xy)), the sparsity of objects in the scene make it rare for random pick locations to actually pick the objects. Because OP3 has access to pointers to the latent entities, we can use these to automatically bias the pick locations to be at the object location, without any supervision (OP3 (entity)). 39

3.7 Visualization of interactive inference for block-manipulation and real-world videos [82]. Here, OP3 interacts with the objects by executing pre-specified actions in order to disambiguate objects already present in the scene by taking advantage of temporal continuity and receiving feedback from how well its prediction of how an action affects an object compares with the ground truth result. **(a)** OP3 does four refinement steps on the first image, and then 2 refinement steps after each prediction. **(b)** We compare OP3, applied on dynamic videos, with IODINE, applied independently to each frame of the video, to illustrate that using a dynamics model to propagate information across time enables better object disambiguation. We observe that initially, both OP3 (green circle) and IODINE (cyan circles) both disambiguate objects via color segmentation because color is the only signal in a static image to group pixels. However, we observe that as time progresses, OP3 separates the arm, object, and background into separate latents (purple) by using its currently estimates latents predict the next observation and comparing this prediction with the actually observed next observation. In contrast, applying IODINE on a per-frame basis does not yield benefits of temporal consistency and interactive feedback (red). 40

4.1 NCS uses a two-level hierarchy to abstract sensorimotor interactions into a graph of learned state transitions. The affected entity is in black. 44

4.2 **Solving object rearrangement requires solving two challenges**. (a) The **correspondence problem** is the problem of abstracting raw sensorimotor signal into representations of entities such that there is a correspondence between how an agent intervenes on an entity and how its action affects an object in the environment. k denotes the index of the entity, z denotes its type (shown with solid colors), and s denotes its state (shown with textures). The entity representing the moved object is in black. (b) The **combinatorial problem** is the problem of representing the combinatorial task space in a way that enables an agent to transfer knowledge of a given state transition (indicated by the dotted circle) to different contexts. 46

4.3 **Modeling**. NCS constructs a two-level abstraction hierarchy to model transitions in the experience buffer. (a) **Level 1:** NCS learns to infer a set of entities from sensorimotor transitions with pick-and-move actions, in which one entity is moved per transition. We enforce that the type z (shown with solid colors) of an entity remains unchanged between time-steps. The GPT dynamics model learns to sparsely predict the states s (shown with textures) of the entities at the next time-step. *This addresses the correspondence problem by forcing the network to use predict and reconstruct observations through the entity bottleneck.* (b) **Level 2:** NCS abstracts transitions over entity-sets into transitions over states of individual entities, constructing a graph where states are nodes and transitions between them are edges. This is done by clustering entity transitions that share similar initial states and final states. *This addresses the combinatorial problem by making it possible for state transitions to reused for different entity types and with different context entities.* 48

4.4 **Planning and control**. Given a rearrangement problem specified only by the current and goal observations (o_0, o_g), NCS decomposes the rearrangement problem into one subproblem (o_t, o_g) per entity. (a) shows the computations NCS uses to solve each subproblem and (b-d) show these steps in context. For each subproblem (o_t, o_g), NCS infers entities from both the current and goal observations. The states of the goal entities indicate constraints on the desired locations of the current entities. (b) NCS aligns the indices of the current entities to those of the goal entities with corresponding types. (c) It selects the index k of the next goal constraint s_g^k to satisfy, as indicated by the red box. The selected goal constraint and current entity are also colored black in (a), and note that their types are the same but states are different; we want to choose the action to transform the state of the current entity to the state of the goal constraint. (d) It binds the selected goal constraint and its corresponding current entity to nodes s_* and s'_* in the transition graph. Lastly, it identifies the edge connecting those two nodes and executes the action tagged to that edge in the environment. 52

4.5 Our environments are *block-rearrange* and *robogym-rearrange*. Fig. 4.5a shows a complete specification of goal constraints; Fig. 4.5b shows a partial specification that only specifies the desired locations for two objects. 54

4.6 **Nodes as equivalent classes over states.** We show a clustering of states inferred for *robogym-rearrange*, where each cluster centroid is treated as a node in our transition graph. A subset of clusters are labeled with an attention mask computed by averaging the slot attention masks for the entities associated with the cluster. 57

5.1 **(a)** Consider a multitask family of problems, whose subproblems are shared within and across problems. Standard approaches either **(b)** train a separate learner per task or **(c)** train a single learner for all tasks. Both have difficulty generalizing to longer compositional problems. **(d)** Our goal is to re-use previously learned sub-solutions to solve new problems by composing computational modules in new ways. 60

5.2 **Compositional recursive learner (CRL):** *top-left*: CRL is a symbiotic relationship between a controller and evaluator: the controller selects a module m given an intermediate representation x and the evaluator applies m on x to create a new representation. *bottom-left*: CRL learns dynamically learns the structure of a program customized for its problem, and this program can be viewed as a finite state machine. *right*: A series of computations in the program is equivalent to a traversal through a Meta-MDP, where module can be reused across different stages of computation, allowing for recursive computation. 62

5.3 **Multilingual Arithmetic (Quantitative).** CRL generalizes significantly better than the RNN, which, even with ten times more data, does not generalize to 10-length multilingual arithmetic expressions. Pretraining the RNN on domain-specific auxiliary tasks does not help the 10-length case, highlighting a limitation of using monolithic learners for compositional problems. By comparing CRL with a version trained without a curriculum ("No Curr": blue), we see the benefit of slowly growing the complexity of problems throughout training, although this benefit does not transfer to the RNN. The vertical black dashed line indicates at which point all the training data has been added when CRL is trained with a curriculum (red). The initial consistent rise of the red training curve before this point shows CRL exhibits forward transfer [209] to expressions of longer length. Generalization becomes apparent only after a million iterations after all the training data has been added. **(b, c)** only show accuracy on the expressions with the maximum length of those added so far to the curriculum. "1e4" and "1e5" correspond to the order of magnitude of the number of samples in the dataset, of which 70% are used for training. 10, 50, and 90 percentiles are shown over 6 runs. 65

5.4 **Left:** For multilingual arithmetic, blue denotes the language pairs for training and red denotes the language pairs held out for evaluation in Fig 5.3b,c. **Center:** For transformed MNIST classification, blue denotes the length-2 transformation combinations that produced the input for training, red denotes the length-2 transformation combinations held out for evaluation. Not shown are the more complex length-3 transformation combinations (scale then rotate then translate) we also tested on. **Right:** For transformed MNIST classification, each learner performs better than the others in a different metric: the CNN performs best on the training subproblem combinations, the STN on different subproblem combinations of the same length as training, and CRL on longer subproblem combinations than training. While CRL performs comparably with the others in the former two metrics, CRL's $\sim 40\%$ improvement for more complex image transformations is significant. 67

5.5 **Multilingual Arithmetic (Qualitative).** A randomly selected execution trace for generalizing from length-5 to length-10 expressions. The input is $0-6+1+7\times3\times6-3+7-7\times7$ expressed in Pig Latin. The desired output is *seis*, which is the value of the expression, 6, expressed in Spanish. The purple modules are reducers and the red modules are translators. The input to a module is highlighted and the output of the module is boxed. The controller learns order of operations. Observe that reducer m_9 learns to reduce to numerals and reducer m_{10} to English terms. The task-agnostic nature of the modules forces them to learn transformations that the controller would commonly reuse across problems. Even if the problem may not be compositionally structured, such as translating Pig Latin to Spanish, CRL learns to design a compositional solution (Pig Latin to Numerals to Spanish) from previous experience (Pig Latin to Numerals and Numerals to Spanish) in order to generalize: it first reduces the Pig Latin expression to a numerical evaluation, and then translates that to its Spanish representation using the translator m_6. Note that all of this computation is happening internally to the learner, which computes on softmax distributions over the vocabulary; for visualization we show the token of the distribution with maximum probability. 71

5.6 **Image Transformations:** CRL reasonably applies a sequence of modules to transform a transformed MNIST digit into canonical position, and generalizes to different and longer compositions of generative transformations. m_0 is constrained to output the sine and cosine of a rotation angle, m_1 is constrained to output the scaling factor, and m_2 through m_{13} are constrained to output spatial translations. Some modules like m_2 and m_6 learn to translate up, some like m_3 and m_{10} learn to translate down, some like m_7 learn to shift right, and some like m_{13} learn to shift left. Consider (d): the original generative transformations were "scale big" then "translate left," so the correct inversion should be "translate right" then "scale small." However, CRL chose to equivalently "scale small" and then "translate right." CRL also creatively uses m_0 to scale, as in (e) and (f), even though its original parametrization of outputting sine and cosine is biased towards rotation. 72

6.1 We study how a society of primitives agents can be abstracted as a super-agent. The incentive mechanism is the abstraction barrier that relates the optimization problems of the super-agent with those of its constituent primitive agents. 75

6.2 **The cloned Vickrey society.** In this market economy of primitive agents, wealth is distributed not directly from the global MDP objective but based on what future primitives decide to bid for the fruits of the labor of information processing carried out by past primitives transforming one state to another. The primitive $\omega_t^{0'}$ that wins the auction at time t receives an environment reward $r(s_t, \omega_t^{0'})$ as well as payment $b_{t+1}^{1'}$ from $\omega_{t+1}^{1'}$ for transforming s_t to s_{t+1}. By the Vickrey auction, the price $\omega_{t+1}^{1'}$ pays to transform s_{t+1} is the second highest bid b_{t+1}^{1} at time $t+1$. Because each primitive ω^i and its clone $\omega^{i'}$ have the same valuations, their bids are equivalent and so credit is conserved through time. 81

6.3 **Market Bandits.** We compare the cloned Vickrey society (Cloned Vickrey Auction) against solitary societies that use the first-price auction mechanism (first price auction), the Vickrey auction mechanism (Vickrey Auction), and a mechanism whose utility is only the environment reward (Environment Reward). The dashed line in (a) indicates truthful bidding of valuations. The cloned Vickrey society's bids are closest to the true valuations which also translates into the best global policy (b). 84

6.4 **Implementations.** The table shows the bid price that temporally consecutive winners $\hat{\omega}_{t+1}$ and $\hat{\omega}_t$ pay and receive based on three possible implementations of the cloned Vickrey society: *CCV*, *BB*, *V*, with tradeoffs depicted in the Venn diagram. We use $\hat{\mathbf{b}}_{t+1}$ and $\mathbf{b}'_{t+1}$ to denote the highest and second highest bids at time $t+1$ respectively. 85

6.5 **Learned Bidding Strategies for *Chain*.** We organize the analysis by distinguishing between the credit-conserving (*CCV* and *BB*) and the non-credit-conserving (*V*) implementations. The solitary *CCV* (a) and *BB* (b) implementations learn to bid very close to 0: *CCV* because the valuation for a primitive at t is only the second-highest bid at $t+1$, resulting in a rapid decay in the valuations leftwards down the chain; *BB* because each primitive is incentivized to pay as low of a price for winning as possible. The cloned *CCV* (d) and *BB* (e) implementations learn to implement a form of return decomposition [16] that redistributes the terminal reward into a series of positive payoffs back through the chain, each agent getting paid for contributing to moving the society closer to the goal state, where the *CCV* implementation's bids are closer to the optimal societal Q-value than those of the *BB* implementation. Because both the solitary (c) and cloned (f) versions of the *V* implementations do not conserve credit, they learn to bid close to the optimal societal Q-value, but both suffer from market bubbles where the primitive for going left bids higher than the primitive for going right, even though the optimal global policy is to keep moving right. 86

6.6 **Multi-Step MDPs.** (a) In the *Chain* environment, the society starts at state s_0 and the goal state is s_5. Only activating primitive ω_1 at state s_4 yields reward. The optimal global policy is to directly move right by continually activating ω_1. Without credit conservation, the society may get stuck going back and forth between s_0 and s_4 without reaching the goal. (b) In the *Duality* environment, the society starts at state s_0. s_{-1} is an absorbing state with perpetual negative rewards. The optimal societal policy is to cycle between s_0 and s_1 to receive unbounded reward, but without redundant primitives, the society may end up in a suboptimal perpetual self-loop at s_1. 87

6.7 **CCV Bidding Curves for *Duality*.** Each column shows the bidding curves of the solitary (top row) and cloned (bottom row) *CCV* societies for states s_{-1}, s_0, and s_1. Without redundant primitives to force the second-highest and highest valuations to be equal, the dominant strategy of truthful bidding may not coincide with the globally optimal policy because the solitary *CCV* implementation does not guarantee Bellman optimality. The bidding curves in (c) show that ω^1 learns a best response of bidding *higher* than primitive ω^0 at state s_1, even though it would be globally optimal for the society if ω^0 wins at s_1. Adding redundant primitives causes the second-highest and highest valuations to be equal, causing ω^0 to learn to bid highly as well at s_1, which results in a more optimal return as shown in Figure 6.8b. 88

6.8 **Multi-Step MDP Global Learning Curves.** We observe that cloned societies are more robust against suboptimal equilibria than solitary societies. Furthermore the cloned *CCV* implementation achieves the best sample efficiency, suggesting that truthful bidding and credit-conservation are important properties to enforce for enabling the optimal global policy to emerge. 89

6.9 **Mental Rotation.** The cloned Vickrey society learns to transform the image into a form that can be classified correctly with a pre-trained classifier by composing two of six possible affine transformations of rotation and translation. Clones are indicated by an apostrophe. In this example, the society activated primitive $\omega^{2'}$ to translate the digit up then primitive ω^1 to rotate the digit clockwise. Though the bidding policies ψ^i and $\psi^{i'}$ of the clones ω^i and $\omega^{i'}$ have the same parameters, their sampled bids may be different because the bidding policies are stochastic. 89

6.10 **Two Rooms.** The cloned Vickrey society adapts more quickly than the hierarchical monolithic baseline in both the pre-training and the transfer tasks. The bottom-right figure, which is a histogram of the absolute values of how much the weights have shifted from fine-tuning on the transfer task, shows that more weights shift, and to a larger degree, in the hierarchical monolithic baseline than in our method. This seems to suggest that the cloned Vickrey society is a more modular learner than the hierarchical monolithic baseline. The non-hierarchical monolithic baseline does not learn to solve the task from scratch. 90

7.1 **Minimal motivating example.** The optimal action sequence for the training task is $A \to B \to C$, and the optimal sequence for the transfer task differs only in the last time-step. Continuing to train an optimal policy from the training task on the transfer task with the cloned Vickrey society (CVS) from Chang et al. [54] transfers 13.9x more efficiently than with PPO [282], even though learning efficiency for both on-policy algorithms during training is comparable. This chapter suggests that this is due to **dynamic modularity**: the algorithmic independence among CVS's learnable mechanisms and among their gradients. 93

7.2 **Key Ideas.** A system can be represented as a algorithmic causal graph $\mathtt{G}$. (a) A modification to a mechanism in graph $\mathtt{G}^i$ generates a new graph $\mathtt{G}^{i+1}$. (b) A dynamic system encompasses an outer process that generates a sequence of graphs via a series of modifications to the mechanisms. (c) Learning algorithms are examples of dynamic systems, where the outer process is the model of credit assignment $\mathtt{C}$, which modifies the mechanisms of the model of execution $\mathtt{E}$, which represents the forward pass of the learner. By flattening the learning algorithm as one algorithmic causal graph, we can determine whether the causal structure of the credit assignment mechanism makes independent modification of learnable mechanisms possible by inspecting whether the gradients are d-separated by the previous graph $\mathtt{C}^i$. A credit assignment mechanism is (d) modular if they are d-separated and (e) not modular if not. 97

7.3 **Modularity in RL.** In RL, the forward pass is a rollout in the MDP. (a) The societal decision-making framework exposes the learnable decision mechanisms of the policy as separate components in the model of execution. The bids b represent either action probabilities are estimated action-specific Q-values. The credit assignment mechanisms of (b) policy gradient methods and (c) TD($n > 1$) methods, like using Monte Carlo estimation, contain shared hidden variable and thus do not produce algorithmically independent gradients δ. (d) TD(0) methods have modular credit assignment mechanisms in generic cases. The red crosses indicate a lack of d-separation, whereas the green checkmarks do. 100

7.4 **How transfer tasks are generated.** We consider transfer problems where the optimal decision sequence of the transfer task differs from that of the training task by a single decision. As above, the transfer MDP and the training MDP differ in that the effect of action **B**; all other transitions remain the same. The agent must learn to choose action **D** instead of **C** while re-using other previously optimal decisions. . . . 104

7.5 **How the decision mechanisms change during transfer.** Shown the three states of the decision sequence. The optimal last decision must change from action C (purple) to action D (green). CVS modifies its bids independently. The bids for PPOF are coupled together across decision mechanisms and across time. 105

7.6 **Transfer problems involving triplets of decisions.** For each task topology (leftmost column) we have a training task, labeled (a) and three independent transfer tasks, labeled (b,c,d). Each transfer task is a different way to modify the training task's MDP. CVS consistently exhibits higher sample efficiency than both PPO and PPOF showing that dynamic modularity correlates with more efficient transfer. Notably the gap between CVS and the other methods in the bottom-right (e.g. 13.9x more efficient than PPO) is so wide that we had to extend the chart width. We set the convergence time as the first time after which the return deviates by no more than $\varepsilon = 0.01$ from the optimal return, 0.8, for 30 epochs of training. Shown are runs across ten seeds. . 106

7.7 **Modularity and forgetting.** The optimal solutions for tasks (a) and (b) involve a disjoint set of decisions: $A \to C$ for task (a) and $B \to D$ for task (b). We first train on task (a), then transfer from (a) to (b), then transfer back from (b) to (a). The purpose of this experiment is to test whether dynamic modularity improve the agent's ability to preserve optimal behavior on a previous task after having trained to convergence on a different task in a different context. While both CVS and PPO have similar sample efficiency when initially training on task (a), CVS is more than ten times more sample efficient than PPO when transferring back from (b) to (a), suggesting that PPO "forgot" the optimal behavior for task (a) when training on task (b), which is not the kind of forgetting we want in learning agents. 108

A.1 Implicit differentiation appears to create a stronger dependence among the slots. This figure shows what reconstruction looks if we train and evaluate with 12 slots, then re-render the reconstruction by deleting slots one at a time. When there are still many other slots as context, for both vanilla and implicit SLATE, deleting a slot corresponds to a clean deletion of the corresponding object in the reconstruction, as shown in the inset that highlights what the rendering looks like if we render with eight slots and seven slots. However, as we remove more slots, implicit SLATE generates less coherent compositions than vanilla SLATE, as shown when we render with only one to three slots. What causes this discrepancy is also an open question for future work. 137

A.2 Despite our work pushing the optimization performance for a state-of-the-art model in object-centric learning (Tab. 2.3), and despite implicit slot attention producing similarly intuitive *predicted* segmentation masks as vanilla slot attention (Fig. 2.8), there appears to be a qualitative difference between the *attention* maps of implicit slate and those of vanilla slate. As this figure shows, the attention masks for vanilla SLATE appear to be more localized to each object, the attention masks for implicit SLATE appear to be more smeared out. One observation is that in some cases implicit SLATE appears to attend not only to the object but also its shadow, as circled in green. However, in other cases the attention maps appear to be smeared in other ways that may attend to a shadow that could possibly happen, but not necessarily a shadow in the given scene. What causes this discrepancy is open question for future work. 138

A.3 **Comparing different orders of Neumann approximation.** We sought to understand how the different orders of Neumann approximation affected performance. We observe that the 1st order approximation still largely performs the best, likely because adding more terms to the series expansion requires backpropagating through more iterations of slot attention, which was the problem we had sought to avoid in the first place. However, most approximations still perform better than the vanilla model with the same number of forward iterations. 139

A.4 **Qualitative visualizations without gradient clipping: implicit.** This figure shows qualitative visualizations of implicit SLATE's reconstructions and attention masks when trained without gradient clipping. Compared to Fig. A.5, implicit SLATE's reconstructions matches the ground truth much more closely, and its masks are more coherent. Vanilla SLATE's masks are much noisier, and become degenerate in later stages of training as its Jacobian norm explodes. 140
A.5 **Qualitative visualizations without gradient clipping: vanilla.** Compared to Fig. A.4, vanilla SLATE's masks are much noisier, and become degenerate in the later stages of training as its Jacobian norm explodes, whereas implicit SLATE's reconstructions matches the ground truth much more closely, and its masks are more coherent. 141

B.1 Qualitative results on building a structure from the dataset in [164]. The input is an "action image," which depicts how an action intervenes on the state by raising a block in the air. OP3 is trained to predict the steady-state outcome of dropping the block. We see how OP3 is able to accurately and consistently predict the steady state effect, successively capturing the effect of inertial dynamics (gravity) and interactions with other objects. 150
B.2 We show a demonstration of a rollout for the dataset from [164]. The first four columns show inference iterations (refinement steps) on the single input image, while the last column shows the predicted results using the dynamics module on the learnt hidden states. The bottom 5 rows show the subimages of each entity at each iteration, demonstrating how the model is able to capture individual objects, and the dynamics afterwards. Notice that OP3 only predicts a change in the yellow block while leaving the other latents unaffected. This is a desriable property for dynamics models that operate on scenes with multiple objects. . . . 151
B.3 Two-dimensional (left) and three-dimensional (right) visualization of attention values where colors correspond to different latents. The blocks are shown as the green squares in the 2D visualizatio; picking anywhere within the square automatically picks the block up. The black dots with color crosses denote the computed `pick_xy` for a given h_k. We see that although the individual values are noisy, the means provide good estimates of valid pick locations. In the right plot we see that attention values for all objects are mostly 0, except in the locations corresponding to the objects (purple and red). 152

C.1 An example of solving a task in the robogym rearrange environment used in this chapter. 158
C.2 The original Robogym rearrange setup 158
C.3 The performance of our method as the number of initialized clusters and batches from the training set used to construct the graph, and the number of slots are varied. 160

C.4 **Varying interaction horizon.** The performance of the NF (b) and MPC (c) baselines compared to NCS (d, reproduced from Fig. C.5) and the random baseline (a) on *robogym-rearrange* as we vary the interaction horizon (as a multiple of the minimum steps needed to complete the task). *Note that the scale of the y-axis is not the same.* While a longer horizon improves performance, NCS still achieves at least 50x better accuracy with an interaction horizon multiplier of 1 than the performance obtained by increasing the interaction horizon multiplier for the model-based baselines to 8. . . . 161
C.5 **Stress testing NCS** This figure shows the performance of NCS on *robogym-rearrange* as we vary the amount of noise added to the actions (left) and vary the interaction horizon, defined as a multiple of the minimum steps needed to complete the task (right). 161

D.1 **Numerical math task.** We compare our learner with the RNN baseline. As a sanity check, we also compare with a version of our learner which has a hardcoded controller (HCC) and a learner which has hardcoded modules (HCF) (in which case the controller is restricted to select windows of 3 with an operator in the middle). All models perform well on the training set. Only our method and its HCC, HCF modifications generalize to the testing and extrapolation set. The RNN requires 10 times more data to generalize to the testing and extrapolation set. For **(b, c)** we only show accuracy on the expressions with the maximum length of those added so far to the curriculum. "1e3" and "1e4" correspond to the order of magnitude of the number of samples in the dataset, of which 70% are used for training. 10, 50, and 90 percentiles are shown over 6 runs. 168
D.2 **Variations:** The minimum number of reducers and translators that can solve the multilingual math problems is 1 and m respectively, where m is the number of languages. This is on an extrapolation task, which has more terms *and* different language pairs. (a, b): Four reducers and zero translators (red) is a pathological choice of modules that causes CRL to overfit, but it does not when translators are provided. (c) In the non-pathological cases, regardless of the number of modules, the learner metareasons about the resources it has to customize its computation to the problem. 10, 50, and 90 percentiles are shown over 6 runs. 168
D.3 Extrapolation 169
D.4 **Multilingual Arithmetic Execution Traces** 170

E.1 **The *Duality* environment.** 178
E.2 **The *Two Rooms* environment.** 179
E.3 **The *Mental Rotation* environment.** 180

F.1 This figure shows the computation graph of $\mathbb{L}$ across one credit assignment update. Inputs to the credit assignment mechanism are shaded. A modular credit assignment mechanism (shown with blue edges) is equivalent to showing the gradients δ_t as conditionally independent, as shown by the plate notation labeled with T. Dynamic modularity at iteration $i-1$ is equivalent to showing that the functions $\mathtt{f}^{k,i}$ are inside the plate labeled with N. Then because the `UPDATE` operation, shown with yellow edges, operates only within the plate labeled with N, the updated functions $\mathtt{f}^{k,i+1}$ are also conditionally independent given $(\mathbf{x}, \mathtt{f})$. 189

F.2 This figure shows part of the computational graph within Π for policy gradient methods. Conditioning on $\mathbf{x}$ implies we condition on the lightly shaded nodes. $\sum_k b_t^k$ is the shared hidden variable that renders $\delta_1, ..., \delta_T$ not d-separated. 191

F.3 This figure shows part of the computational graph within Π for TD($n > 1$) methods. Conditioning on $\mathbf{x}$ implies we condition on the lightly shaded nodes. $\sum_t r_t$ is the shared hidden variable that renders $\delta_1, ..., \delta_T$ not d-separated. 191

F.4 This figure shows part of the computational graph within Π for on-policy and off-policy TD(0) methods. Conditioning on $(\mathbf{x}, \mathtt{f})$ implies we condition on the lightly shaded nodes. For on-policy methods such as CVS and SARSA, the hidden variable would be $\max_k b_{t+1}^k$ for CVS and the bid corresponding to the decision mechanism that was sampled through ε-greedy for SARSA. The figure shows $\max_k b_{t+1}^k$ for concreteness. For off-policy methods such as Q-learning, the bids b_{t+1} are computed from s_{t+1} and $\mathtt{f}$, both of which we condition on. In both cases, the hidden variable is only parent to one of the δ_t's, and thus the $\delta_1, ..., \delta_T$ remain d-separated. 192

List of Tables

2.1 Complexity 17
2.2 Different instantiations of implicit differentiation 20
2.3 Quantitative metrics for image reconstruction through the slot bottleneck. . . . 22

3.1 Accuracy (%) of block tower builds by the SAVP baseline, the O2P2 oracle, and our approach. O2P2 uses image segmentations whereas OP3 uses only raw images as input. 38
3.2 Accuracy (%) of multi-step planning for building block towers. (xy) means `(pick_xy, place_xy)` action space while (entity) means `(entity_id, place_xy)` action space. . 38

4.1 This table compares NCS with various baselines in the complete and partial evaluation settings of *block-rearrange* and *robogym-rearrange*. The methods were trained on 4 objects and evaluated on generalizing to 4, 5, 6, and 7 objects. We report the fractional success rate, with a standard error computed over 10 seeds. 56

B.1 Accuracy of ablations. The no weight sharing model did not converge during training. 153

C.1 **Hyperparameters for training dSLATE** These hyperparameters are almost identical to those found in Singh, Deng, and Ahn [289, Fig. 7], but because dSLATE operates on video demonstrations rather than static images, we changed some hyperparameters to save memory cost. We changed the batch size from 50 to 32, the number of transformer layers and heads from 8 to 4, the number of slot attention iterations from 7 to 3 without observing a significant change in performance. Because each video in the experience buffer contains four objects, we used five slots, one more than the number of objects, following the convention used in Van Steenkiste et al. [317] and Veerapaneni et al. [319]. 155
C.2 **Hyperparameters for constructing the transition graph with NCS**. This table shows the distance metrics we use for the `isolate`, `cluster`, and `bind` functions described in 4.4. For *block-rearrange* we use the SA attention mask α as the state s, and for *robogym-rearrange* we use the action-dependent part of the SA slot $\boldsymbol{\lambda}^s$ as the state s. 156
C.3 **Number of clusters used for constructing the nodes of the transition graph.** 156

D.1 Numerical Arithmetic Dataset 163
D.2 Multilingual Arithmetic Dataset 164

E.1 **Societal Decision-Making.** This table specifies the **agent**, **environment**, **objective**, and **problem** at both the **global** and **local** levels of abstraction in the societal-decision-making framework. 173

Acknowledgments

The Ph.D. has been a formative period of my life in both my personal and professional development. I am grateful to my mentors, colleagues, friends, and family with whom I had the privilege to share this journey.

I want to thank my committee members, Alison Gopnik, Jonathan Ragan-Kelley, Sergey Levine, and Tom Griffiths for their support in my critical Ph.D. milestones. I would like to thank my advisors who have opened my mind to, and guided me in exploring, new dimensions and depths of human intellect I was not previously aware of.

I thank my advisor Sergey Levine for challenging me to think critically and clearly. Even when I first met him at NeurIPS 2016, when I was applying to Ph.D. programs, he challenged me to question basic premises I had not examined in my research on object-centric learning. I thank Sergey for teaching me the value of clarity and precision in thinking and communication, and of embracing the iterative nature of producing creative work. Sergey also gave me the privilege to co-lead Berkeley's outreach program for connecting underrepresented undergraduates to graduate mentors in AI. He has challenged me to break the cognitive barriers that limited my intellectual and professional growth. Growth is not comfortable; I am extremely thankful to Sergey for pushing me past the point of discomfort, for not picking me up when I fall, and thus giving me the opportunity to learn the confidence and skills for overcoming challenges in the rest of my life.

I thank my advisor Tom Griffiths for his steadfast support and guidance in my intellectual and professional development. Tom has taught me how to structure my thinking and clearly formulate research questions. He showed me that effectively doing so was not a kind of black magic that only comes inscrutably through the right talent or experience, but that there was a clear process for how to do so, this process was possible to study, and that it was worth studying and practicing. During my stay at Princeton in January 2020, Tom not only connected me with crucial collaborators but also proactively offered to fund my housing for that month. It is examples like these that show that Tom truly embodies his motto of "people over papers;" I am extremely thankful for everything he has gone out of his way to do on behalf of me to support me to achieve my goals.

I would not have reached this point without the help of all my professional mentors who believed in me and gave me opportunities to accelerate my journey in research. I want to thank Honglak Lee, Yuting Zhang, and Ruben Villegas who gave me my first machine learning research opportunity the summer after my sophomore year; Will Whitney, Tejas Kulkarni, Tomer Ullman, Antonio Torralba, and Josh Tenenbaum who gave me the opportunity to do my first full-scale research projects and who supported me in applying to grad school; Sjoerd van Steenkiste, Klaus Greff, and Jürgen Schmidhuber for mentoring me in my first project after undergrad; Amy Zhang and Franziska Meier for mentoring me at my internship at Meta; Murray Shanahan and Antoine Miech for mentoring me at my internship at DeepMind. I am lucky to continue to consider them mentors both inside and outside of my professional life.

I want to thank all my collaborators for many exciting research projects: Abhishek Gupta, Sophia Sanborn, David Bourgin, Fred Callaway, Erin Grant, Paul Krueger, Falk Lieder, Jason

Peng, Grace Zhang, Pieter Abbeel, Fenglu Hong, Thanard Kurutach, Rishi Veerapeneni, J.D. Co-Reyes, Michael Janner, Chelsea Finn, Jiajun Wu, Josh Tenenbaum, Sid Kaushik, Matt Weinberg, Yash Sharma, Yilun Du, Arnaud Fickinger, Natasha Jaques, Samyak Parajuli, Nick Rhinehart, Glen Berseth, Stuart Russell, Alyssa Li Dayan, Franziska Meier, Amy Zhang, Bhishma Dedhia, and Jake Snell. I especially highlight my undergraduate collaborators Rishi Veerapaneni and Sid Kaushik, whom I had the privilege to learn from and work with.

I want to thank the RAIL lab at Berkeley, the Cocosci lab at Princeton, and the broader BAIR and Berkeley EECS community who all made my Ph.D. experience fun. I want to thank Shirley Salanio, Jean Nguyen, and Angie Abbatecola for their help in making my Ph.D. experience smooth and enjoyable. I am grateful to have been supported by the Berkeley EECS Department Fellowship and the National Science Foundation (NSF) Graduate Research Fellowship Program while working on this thesis.

The Ph.D. is a lonely spiritual journey and I could not have accomplished this without the guidance and support of my friends and family. I want to thank my swim coaches Gaby, Rod, Andi, Adam, and Ken for challenging me to develop mental endurance and toughness at a young age. I want to thank all the deep relationships I have formed over this journey, which are too numerous to mention, and I hope to correct any omissions, but I would like to especially highlight Parsa Mahmoudieh, Thanard Kurutach, Frederik Ebert, Aravind Srinivas, Nalini Singh, Abhishek Gupta, Allan Jabri, Fangyin Wei, Erin Grant, Erin Reynolds, Kelvin Xu, Jason Peng, Ashvin Nair, Vitchyr Pong, Justin Fu, Rose Yin, Rose Wang, Michael Janner, Feras Saad, Jenny Huang, Kumar Krishna Agrawal, Cathy Chen, Jessy Lin, Pulkit Agrawal, Vickie Ye, Kristin Cheng, Sam Toyer, Olivia Watkins, Ajay Jain, Paras Jain, and Kaitlyn Hennacy, who all have made my Ph.D. journey especially meaningful. I want to thank them for believing in me when it was hard for me to believe in myself.

At Berkeley, I especially thank Parsa Mahoumdieh, Thanard Kurutach, and Michael Janner for their steadfast friendship through the most emotionally difficult and happy parts of the Ph.D. I consider them all role models in kindness, compassion, and clear thinking.

And most importantly, I would like to thank my family, for all the sacrifices they had made to help me on my journey and their love and presence no matter what happens. I thank my grandparents for their sacrifices that have given me opportunities to succeed. I thank my parents for motivating me, and supporting me, and keeping me focused. I thank my brother for showing me what true confidence and discipline looks like. This Ph.D. is your Ph.D.; we did it together.

Chapter 1

Introduction

If machines are to solve problems for us, then they need to automatically model and manipulate systems. A problem is simply the gap between an existing and desired state of a particular underlying system: solving a problem requires manipulating the system into a state that what we want, and knowing how to manipulate the system requires modeling the system.

But to a large extent, automatically modeling and manipulating systems is still something that only humans do. Computers have allowed us to manually encode how we model and manipulate systems via software, but we do not yet know how to enable machines to automatically determine *what* system to model and manipulate for solving a problem.

The very concept of a system is an abstraction over an agent's interface to the real world – the sensorimotor stream. What makes an abstraction of a system powerful is not whether it supports solving a specific problem but whether it can be reused for new problems that have not yet been encountered before. Therefore building machines that automatically model and manipulate systems implies building machines that learn reusable abstractions from sensorimotor experience. But we do not yet know how to do this.

The problem of automatically modeling and manipulating systems is usually formulated in a artificial intelligence (AI) as an agent-environment interaction loop, where an agent manipulates environment states by taking actions. A state is a system configuration underlying the environment, and an action is a choice of a transformation in the environment that changes the system to a different state. But the strength of AI capabilities have largely depended on whether or not humans have already provided the appropriate abstractions of system configurations and transformations to the agent.

When appropriate system abstractions are given, AI methods excel. Examples include modeling language [38] and simulating proteins [168], where words and amino acids are reusable abstractions that humans have already predefined. But when appropriate system abstractions are not given, methods often fail to be useful. Deep reinforcement learning (RL) is the primary class of machine learning methods for learning directly from the sensorimotor stream. Such methods can learn to solve fixed tasks, such as opening doors [197] or moving individual toys [6]. But they often fail at what I call **combinatorial generalization**: generalizing to new systematic variations in the agent-environment interaction [172].

Combinatorial generalization

The combinatorial space of possible real world systems motivates why combinatorial generalization is a good criterion for evaluating useful interaction with systems. Meadows [219] defines a **system** as a set of interrelated entities that produce behavior. We can unpack this definition by defining the space of **system behaviors** as the space of entity configurations and of system transformations. We can then extend this definition to the agent-environment interaction by defining the space of **system interactions** as the space of system behaviors in the environment and of choices the agent makes for which transformations to apply. Combinatorial generalization can thus be more specifically decomposed as generalizing over the space of possible combinations of entities, of transformations, and of choices.

As an evaluation criterion, combinatorial generalization enables us to test to what extent the agent represents the appropriate abstraction of an underlying system: train the agent to solve problems under a subset of possible system interactions and test the agent's ability to generalize to new problems under a disjoint subset of other system interactions. If the agent has represented entities, transformations, and choices in a way that corresponds to the structure of the actual system it is interacting with, then it should be able to generalize over the space of possible combinations of entities, of transformations, and of choices. One could argue that combinatorial generalization underlies science and engineering in general: in science we seek explanations that remain true, and in engineering we seek methods that remain robust, across a combinatorial space of varying factors. But combinatorial generalization also shows up in simple mundane problems.

Consider even the intuitive problem of object rearrangement: from visual input, rearrange a set of objects to new locations over a tabletop. If the agent trains only four objects, can it generalize to rearranging six objects? This requires it to have learned the appropriate abstractions of entities, transformations, and choices. Specifically, it requires (1) representing objects as independent entities in a way that can be recombined within larger sets of other entities, (2) representing the process of moving an object from one location to another as an independent transformation than can be recombined within longer sequences of transformations, and (3) representing the process of selecting these transformations as independent choices that can be recombined within longer sequences of choices for solving new problems. Though this kind of generalization may seen trivial for humans, it is challenging for current RL methods. Even seemingly simple problems, such as arithmetic with varying length expressions or numerical values, still challenge even the best state-of-the-art systems [39].

Shortcomings of monolithic approaches

Why has combinatorial generalization been difficult for existing deep RL methods? One possible reason is that these methods represent the agent-environment interaction monolithically: they encode, process, and act upon the distribution of possible environment configurations and transformations, as well as agent choices, with the same neural network. This might not be unreasonable if the environment consists of a single entity, such that training data

can sufficiently cover the space of possible variations the agent would encounter. But it is generally infeasible to sufficiently cover the combinatorial space of *system* interactions. Without sufficient coverage, training neural networks to monolithically represent systems introduces spurious correlations from the co-presence of entities, transformations, or choices during training that may be broken when the agent encounters new problems, thereby limiting the agent's capacity for combinatorial generalization.

This suggests a perhaps straightforward conclusion that to enable combinatorial generalization we need to move beyond the monolithic approach for abstracting the sensorimotor interface and instead model and manipulate systems as they are: by building agents that represent discrete sets of entities, transformations, and choices that reflect the underlying systematic structure of the agent-environment interaction.

The trouble of course is that there no ground truth for what this structure should be. Even humans have revised our abstractions of reality for thousands of years. We can at least turn the problem into one we can study by testing whether the agent recovers human abstractions. But even so, the challenge is how to enable the agent to do so automatically without explicitly observing or receiving supervision on what the structure should be. If we choose not to directly provide or supervise this structure, then the other option is to enforce this structure to emerge through learning constraints. To enable the structure that emerges to be suitable for combinatorial generalization, such constraints must prevent spurious correlations among representations of entities, transformations, and choices. They must enforce some form of **independence** among these representations.

Enforcing independence in representations

Enforcing independence among representations is not trivial. Consider what this means for producing entity representations from images. Simply splitting a monolithic image encoder into one with multiple heads does not work, because fixing a number of heads prevents generalizing to scenes with more entities than what has been encountered in training. Even if this issue were somehow fixed, backpropagating with respect to a global loss function still trains each head to be co-dependent. It is also not clear that going the opposite extreme of the monolithic approach, by maximally splitting images into one representation per pixel, helps either. Such representations are known as **tokens** and are used to represent the fine-grained units of images and text – image patches and subwords. But splitting representations as tokens has the opposite problem of not capturing the dependence among different tokens that do refer to the same concept [310]. Representations of entities should be not as monolithic as representations of scenes but also not as fine-grained as representations of pixels.

Worse, it is not even clear how to formalize independence in the first place. The concept of statistical independence used in machine learning is a theoretical abstraction of an infinite sampling process, but the independence we are interested has nothing to do with repeated sampling. Rather, we are interested in the independence of the *descriptions* of the representations: for example, the entities of a system can still be described independently without referencing the descriptions of other entities, regardless of whether we observe multiple

samples of the system or not. The model of **algorithmic independence** [201, 294, 181] deals with independence of descriptions, but this model is also only of theoretical use because it is based on the uncomputable Kolmogorov complexity measure.

The obvious ways of factorizing monolithic networks do not work, and our mathematical descriptions of "independence" are loose descriptions of the property we want in our representations of entities, transformations, and choices. However, there is a different notion of independence that does not fit neatly in existing mathematical frameworks of machine learning, but does capture exactly the kind of independence we seek to enforce in our neural representations. It is the principle of **separation of concerns** that was invented for circuit and software design, a principle that has enabled billions of circuit components to be composed in different ways to make modern computers.

Proposal

How can we enable neural networks to create reusable abstractions of systems that support combinatorial generalization? I argue in this thesis that the answer might lie in how we have designed computers themselves. The transition from electronic circuits to software a century ago involved solving a very similar problem of enforcing independence among different circuit components. This led to the **digital abstraction**, an instance of the broader principle of **separation of concerns**. The digital abstraction was the crucial idea that allowed us to scale electronic circuits to general-purpose software.

The von Neumann architecture is the general-purpose circuit architecture that underlies the structure of modern software today. It is perhaps no accident that the three kinds signals processed by the von Neumann architecture – data, programs, and controls – correspond exactly to the entities, transformations, and choices that underpin the combinatorial space of system interactions that software is uniquely designed to generalize over. Given that we seek the same combinatorial generalization in our learning agents, this thesis proposes to consider the implications of drawing an analogy between electronic circuits and neural networks – the *neural* circuits underlying current AI systems. Perhaps the kinds of methods that enabled us to build programmed software from circuits for modeling and manipulating any system via programs can inform how we can build *learning software* from neural networks for automatically modeling and manipulating any system.

The point of view behind this thesis is that **contextual refinement**, the core implementational principle behind the digital abstraction, and separation of concerns more broadly, can be instantiated in neural networks to enable learning reusable abstractions of systems that support combinatorial generalization. This thesis presents examples of how this principle can be incorporated into neural networks to build reusable abstractions of entities, transformations and choices, thereby taking a step toward building machines that automatically model and manipulate systems as humans do.

1.1 Contextual refinement

Computers are such versatile tools for solving problems that it may be easy to forget that computers were not always universal machines, but began as highly specialized electronic circuits. The first machine that Alan Turing himself built was a circuit that could do no more than crack German codes during World War II [87]. The neural networks driving recent AI advances have been similarly specialized. Whether it is for classifying images, controlling robotic arms, or answering questions, neural networks have traditionally been trained for solving specific tasks just as electronic circuits were designed to solve specific tasks.

Yet in the past century, we have transformed specialized electronic circuits into universal machines that can be programmed to model and manipulate any system. How we generally represent system interactions exactly mirrors the flow of computation in modern computer architectures: system behavior can be represented as a factor graph unrolled through time, with nodes as entities, factors as transformations, and the connectivity between factors and nodes as choices an agent makes to apply transformations to entities [166]. Similarly, the flow computation can be represented as the same kind of factor graph – usually called a computation graph [55], with nodes as data, factors as programs, and connectivity as controls.

What makes writing software uniquely powerful is that the code we write for representing data, programs, and controls can be reused and combined with other code to create software of arbitrary complexity. This ability to reuse and combine representations of entities, transformations, and choices is also exactly the combinatorial generalization we seek for neural networks. But the generalization capabilities of software did not come by accident; we had to invent methods for transforming the continuous noisy voltages of electronic circuits into the discrete generalizable code of software. What were these methods exactly, and how did they enable the generalization capabilities of modern software?

The standard answer to this question would reference **separation of concerns** as the key design principle behind the various abstractions we invented for organizing circuit and software components into reusable modules. Precisely, encapsulating disparate information in separate modules reduces interdependencies, which enables modules to be reused in more contexts in combination with other modules. Hence, separation of concerns is the notion of independence that we have invented to achieve combinatorial generalization in software.

While this answer is sufficient in the context of computing systems, there is a difference between electronic circuits and neural networks that makes it only part of the answer in the context of learning systems. In computing systems, the representations of entities, transformations, and choices are designed, but in learning systems, these representations are learned. Knowing that encapsulation is the solution to combinatorial generalization is by itself not a sufficient guide if what should be encapsulated is not known beforehand.

What we need to understand is the mechanism by which encapsulation is achieved. If we can bake this mechanism into deep learning systems, then we can allow the learning signal to dictate what the representations should be and rely on this mechanism to enable the representations to be combinatorially composed. We can identify what this mechanism is by

considering how the **digital abstraction** is implemented. This is because turning voltages into bits was the defining leap that enabled the separation of concerns in every higher layer of the computing stack to be implemented precisely.

The digital abstraction was invented as a solution for enforcing precise specifications. Precise specifications enables any system component to be combined with any other system component as long as how they are combined meets their specifications. Like activations in neural networks, voltages in electronic circuits are continuously valued. For analog circuits, continuously valued inputs and outputs resulted in circuit components whose specifications could only be satisfied approximately, rather than precisely. This prevented arbitrary composability because errors would accumulate. The solution we invented was to design circuit components that actively corrected errors by amplifying marginally valid 0s and 1s, possibly corrupted by noise, back to solidly valid 0s and 1s [136].

The need to enforce precise specifications rules out circuits that implement only passive forward propagation [136]. This is why inventing active digital circuits from passive analog circuits was the defining leap that enabled combinatorial generalization in programmed software. Data, programs, and controls are implemented as voltages in the von Neumman circuit, yet thanks to the digital abstraction they can be effectively represented as binary strings that can be modified independently without affecting other representations.

But it is not just voltages that are refined toward their desired specification of binary values; we can understand this active error-correction as an instance of a broader class of mechanisms for **contextual refinement** that applies at every level of the computing stack beyond circuits. One of the main ways to enable different software components to be reused in combination of other components is through wrapper functions. Wrapper functions modify a given program's outputs or restrict a given program from operating on certain inputs, thereby refining the program's pre-defined behavior to match a desired specification that enables it to be newly combined with other programs. In all cases, this refinement is contextualized to the particular execution of the circuit or program. Refining a component's behavior to specification thereby implements encapsulation by enforcing the component to not leak any other information beyond what the specification calls for.

In the context of computing, contextual refinement is how we have implemented encapsulation, encapsulation is how we have instantiated separation of concerns, and separating concerns is how we have enable data, programs, and controls to be combined in novel ways. The key difference between computing and learning is that specifications in computing are designed, whereas specifications in learning are learned. This therefore suggests that we may be able to enable combinatorial generalization in neural networks by implementing mechanisms for contextual refinement, mechanisms that refine towards learned specifications rather than designed specifications. In the section that follows, I summarize the various ways I have explored instantiating the principle of contextual refinement in neural networks for representing entities, transformations, and choices.

1.2 Neural software abstractions

Our goal is to enable machines to automatically model and manipulate systems, and we have identified learning reusable abstractions for combinatorial generalization as the core bottleneck. Specifically the reusable abstractions we seek are representations of entities, transformations, and choices that can be independently recombined in new contexts. Machine learning lacks a useful mathematical framework for describing this notion of independence, and we have argued that naively factorizing monolithic networks does not effectively prevent spurious correlations. The transition from electronic circuits to software required solving a similar problem. The solution for achieving combinatorial generalization in computing was to implement the principle of separation of concerns via contextual refinement towards designed specifications. The question now is whether the method of contextual refinement towards *learned* specifications can enable neural networks to learn representations that can be similarly reused.

Problem

I investigate this question along three dimensions: entities, transformations, and choices.

1. **Entities:** Objects are the entities of the sensorimotor interface. Neural networks traditionally represent visual scenes either as a single monolithic vector representation or as a grid of token representations, one per small image patch. The problem is that monolithic representations do not afford reusing information about entities in new contexts, and token representations are too fine-grained to capture the coherence of objects. What we want instead is for a neural network to learn to infer a set of representations that encapsulate information about an individual object within a representation while removing information about other objects, thereby enabling these representations to be coherently recombined in new ways.

2. **Transformations:** In the broadest sense, transformations are functions. In the context of deep learning we consider the functions that blocks of neural network weights perform on their inputs. Neural networks traditionally are organized either as recurrent architectures that repeat the same function or as feedforward architectures that implement a fixed sequence of different functions. The problem is that forcing all transformations to either be implemented with the same repeated set of weights or the same sequence of weights either introduces spurious correlations among weights or requires massively wide layers that waste capacity. What we want instead is to learn a set of distinct functions that each implement a specific behavior and that can be reused a variable number of times in combination with each other.

3. **Choices:** Unlike entities and transformations, choices are fundamentally tied to the problem being solved, so instead of seeking zero-shot combinatorial generalization it makes more sense to seek efficient transfer to problems that require new combinations

of choices. Agents make choices via policies that map states to actions, and we define a **choice** as a the function that assigns a weight to a particular action, such as a softmax logit or a Q-value. Neural networks policies traditionally entangle all choice function in the same parameters of a monolithically parameterized policy or Q-function. The problem is that the existence of other actions may not be relevant to the choice for a particular action, but the monolithic policy imposes a dependence among the choice functions, which may not only be artificial and unnecessary but also harmful for efficient adaptation, as I show in Chatper 7. What we want instead is for the choice functions to learn via local credit assignment that isolates the modification of one choice function from modifying others.

The desired solution for all three cases involves decomposing a traditional monolithic representation into a discrete set of reusable representations that each encapsulate the information specific to that representation from information specific to other representations. We know from the design of circuits and software that encapsulation is implemented via contextual refinement towards a desired specification. In our context, the specifications of what the representations of entities, transformations, and choices would not be designed but learned. The open question therefore is whether the benefits of encapsulation – the ability to reuse representations in new contexts – can still be achieved by incorporating the *mechanism* of contextual refinement as a constraint on the neural network forward propagation, without specifying *what* the mechanism should refine to beforehand, leaving the target of refinement something that is learned via stochastic gradient descent.

Approach

To answer this question, I present three different ways of implementing contextual refinement in neural networks:

1. **Iterative clustering (for entities)** I frame the problem of inferring reusable representations of objects as a kind of latent clustering problem over latent visual tokens, where the parameters that describe a cluster are treated as the representation of an object. The contextual refinement mechanism is an iterative procedure that alternates between assigning tokens to clusters and updating the cluster parameters to better fit their assigned tokens. Like the expectation-maximization (EM) algorithm for Gaussian mixture models, this iterative process makes each cluster more encapsulated as it adjusts the parameters of each cluster to be more specific to its assigned subset of tokens. Whereas the assignment and update steps of the EM algorithm are analytical, the iterative process I use consists of neural networks that replace these two steps. Therefore, whereas the EM algorithm refines cluster parameter estimates closer to the maximum likelihood estimates, the contextual refinement mechanism I use refines towards representations that are best suited for some downstream task, like image reconstruction, video prediction, or object property classification.

2. **Partial modification (for transformations)** To learn specialized functions that can combined in new contexts, I adopt an approach that restricts a neural block from operating certain parts of its input. Contextual refinement in this context is simply the manual enforcement of the restriction itself. Like code wrappers in software, restricting a function to operate only part of its input leaves the function agnostic to the part that it ignores, enabling it to be reused in new contexts where different values of the ignored part are encountered. Unlike code wrappers, however, I do not specify what information should be ignored or not: that is, for input x, I decompose x as $x = [x_1, x_2]$, and specify that x_2 should be ignored but do not specify what the content of x_2 should be – the neural blocks are free to learn the representations for x_1 and x_2 that would enable the entire network to solve its task. Therefore, the refinement mechanism is a manually imposed restriction on the neural block's input, and the refinement target is a function behavior that is agnostic to the part of the input that is ignored.

3. **Incentivizing truthfulness (for choices)** To train choice functions via local credit assignment, I reframe the sequential decision problem not as a single agent optimization for a policy but as a multi-agent game for the choice functions. In particular, a A choice function's output is interpreted as an auction bids for its associated action to operate on a particular state. The auction winner at the previous time-step sells the state at the auction at the next time-step. The contextual refinement mechanism would thus be the auction mechanism that incentivizes equilibrium strategies of the choice functions to coincide with the optimal policy for the single agent formulation. I show that this can be achieved with the Vickrey auction as the auction mechanism, which incentivizes the choice functions to truthfully bid the optimal Q-value for their associated action. This truthfulness property enables each choice function to compute and update its Q-value independently of other choice functions, thereby encapsulating the representation of each choice function – the neural network weights for computing its bid – from one another. A choice function's optimal Q-value for a state is analogous to a real-world auction bidder's valuation of the auction good, but unlike real-world Vickrey auctions, the optimal Q-value is not known by the choice function beforehand and must be learned. Thus the refinement mechanism is the Vickrey incentive mechanism, and the refinement target for a particular choice function is its optimal Q-value of the state.

All three methods for contextual refinement depart from traditional deep learning in a particular way, which is that they implement a refinement mechanism during the forward execution of the neural network. This is similar to the difference between the active error correction performed by digital circuits and the passive forward propagation performed by analog circuits. One could say that the training of neural networks via stochastic gradient descent is a form of refinement as well, but it is *global* refinement that is a function of all the data the network trains on, rather than the *contextual* refinement specific to a particular input that this thesis proposes.

1.3 Outline and Contributions

This thesis is a compilation of six publications grouped across three main parts – entities, transformations, and choices. Together, they show how contextual refinement enables better combinatorial generalization in neural networks.

Part I: Entities shows that reusable representations of entities can be learned via the contextual refinement mechanism of iterative clustering.

- **Chapter 2** [**53**] presents a method for improving the optimization of neural networks that implement iterative clustering.
- **Chapter 3** [**319**] extends the iterative clustering algorithm from representing entities in static images to representing entities in dynamic videos.

Part II: Transformations shows that reusable representations of transformations can be learned via the contextual refinement mechanism of enforcing partial modification of inputs.

- **Chapter 4** [**56**] shows that training a dynamics model to ignore action-invariant features of objects enables us to construct a planning algorithm that composes transformations over object states that can be reused for different objects in different contexts. **Chapter 5** [**58**] shows that restricting neural blocks to operating over only parts of arithmetic expressions enables generalization to solving expressions of varying length.

Part III: Choices shows that transferable representations of choices functions can be learned via the contextual refinement mechanism of incentivizing truthful bidding

- **Chapter 6** [**54**] introduces the Vickrey auction mechanism as a method for aligning the Nash equilibrium of the auction to coincide with the optimal policy of the single agent sequential decision problem.
- **Chapter 7** [**55**] explains how choice functions trained via the Vickrey auction mechanism learn via local credit assignment.

Part IV: Conclusion concludes the thesis.

Part I

Entities

Chapter 2

Representing Static Entities

2.1 Introduction

Conventionally, neural network models implement feedforward computation, transforming the input x into the output z through a fixed series of operations corresponding to distinct layers as $z = f_N(f_{N-1}(...f_2(f_1(x))...))$. However, a range of more sophisticated models implement *iterative* computation with the network, typically formulated and motivated as some sort of optimization procedure where the correct answer is the fixed point of an iterative refinement $z = f_x(z)$ of an initial guess z_0. This includes diffusion models [293, 153, 297, 272], energy-based models [192, 78], deep equilibrium models [20], iterative amortized inference procedures [215, 214, 216], neural ordinary differential equations [60], meta-learning algorithms [93, 123, 15], and object-centric models [317, 319, 177, 85]. Such iterative refinement procedures may have a number of advantages over direct feedforward computation: they can serve to simplify the learned function (e.g., in the same way that a recursive program might be much simpler than an equivalent program implemented without recursion), introduce an inductive bias into the model that improves generalization, and break symmetries.

In this work, we consider the case of iterative refinement applied to representation learning of latent sets, which has primarily been applied to learning representations of objects from pixels [130, 317, 132, 129, 319, 208, 177, 356, 290]. The particular challenge of this setting is that invariance of set elements to permutation means there are many latent sets that serve as equally plausible explanations for the data. Iterative refinement is especially useful in this context because it breaks the symmetry among these explanations with the randomness of the initial guess z_0 rather than encoding the symmetry-breaking mechanism in the weights of the network, as do conventional methods that learn a direct feedforward mapping from observation to representation. The state-of-the-art of these iterative object-centric methods is the slot attention module from Locatello et al. [208], serving as the focus of this chapter.

Unfortunately, these methods have been notoriously difficult to train. Their nature as unsupervised representation learning methods means that we do not have ground-truth targets to supervise the outputs of each iteration of f, as in other settings (e.g. diffusion models).

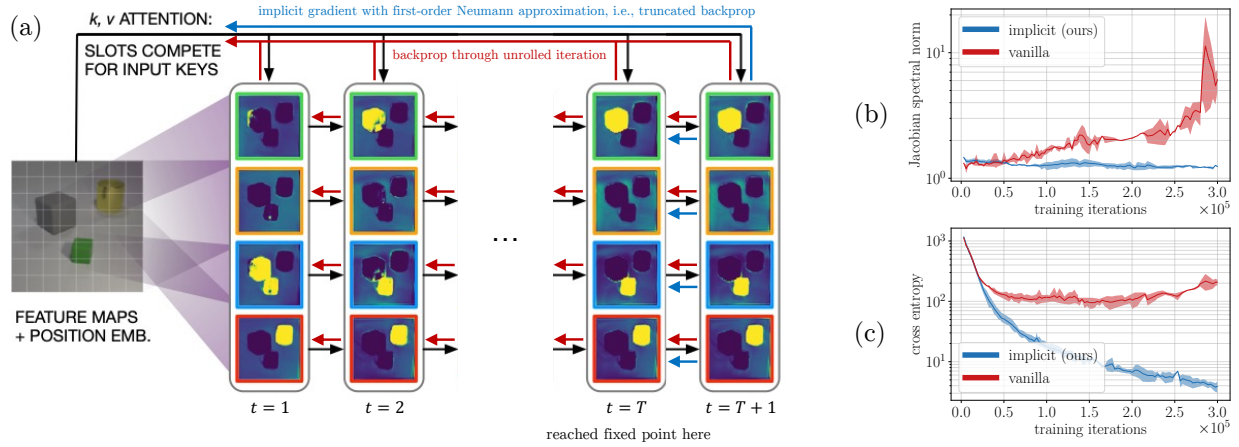


Figure 2.1: **Overview.** We address efficient training of iterative refinement methods for learning representations of latent sets, such as the slot attention model in (a), whose illustration is adapted from Locatello et al. [208]. Vanilla slot attention backpropagates gradients through the unrolled iterative refinement procedure, which leads to training instabilities as shown by the growing Jacobian spectral norm (b). Implicit slot attention uses the first-order Neumann approximation of the implicit gradient, which simply truncates the backpropagation, leading to substantially more effective training, as shown in (c).

As a result, prior works train these methods by differentiating through the unrolled iterations of f. Differentiating through this recurrence contributes to various training instabilities, as we can see by the growing spectral norm of f in Fig. 2.1b. Such instabilities result in sensitivity to hyperparameter choices (e.g., number of refinement steps) and have motivated adding optimization tricks such as gradient clipping, learning rate warm-up, and learning rate decay, all of which make such models more complex and harder to use, restrict the model from optimizing its learning objective fully, and only temporarily delay instabilities that still emerge in later stages of training.

To approach this problem, we observe that previous iterative refinement methods like slot attention have not taken full advantage of the fact that f can be viewed as a fixed point operation. Thus, f can be trained with **implicit differentiation** applied at the fixed point, without backpropagating gradients through the unrolled iterations [48, 80]. This chapter investigates how advances in implicit differentiation in neural models can be applied to improve the training of iterative refinement methods.

Our primary contribution is to propose implicit differentiation for training iterative refinement procedures, specifically slot attention, for learning representations of latent sets. First, we show that slot attention can be cast as a fixed point procedure that can be trained with implicit differentiation, resulting in what we call **implicit slot attention**. Second, we show on the latest state-of-the-art method of this kind, SLATE [290], that using the first-order Neumann approximation of the implicit gradient for the slot attention module

yields substantial improvement in optimization. Third, we show across three datasets that, compared to SLATE, our method for training achieves much lower validation loss in training, as well as lower Fréchet inception distance (FID) [150] and mean squared error (MSE) in image reconstruction. Fourth, our method also removes the need for gradient clipping, learning rate warmup, or tuning the number of iterations, while achieving lower space and time complexity in the backward pass, all with just one additional line of code. Fifth, when integrated with the original slot attention encoders and decoders from Locatello et al. [208], implicit differentiation substantially improves object property prediction and continues to predict intuitive segmentation masks as the vanilla slot attention.

2.2 Related Work

Much early work in artificial intelligence followed a paradigm of using an iterative learning procedure during execution time, whether it be a form of search, inference, or optimization [267]. These include early work on variational inference [70, 102] and energy-based models [192, 337, 338, 335, 336, 78], Hopfield networks [156], Boltzmann machines [4], and associative memory [180]. The rise of modern deep networks in the last decade shifted the method for computing solutions during execution time away from iterative procedures and towards producing the solution directly with a feedforward pass of a network. Recent works have started to shift the paradigm of execution back to combining the best of function approximation and iterative search, with neural networks parameterizing initializations [93], update rules [15, 130, 214], search heurstics [285, 162], and evaluation functions [78]. Our work concerns the optimization of neural networks as update rules for these iterative refinement procedures.

Our methodological novelty is the adaptation of implicit differentiation techniques for training iterative refinement procedures for representing of latent sets, where our key insight is that the iterative procedure used for symmetry-breaking reaches a fixed point, thus allowing implicit differentiation to be used. We are not aware that this has been done before, as current methods that perform iterative refinement for representing latent sets, often referred to as **object-centric learning** [130, 317, 132, 129, 319, 208, 177, 356, 290], all differentiate through the unrolled dynamics of the fixed point procedure, which as we show makes them difficult to train.

Our work draws upon innovations in implicit differentiation that have been applied in various other applications besides object-centric learning, such as embedded optimization layers [11, 5], neural ordinary differential equations [60], meta-learning [254], implicit neural representations [158], declarative layers [118], and transformers [20]. Although we evaluate various techniques for implicit differentiation, our results highlight the benefits of the simplest of these, which is that of using a truncated Neumann approximation [111, 105, 158, 284]. While this technique was used in Zoran et al. [356] without theoretical explanation, we propose implicit differentiation as an explanation for why this technique is beneficial. The closest works to ours is the concurrent work of Zhang et al. [354] which applies implicit differentiation

to predicting properties of set elements and finds similar benefits. Our approach generalizes their results to the unsupervised setting and scales to high dimensional outputs (e.g., images), taking a crucial step toward improving representation learning of latent sets.

2.3 Background

Our work builds on prior works on iterative refinement and implicit differentiation with deep networks.

Iterative refinement for inferring representations of latent sets Current work on inferring representations of latent sets is motivated by learning to represent objects – which we consider *independent* and *symmetric* entities – from perceptual input. Because mixture models are also defined to have *a priori* independent and symmetric mixture components, they have been the model of choice for representing entities: thus these iterative methods model each datapoint x^n (e.g. image) as a set of independent sensor measurements $x^{n,m}$ (e.g. pixels) which are posited as having been generated from a mixture model whose components represent the entities. Under a clustering lens, the problem reduces to finding the K groups of cluster parameters $\boldsymbol{\theta}^n := \{\theta^{n,k}\}_{k=1}^K$ and cluster assignments $\boldsymbol{\phi}^{n,m} := \{\phi^{n,m,k}\}_{k=1}^K$ that were responsible for the measurements $x^{n,m}$ of the datapoint x^n.

A network f breaks symmetry among components by alternately updating $\boldsymbol{\theta}^n$ and $\boldsymbol{\phi}^{n,m}$ starting from independent randomly initialized $\theta^{n,k}$'s. The slot attention module [208], e.g., computes $\boldsymbol{\theta}_{t+1}^n \leftarrow f\left(\boldsymbol{\theta}_t^n, x^n\right)$, where $\boldsymbol{\phi}^{n,m}$ is updated as an intermediate step inside f. The $\boldsymbol{\theta^n}$, called *slots*, serve as input to a downstream objective, e.g. image reconstruction, whose gradients are backpropagated through the unrolling of f. Earlier works applied this approach to binary images [130] and videos [317].

Implicit differentiation Implicit differentiation is a technique for computing the gradients of a function defined in terms of satisfying a joint condition on the input and output. For example, a fixed point operation f is defined to satisfy "find z such that $z = f(z, x)$" (or written as $z = f_x(z)$) rather than through an explicit parameterization of f. This fixed point z_* can be computed by simply repeatedly applying f or by using a black-box root-finding solver. Letting $f_{\mathbf{w}}$ be parameterized by weights $\mathbf{w}$, with input x and fixed point z_*, the implicit function theorem [48] enables us to directly compute the gradient of the loss ℓ with respect to $\mathbf{w}$, using only the output z_*:

$$\frac{\partial \ell}{\partial \mathbf{w}} = \underbrace{\frac{\partial \ell}{\partial z_*}\left(I - J_{f_{\mathbf{w}}}\left(z_*\right)\right)^{-1}}_{\mathbf{u}^\top} \frac{\partial f_{\mathbf{w}}\left(z_*, x\right)}{\partial \mathbf{w}}, \tag{2.1}$$

where $J_{f_{\mathbf{w}}}\left(z_*\right)$ is the Jacobian matrix of $f_{\mathbf{w}}$ evaluated at z_*. Compared to backpropagating through the unrolled iteration of f, which is just one of many choices of the solver, implicit

differentiation via Eq. 2.1 removes the cost of storing any intermediate results from the unrolled iteration. Deep equilibrium models (DEQ) [20] represent the class of functions f parameterized by neural networks, which have successfully been trained with implicit differentiation and empirically produce stable fixed points even though their convergence properties remains theoretically not well understood.

Much effort has been put into approximating the inverse-Jacobian term $\left(I - J_{f_{\mathbf{w}}}\left(z_*\right)\right)^{-1}$ which has $\mathcal{O}(n^3)$ complexity to compute. Using notation from Bai, Koltun, and Kolter [21], Pineda [249] and Almeida [10] propose to approximate the $\mathbf{u}^\top$ term in Eq. 2.1 as the fixed point of the linear system:

$$\mathbf{u}^\top = \mathbf{u}^\top J_{f_{\mathbf{w}}}\left(z_*\right) + \frac{\partial \ell}{\partial z_*}, \tag{2.2}$$

which can be solved with any black-box solver. However, in the context of applying implicit differentiation to neural networks, this approach has in practice required some expensive regularization to maintain stability [21], so Geng et al. [111], Fung et al. [105], Huang, Bai, and Kolter [158], and Shaban et al. [284] propose instead to approximate $\left(I - J_{f_{\mathbf{w}}}\left(z_*\right)\right)^{-1}$ with its Neumann series expansion, yielding

$$\frac{\partial \ell}{\partial \mathbf{w}} = \lim_{T \to \infty} \frac{\partial \ell}{\partial z_*} \sum_{i=0}^{T} J_{f_{\mathbf{w}}}\left(z_*\right)^i \frac{\partial f_{\mathbf{w}}\left(z_*, x\right)}{\partial \mathbf{w}}. \tag{2.3}$$

The first-order approximation ($T = 1$) amounts to applying f once to the fixed point z_* and differentiating through the resulting computation graph. This is not only cheap to compute and easy to implement, but has also been shown empirically [111] to have a regularizing effect on the spectral norm of $J_{f_{\mathbf{w}}}$ without sacrificing performance.

2.4 Implicit Iterative Refinement

Our main contribution is to propose treating iterative refinement algorithms used for inferring latent sets as fixed-point procedures, thereby motivating the use of implicit differentiation to train them.

Motivation

Our approach is inspired by the structural resemblance between iterative refinement for latent sets and the Expectation-Maximization algorithm [72], which also has been pointed out by Greff, Van Steenkiste, and Schmidhuber [130] and Locatello et al. [208]. If f were to update the components $\boldsymbol{\theta}^n$ and assignments $\boldsymbol{\phi}^{n,m}$ in a way that monotonically improves the evidence lower bound (ELBO) $\mathcal{L}^n$ on the log-likehood of each image x^n with respect to $\boldsymbol{\theta}^n$ and $\boldsymbol{\phi}^{n,m}$, then we know from Neal and Hinton [231] and Wu [333] that such an approach is a fixed point operation whose fixed point locally maximizes $\mathcal{L}^n$. With this interpretation, such

iterative refinement algorithms for inferring latent sets can be viewed as performing a nested optimization with three levels, where the weights of the slot attention module are optimized across images x^n, the components $\boldsymbol{\theta}^n$ are optimized per-image x^n but across measurements $x^{n,m}$, and the assignments $\boldsymbol{\phi}^{n,m}$ are optimized per-measurement $x^{n,m}$.

How these iterative refinement methods are implemented differs from optimizing the per-datapoint ELBO described above, however. Indeed, both Greff, Van Steenkiste, and Schmidhuber [130] and Locatello et al. [208] implemented ablations to their methods that do implement a variant of Expectation-Maximization on a well-defined mixture model, but found that replacing the update rule with a recurrent neural network empirically performs better at extracting representations.

Iterative refinement as a fixed point procedure

While in general we still lack the theory to understand whether and how the slots of slot attention optimize a well-defined objective like the ELBO, and thus have no theoretical guarantees that slot attention does converge to a fixed point, we empirically observe from its stable forward relative residuals (Fig. 2.3a) that it does appear to approximately converge to a fixed point. This suggests that slot attention can be understood as an instance of a DEQ that uses naive forward iteration to find a fixed point and backpropagates through the iteration to compute the gradient. The novel implication of this to inferring latent sets is that *any* root-finding solver and implicit gradient estimator can in principle be used to train slot attention, and we find that many combinations do train slot attention effectively in Fig. 2.3b. This insight is one of our main contributions, but in the next section we describe a particular combination that we empirically found effective.

Implicit slot attention

We propose **implicit slot attention**: a method for training the state-of-the-art slot attention module [208] with the simplest and most effective method that we have empirically found for approximating the implicit gradient, which is its first-order Neumann approximation (Eq. 2.3). It can be implemented by simply differentiating the computation graph of applying the slot attention update *once* to the fixed point $\boldsymbol{\theta}^n_*$, where $\boldsymbol{\theta}^n_*$ is computed by simply iterating the slot attention module forward as usual, but *without* the gradient tape, amounting to truncating the backpropagation. The time and space complexity of backpropagation for implicit slot attention compared to vanilla slot attention as a function of the number of slot attention iterations n, is shown in Table 2.1. While other methods for implicit differentiation are more complex to implement, this method requires only one additional line of code (Fig. 2.2).

Table 2.1: Complexity

	vanilla	implicit
time (forward)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
space (forward)	$\mathcal{O}(n)$	$\mathcal{O}(1)$
time (backward)	$\mathcal{O}(n)$	$\mathcal{O}(1)$
space (backward)	$\mathcal{O}(n)$	$\mathcal{O}(1)$.

```
def step(slots, k, v):
    # compute assignments given slots
    q = project_q(norm_slots(slots))
    k = k * (slot_size ** (-0.5))
    attn = F.softmax(torch.einsum('bkd,bqd->bkq', k, q), dim=-1)
    attn = attn / torch.sum(attn + epsilon, dim=-2, keepdim=True)
    # update slots given assignments
    updates = torch.einsum('bvq,bvd->bqd', attn, v)
    slots = gru(updates, slots)
    slots = slots + mlp(norm_mlp(slots))
    return slots
```

```
def iterate(f, x, num_iters):
    for _ in range(num_iters):
        x = f(x)
    return x

def forward(inputs, slots):
    inputs = norm_inputs(inputs)
    k, v  = project_k(inputs), project_v(inputs)
    slots = iterate(lambda z: step(z, k, v), slots, num_iterations)
    slots = step(slots.detach(), k, v)
    return slots
```

Figure 2.2: **Code.** The first order Neumann approximation to the implicit gradient adds only one additional line of Pytorch code [243] to the original forward function of slot attention, but yields substantial improvement of optimization. `attn` and `slots` correspond to ϕ and θ in the text respectively.

2.5 Experiments

Our main hypothesis is that iterative refinement methods for latent sets, specifically slot attention, can be trained as DEQs, and that consequently implicit differentiation would improve their optimization. We test this premise by replacing the backward pass of the slot attention module in the state-of-the-art SLATE [290] with the first-order Neumann approximation of the implicit gradient. This requires only one additional line of code to the original slot attention implementation (Fig.2.2) and in one instance improves reconstruction mean-squared error by almost 7x.

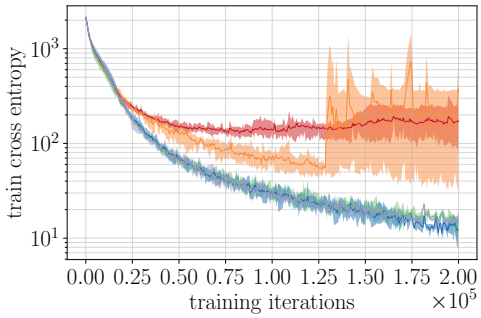
Experimental setup

We summarize the SLATE architecture, datasets we considered, and relevant implementation details.

SLATE SLATE uses a discrete VAE [255] to compress an input image into a grid of discrete tokens. These tokens index into a codebook of latent code-vectors, which, after applying a learned position encoding, serve as the input to the slot attention module. An Image GPT decoder [59] is trained with a cross-entropy loss to autoregressively reconstruct the latent code-vectors, using the outputted slots from slot attention as queries and the latent code-vectors as keys/values. Gradients are blocked from flowing between the discrete VAE and the rest of the network (i.e. the slot attention module and the Image GPT decoder), but the entire system is trained simultaneously.

Data We consider three datasets: CLEVR-Mirror [290], Shapestacks [135], and COCO-2017 [204], the former two of which were used in the original SLATE paper. We obtained CLEVR-Mirror directly from the SLATE authors and used a 70-15-15 split for training, validation, and testing. We pooled all the data variants of Shapestacks together as Singh,

(a) $\frac{\|f_{\mathbf{w}}(z,x)-z\|}{\|f_{\mathbf{w}}(z,x)\|}$



(b) Different solvers

Figure 2.3: **Can slot attention by trained as a fixed point operation with implicit differentiation?** (2.3a) The relative residual of the forward iteration of slot attention is close to zero, which motivates its treatment as a fixed point operation. (2.3b) The forward and backward computations of vanilla slot attention can be swapped out with different solvers, most of which result in the same improved optimization performance. Here we compare with *IB*, *BB*, *IN*, and *BN* variants of implicit differentiation across 4 seeds. Table 2.2 defines these acronyms.

Deng, and Ahn [290] did and used the original train-validation-test splits. The COCO-2017 dataset was downloaded from FiftyOne and used the original train-validation-test splits.

Implementation details In all of our experiments we used the same model and training hyperparameters as those used for the ShapeStacks experiment from Singh, Deng, and Ahn [290, Table 6], run on an A100 GPU. The only difference is in the image resolution (and consequently number of image tokens) because the largest image size we could train with was 96x96 due to computing constraints, rather than the 128x128 used for their CLEVR experiment. However, as we show for resolutions of both 96x96 and 64x64, the improvement in optimization appears to be hold across image resolutions.

For a clean comparison, we simply integrated the official SLATE implementation[1] with the solvers and backward gradient hook from the official implementation of deep equilibrium models[2]. Besides implementing logging and evaluation code, as well as the first-order Neumann approximation (Eq. 2.3), we did not change anything in the above two official implementations otherwise. The SLATE implementation serves as the baseline in all our experiments.

Training slot attention with implicit differentiation

The first test to conduct is to see whether slot attention can be trained with implicit differentiation at all. *In summary:* ***Test:*** *Swap the forward pass with a different black-box*

[1] `https://github.com/singhgautam/slate`
[2] `https://github.com/locuslab/deq`

Table 2.2: Different instantiations of implicit differentiation

abbrv.	forward computation	gradient estimation
IB	iteration	Broyden
BB	Broyden	Broyden
IN	iteration	Neumann approximation
BN	Broyden	Neumann approximation

solver, and train with implicit gradients computed via various gradient estimation methods for Eq. 2.1. ***Hypothesis:*** *There exists a (forward solver, implicit gradient estimator) that optimizes cross entropy no worse than backpropagating through unrolled inference.* ***Result:*** *Yes, such a pair exists, and in fact it enables significantly better optimization.*

To conduct this test, we compare vanilla slot attention with four variants trained with implicit differentiation, shown in Table 2.2, labeled *IB*, *BB*, *IN*, and *BN*. *B* stands for the Broyden solver, used by Bai, Kolter, and Koltun [20]. We test configurations where the Broyden solver was both used to find the fixed point of slot attention and used to estimate the implicit gradient with Eq. 2.2. *N* stands for the first-order Neumann approximation for computing the explicit gradient (Eq. 2.3). Our hypothesis would be tentatively refuted if the cross-entropy learning curves of none of these variants converges as efficiently as the baseline. We train these methods to model the CLEVR-Mirrors dataset with an image resolution of 64x64. The slot attention baseline unrolls the slot attention cell for seven iterations and we also limit the maximum number of Broyden iterations to seven.

Results and analysis Figure 2.3 shows that not only is it possible to train slot attention with implicit differentiation, but three out of four of the implicit differentiation configurations from Table 2.2 optimize significantly better than vanilla slot attention. Whereas vanilla slot attention plateaus at a cross entropy of around 130, the *BB*, *IN*, and *BN* variants all achieve a 10x lower cross entropy loss. Moreover, *BB*, *IN*, and *BN* all follow the same learning curve, suggesting that the optimization improvement is largely due to whether we use implicit differentiation or not, rather than the choice of how we implement the implicit differentiation. The *IB* variant is less stable than the others, and we hypothesize that this can be explained by the tendency for the fixed point of Eq. 2.2 to become increasingly hard to estimate, an issue discussed in Bai, Koltun, and Kolter [21].

Does this improvement generalize across datasets?

We now test whether this improved optimization holds across different datasets. If the improvement was simply due to implicit differentiation being serendipitously useful for 64x64 images CLEVR-Mirror, then we should expect implicit differentiation to not help for other

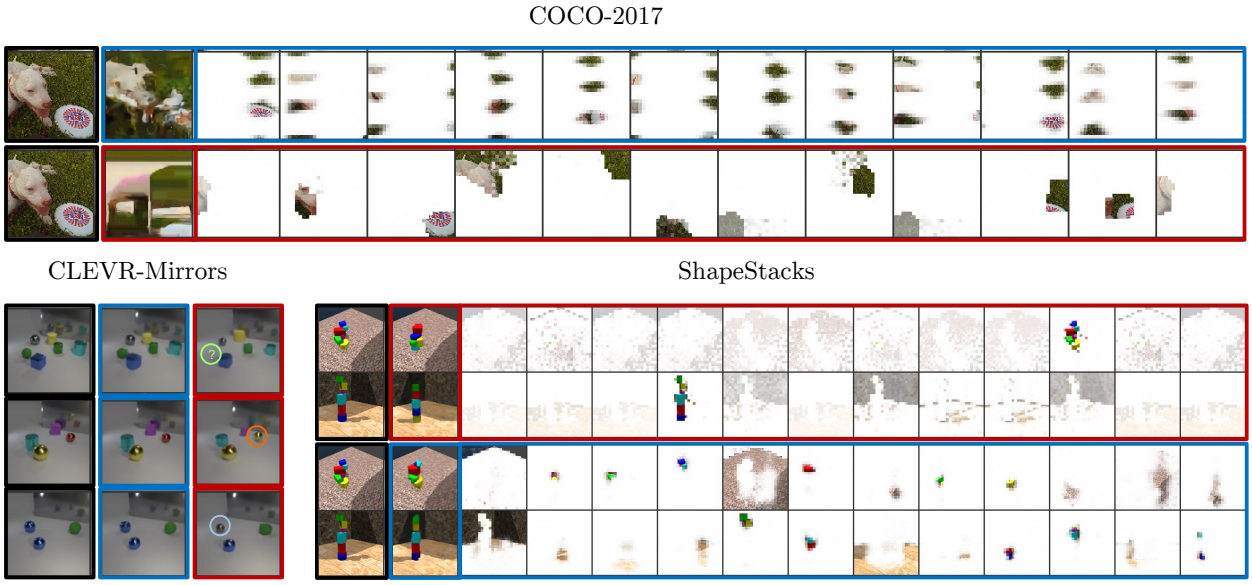


Figure 2.4: **Qualitative results.** Across three datasets, optimizing SLATE with implicit differentiation leads to improved image reconstructions through the slot bottleneck. Black borders indicate the ground truth image, blue border indicate our method, and red borders indicate vanilla SLATE. The rest of the panels visualize attention masks. In the CLEVR-Mirrors dataset, whereas vanilla SLATE misses objects, changes their size, or changes their color (indicated by the circles), implicit SLATE reconstructs the ground truth more faithfully.

datasets. We focus our attention on *IN* variant to conduct this test because it is the minimal modification to the baseline slot attention for gaining the benefits of implicit differentiation, as it requires adding only one line of code on top of the baseline slot attention implementation and does not require implementing any black-box solvers. We henceforth refer to the *IN* variant as **implicit slot attention** (and correspondingly implicit SLATE), as described in §2.4. *In summary:* ***Test:*** *Apply implicit slot attention to CLEVR-Mirror, ShapeStacks, and COCO with 96x96 image resolution.* ***Hypothesis:*** *Implicit slot attention significantly improves optimization across all three datasets.* ***Result:*** *Yes it does.*

Results and quantitative analysis Using the two primary metrics used in Singh, Deng, and Ahn [290], images generated by implicit SLATE achieve both lower pixel-wise mean-squared error and FID score [150]. The FID score was computed with the PyTorch-Ignite [98] library using the inception network from the PyTorch port of the FID official implementation. All methods were trained for 250k gradient steps. Table 2.3 compares the FID and MSE scores of the images that result from compressing the SLATE encoder's set of discrete tokens through the slot attention bottleneck, using Image-GPT to autoregressively re-generate these image tokens one by one, and using the discrete VAE decoder to render the generated image tokens. Implicit differentiation significantly improves the quantitative image reconstruction

Table 2.3: Quantitative metrics for image reconstruction through the slot bottleneck.

Data	Implicit	Vanilla
CLEVR (FID)	**22.19**	25.89
CLEVR (MSE)	**10.66**	67.04
COCO (FID)	**127.79**	147.48
COCO (MSE)	**1659.15**	1821.75
ShapeStacks (FID)	**34.2**	34.76
ShapeStacks (MSE)	**108.67**	312.14

metrics of SLATE across the test sets of CLEVR-Mirrors, Shapestacks, and COCO. In the case of MSE for CLEVR, this is almost a 7x improvement.

Qualitative analysis The higher quantitiatve metrics also translate into better quality reconstructions on the test set, as shown in Figure 2.4. For CLEVR-Mirrors, vanilla SLATE sometimes drops or changes the appearance of objects, even in simple scenes with three objects. In contrast, the generations produced from implicit SLATE match the ground truth very closely. For Shapestacks, implicit SLATE consistently segments the scene into constituent objects. This is sometimes the case with vanilla SLATE on the training and validation set as well, but we observed for both of the seeds we ran for the final evaluation that vanilla SLATE produced degenerated attention maps where one slot captures the entire foreground, and the background is divided among the other slots. The visual complexity of the COCO dataset is much higher than either CLEVR-Mirrors and Shapestacks, and the reconstructions on the COCO dataset are quite poor, for both SLATE's discrete VAE and consequently for the reconstruction through the slot bottleneck. This highlights the gap that still exists between using the state-of-the-art in object-centric learning out-of-the-box and what the community may want these methods to do. The attention masks for both vanilla and implicit SLATE furthermore do not appear to correspond consistently to coherent objects in COCO but rather patches on the image that do not immediately seem to match with our human intuition of what constitutes a visual entity.

Can we simplify the need for optimization tricks?

To further understand the benefits of implicit differentiation, we then ask whether it stabilizes the training of slot attention without the need for optimization tricks like learning rate decay, gradient clipping, and learning warmup. *In summary:* ***Test:*** *Remove learning rate decay, gradient clipping, and learning warmup each from vanilla SLATE and our method.* ***Hypothesis:*** *Our method should not require these tricks to optimize stably.* ***Result:*** *No, it generally does not require these tricks.*

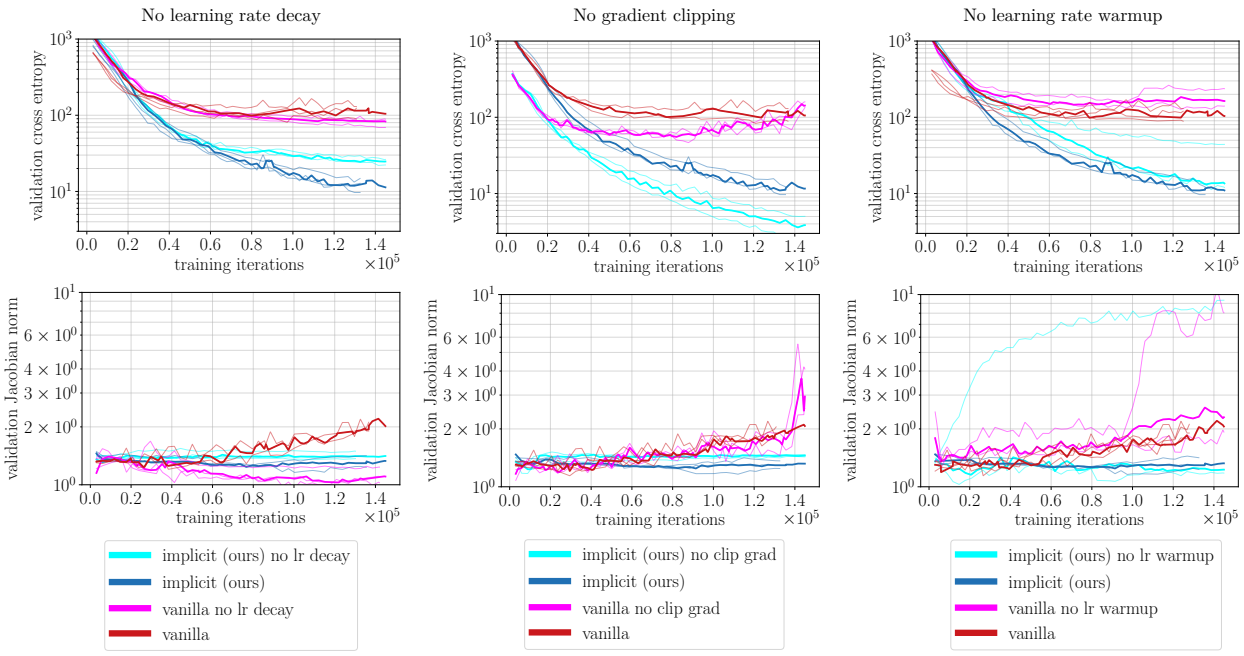
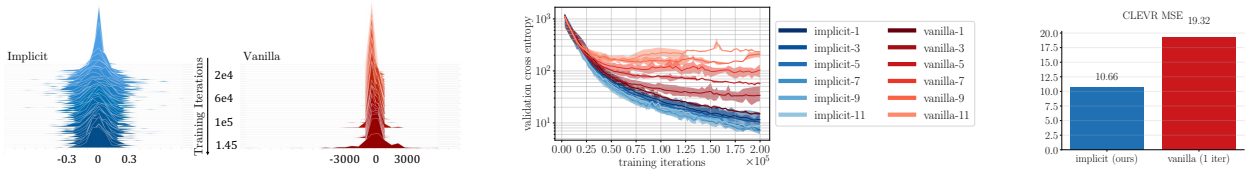


Figure 2.5: **Does implicit differentiation remove the need for various optimization tricks?** We ablate three heuristically-motivated optimization tricks from both vanilla SLATE and our method, and show that for two out of the three, removing the optimization trick quantitatively hurts the vanilla model but not the implicit model. Whereas removing gradient clipping and learning rate warmup causes vanilla SLATE's training to become unstable, as indicated by the growth of the Jacobian norm of the slot attention cell, our method trains significantly more stably and can take advantage of the larger gradient steps.

Fig. 2.5 shows that the ease and stability of training correlates with the spectral norm of the Jacobian of the slot attention cell. Decaying the learning rate regularizes the Jacobian norm from exploding, but it also hurts optimization performance for both our method and vanilla SLATE, as expected. When we remove gradient clipping the Jacobian norm of vanilla SLATE explodes, as do its gradients (Fig. 2.6a), whereas both stay stable for our method. This translates into a qualitative drop in performance for vanilla SLATE (Fig. A.5) but not for implicit SLATE (Fig. A.4). Lastly, removing learning rate warmup also consistently makes vanilla SLATE's training unstable, whereas it only affects the stability of our method for one out of three seeds. Finally, Fig. 2.6b shows that implicit slot attention is not sensitive to the number of iterations with which to iterate the slot attention cell, whereas vanilla slot attention is, with more iterations being harder to train.



(a) No need for gradient clipping. (b) No sensitivity to hyperparameters. (c) c.f. 1 iteration

Figure 2.6: (a) Without gradient clipping, our implicit differentiation technique keeps gradients small while backpropagating through the unrolled iterations causes gradients to explode. (b) Training with implicit differentiation also is not sensitive to the number of iterations with which to iterate the slot attention cell. (c) Using one iteration for vanilla slot attention trains as stably, but reconstructs more poorly, than implicit slate.

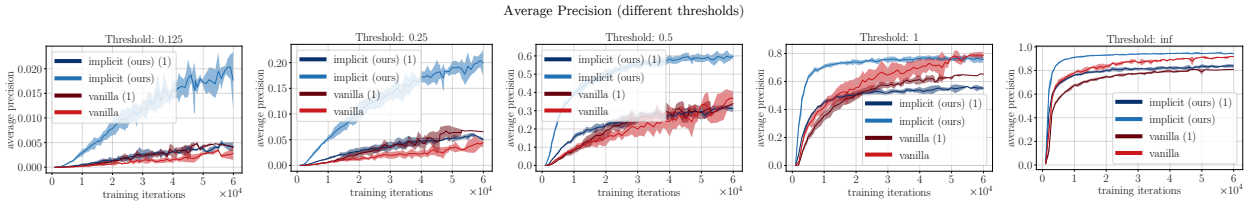


Figure 2.7: We outperform vanilla slot attention on object property prediction.

Does this mean that iterating slot attention for one iteration is enough?

One might be tempted to interpret Fig. 2.6b to suggest that fewer iterations of vanilla slot attention is sufficient to improve performance. This is not necessarily the case: although vanilla slot attention with one iteration trains more stably than its seven iteration counterpart, it still achieves 2x worse reconstruction MSE than implicit slot attention with seven iterations on CLEVR (Fig. 2.6c).

Futhermore, Fig. 2.7 shows that using only a single iteration is not enough to improve optimization of vanilla slot attention for the object property prediction task used by Locatello et al. [208]. We directly modified the released code from Locatello et al. [208], which used three iterations, to use implicit differentiation. Implicit slot attention can instead scale to as many forward iterations as needed.

Does implicit slot attention still produce intuitive masks with a different architecture?

We sought to check whether implicit differentiation still preserves the quality of the segmentation masks produced by the original slot attention architecture by Locatello et al. [208], which uses a spatial broadcast decoder [327] rather than a transformer decoder as SLATE

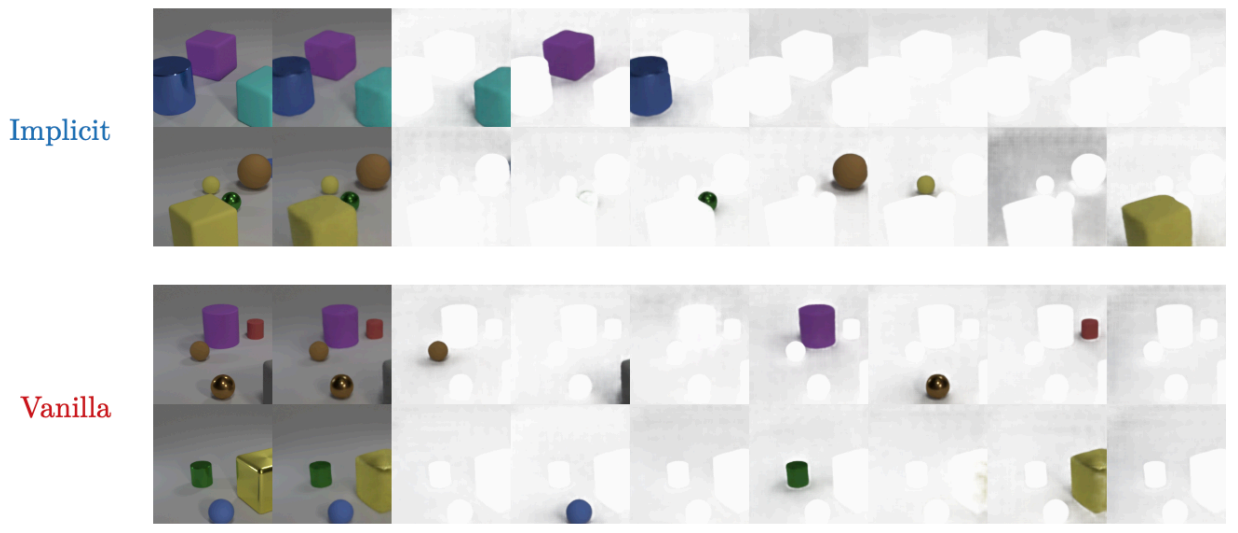


Figure 2.8: Implicit differentiation preserves the quality of predicted segmentations from Locatello et al. [208].

does. It indeed does (Fig. 2.8), suggesting that our findings are not specific to SLATE but apply to slot attention more broadly.

2.6 Discussion

The connection we made in this chapter between slot attention and deep equilibrium models also highlights various other properties about iterative refinement procedures for inferring latent sets that suggest connections to other areas of research that are worth theoretically developing in the future. First is the connection to the literature on fast weights [276, 17, 159]: interpreting slots as parameters that are modified during the inner optimization during execution time may give us novel formulation for how to represent and update fast weight memories. Second is the connection to the literature on meta-learning [275, 93, 312, 15]: interpreting slots as solutions to an inner optimization problem during execution time may give us a novel perspective on perception as itself a learning process. Third is the connection to causality [245, 248]: interpreting the independently generated and symmetrically processed slots as parameterizing independent causal mechanisms [139] may give us a novel approach for learning to represent causal models within a neural scaffolding. Fourth is the connection to dynamical systems [233]: interpreting slots as a set of attractor basins may offer a novel theory of how the error-correcting properties of discrete representations emerge from continuous ones. These different fields have their own conceptual and implementation tools that could potentially improve our understanding of how to build better iterative refinement algorithms and inform how objects could potentially be represented in the mind.

Conclusion We have proposed implicit differentiation for training iterative refinement procedures for inferring representations of latent sets. Our results show clear signal that implicit differentiation can offer a significant optimization improvement over backpropagating through the unrolled iteration of slot attention, and potentially any other iterative refinement algorithm, with lower space and time complexity and only one additional line of code. Because it is so simple to apply implicit differentiation to any fixed point algorithm, we hope our work inspires future work to leverage tools developed for implicit differentiation for improving learning representations of latent sets and iterative refinement methods more broadly.

Chapter 3

Representing Dynamic Entities

3.1 Introduction

A powerful tool for modeling the complexity of the physical world is to frame this complexity as the composition of simpler entities and processes. For example, the study of classical mechanics in terms of macroscopic objects and a small set of laws governing their motion has enabled not only an explanation of natural phenomena like apples falling from trees but the invention of structures that never before existed in human history, such as skyscrapers. Paradoxically, the creative *variation* of such physical constructions in human society is due in part to the *uniformity* with which human models of physical laws apply to the literal building blocks that comprise such structures – the reuse of the same simpler models that apply to primitive entities and their relations in different ways obviates the need, and cost, of designing custom solutions from scratch for each construction instance.

The challenge of scaling the generalization abilities of learning robots follows a similar characteristic to the challenges of modeling physical phenomena: the complexity of the task space may scale combinatorially with the configurations and number of objects, but if all scene instances share the same set of objects that follow the same physical laws, then transforming the problem of modeling scenes into a problem of modeling objects and the local physical processes that govern their interactions may provide a significant benefit in generalizing to solving novel physical tasks the learner has not encountered before. This is the central hypothesis of this chapter.

We test this hypothesis by defining models for perceiving and predicting raw observations that are themselves compositions of simpler functions that operate locally on *entities* rather than globally on scenes. Importantly, the symmetry that all objects follow the same physical laws enables us to define these learnable entity-centric functions to take as input argument a variable that represents a generic entity, the specific instantiations of which are all processed by the same function. We use the term *entity abstraction* to refer to the abstraction barrier that isolates the abstract *variable*, which the entity-centric function is defined with respect to, from its concrete *instantiation*, which contains information about the appearance and

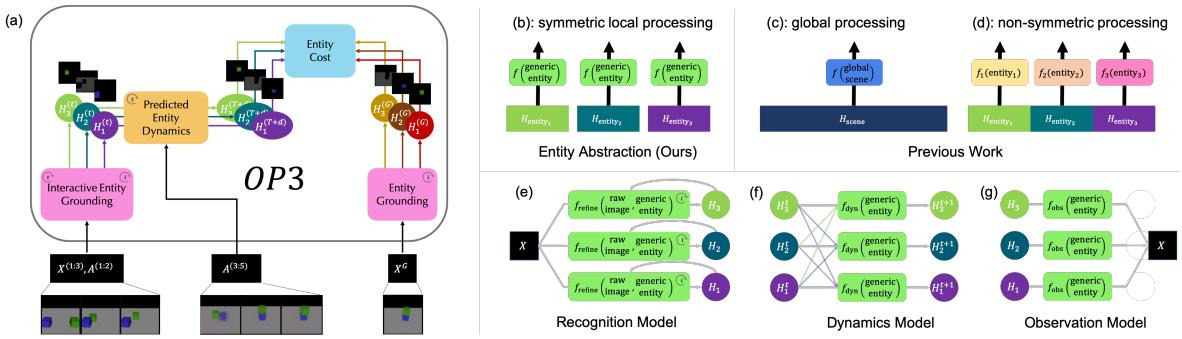


Figure 3.1: **OP3**. **(a)** OP3 can infer a set of entity variables $H_{1:K}^{(T)}$ from a series of interactions (interactive entity grounding) or a single image (entity grounding). OP3 rollouts predict the future entity states $H_{1:K}^{(T+d)}$ given a sequence of actions $a^{(T:T+d)}$. We evaluate these rollouts during planning by scoring these predictions against inferred goal entity-states $H_k^{(G)}$. **(b)** OP3 enforces the **entity abstraction**, factorizing the latent state into *local* entity states, each of which are symmetrically processed with the same function that takes in a *generic entity* as an argument. In contrast, prior work either **(c)** process a global latent state [145] or **(d)** assume a fixed set of entities processed in a permutation-sensitive manner [94, 184, 340, 326]. **(e-g)** Enforcing the entity-abstraction on modeling the **(f)** dynamics and **(g)** observation distributions of a POMDP, and on the **(e) interactive inference** procedure for grounding the entity variables in raw visual observations. Actions are not shown to reduce clutter.

dynamics of an object that modulates the function's behavior.

Defining the observation and dynamic models of a model-based reinforcement learner as neural network functions of abstract entity variables allows for symbolic computation in the space of entities, but the key challenge for realizing this is to ground the values of these variables in the world from raw visual observations. Fortunately, the language of partially observable Markov decision processes (POMDP) enables us to represent these entity variables as latent random state variables in a state-factorized POMDP, thereby transforming the variable binding problem into an inference problem with which we can build upon state-of-the-art techniques in amortized iterative variational inference [214, 215, 131] to use temporal continuity and interactive feedback to infer the posterior distribution of the entity variables given a sequence of observations and actions.

We present a framework for *object-centric perception, prediction, and planning* (OP3), a model-based reinforcement learner that predicts and plans over entity variables inferred via an *interactive inference* algorithm from raw visual observations. Empirically OP3 learns to discover and bind information about actual objects in the environment to these entity variables *without any supervision on what these variables should correspond to.* As all computation within the entity-centric function is *local in scope* with respect to its input entity, the process of modeling the dynamics or appearance of each object is *protected* from the computations involved in modeling other objects, which allows OP3 to generalize to modeling a variable number of objects in a variety of contexts with no re-training.

Contributions: Our conceptual contribution is the use of entity abstraction to integrate graphical models, symbolic computation, and neural networks in a model-based reinforcement learning (RL) agent. This is enabled by our technical contribution: defining models as the composition of locally-scoped entity-centric functions and the interactive inference algorithm for grounding the abstract entity variables in raw visual observations without any supervision on object identity. Empirically, we find that OP3 achieves two to three times greater accuracy than state of the art video prediction models in solving novel single and multi-step block stacking tasks.

3.2 Related Work

Representation learning for visual model-based reinforcement learning: Prior works have proposed learning video prediction models [330, 73, 194, 94] to improve exploration [234] and planning [95] in RL. However, such works and others that represent the scene with a single representation vector [145, 352, 225, 235] may be susceptible to the binding problem [128, 262] and must rely on data to learn that the same object in two different contexts can be modeled similarly. But processing a disentangled latent state with a single function [329, 62, 185, 184, 114] or processing each disentangled factor in a permutation-sensitive manner [194, 341, 184] (1) assumes a fixed number of entities that cannot be dynamically adjusted for generalizing to more objects than in training and (2) has no constraints to enforce that multiple instances of the same entity in the scene be modeled in the same way. For generalization, often the particular arrangement of objects in a scene does not matter so much as what is constant across scenes – properties of individual objects and inter-object relationships – which the inductive biases of these prior works do not capture. The entity abstraction in OP3 enforces symmetric processing of entity representations, thereby overcoming the limitations of these prior works.

Unsupervised grounding of abstract entity variables in concrete objects: Prior works that model entities and their interactions often pre-specify the identity of the entities [57, 27, 146, 163, 230, 24, 7], provide additional supervision [113, 149, 323, 344], or provide additional specification such as segmentations [164], crops [100], or a simulator [334, 172]. Those that do not assume such additional information often factorize the entire scene into pixel-level entities [271, 347, 79], which do not model objects as coherent wholes. None of these works solve the problem of grounding the entities in raw observation, which is crucial for autonomous learning and interaction. OP3 builds upon recently proposed ideas in grounding entity representations via inference on a symmetrically factorized generative model of static [128, 130, 131] and dynamic [317] scenes, whose advantage over other methods for grounding [355, 88, 41, 182, 326] is the ability to refine the grounding with new information. In contrast to other methods for binding in neural networks [200, 171, 292, 318], formulating inference as a mechanism for variable binding allows us to model uncertainty in the values of the variables.

	OP3	COBRA	Transporter	C-SWM
		Observation Model		
Fully symmetric?	Yes	Yes	N/A	N/A
Models segmentations?	Yes	Yes	No	No
		Dynamics Model		
Fully symmetric?	Yes	Yes	Yes	Yes
Models entity interactions?	**Explicitly**	No	Implicitly	Explicitly
Models stochastic transitions?	Yes	No	No	No
		Entity Grounding		
Method for entity disambiguation	**Iterative inference**	Autoregressive attention	Encoder forward pass	Encoder forward pass
Method for breaking symmetry	**Samples random noise**	Autoregressive attention	Associates content with representation index	Associates content with representation index
Method for temporal consistency	**HMM-like filtering update**	N/A	Associates content with representation index	Associates content with representation index
Updates grounding over time?	Yes	No	No	No
		Training		
Model training loss	**Full ELBO for dynamic latent variable model**	ELBO for observation model, pixel MSE for dynamics model	Pixel MSE	Contrastive loss between latents predicted from dynamics model and latents inferred from encoder
Exploration objective?	No	Yes	Yes	No

Figure 3.2: **Comparison with other methods.** Unlike other methods, OP3 is a fully probabilistic factorized dynamic latent variable model, giving it several desirable properties. First, OP3 is naturally suited for combinatorial generalization [28] because it enforces that local properties are invariant to changes in global structure. Because every learnable component of the OP3 operates symmetrically on each entity, including the mechanism that disambiguates entities itself (c.f. COBRA, which uses a learned autoregressive network to disambiguates entities, and Transporter and C-SWMs, which use a forward pass of a convolutional encoder for the global scene, rather than each entity), the weights of OP3 are invariant to changes in the number of instances of an entity, as well as the number of entities in the scene. Second, OP3's recurrent structure makes it straightforward to enforce spatiotemporal consistency, object permanence, and refine the grounding of its entity representations over time with new information. In contrast, COBRA, Transporter, and C-SWMs all model single-step dynamics and do not contain mechanisms for establishing a correspondence between the entity representations predicted from the previous timestep with the entity representations inferred at the current timestep.

Comparison with similar work: The closest three works to OP3 are the Transporter [184], COBRA [326], and C-SWMs [176]. The Transporter enforces a sparsity bias to learn object keypoints, each represented as a feature vector at a pixel location, and the method's focus on keypoints has the advantage of enabling long-term object tracking, modeling articulated composite bodies such as joints, and scaling to dozens of objects. C-SWMs learn entity representations using a contrastive loss, which has the advantage of overcoming the difficulty in attending to small but relevant features as well as the large model capacity requirements usually characteristic of the pixel reconstructive loss. COBRA uses the autoregressive attention-based MONet [41] architecture to obtain entity representations, which has the advantage of being more computationally efficient and stable to train. Unlike works such as [130, 88, 41, 131] that infer entity representations from static scenes, these works represent complementary approaches to OP3 (Figure 3.2) for representing dynamic scenes.

Symmetric processing of entities – processing each entity representation with the same function, as OP3 does with its observation, dynamics, and refinement networks – enforces the invariance that local properties are invariant to changes in global structure because it prevents the processing of one entity from being affected by other entities. How symmetric the process is for obtaining these entity representations from visual observation affects how straightforward it is to directly transfer models of a single entity across different global contexts, such as in modeling multiple instances of the same entity in the scene in a similar way or in generalizing to modeling different numbers of objects than in training. OP3 can exhibit this type of zero-shot transfer because the learnable components of its refinement process are fully symmetric across entities, which prevents OP3 from overfitting to the global structure of the scene. In contrast, the KeyNet encoder of the Transporter and the CNN-encoder of C-SWMs associate the *content* of the entity representation with the *index* of that entity in a global representation vector (Figure 3.1d), and this permutation-sensitive mapping entangles the encoding of an entity with the global structure of the scene. COBRA lies in between: it uses a learnable autoregressive attention network to infer segmentation masks, which entangles local object segmentations with global structure but may provide a useful bias for attending to salient objects, and symmetrically encodes entity representations given these masks[1].

As a recurrent probabilistic dynamic latent variable model, OP3 can refine the grounding of its entity representations with new information from raw observations by simply applying a belief update similar to that used in filtering for hidden Markov models. The Transporter, COBRA, and C-SWMs all do not have mechanisms for updating the belief of the entity representations with information from subsequent image frames. Without recurrent structure, such methods rely on the assumption that a single forward pass of the encoder on a static image is sufficient to disambiguate objects, but this assumption is not generally true: objects can pop in and out of occlusion and what constitutes an object depends temporal cues, especially in real world settings. Recurrent structure is built into the OP3 inference update (Appendix 4), enabling OP3 to model object permanence under occlusion and refine its object representations with new information in modeling real world videos (Figure 3.7).

3.3 Problem Formulation

Let x^* denote a physical scene and $h^*_{1:K}$ denote the objects in the scene. Let X and A be random variables for the image observation of the scene x^* and the agent's actions respectively. In contrast to prior works [145] that use a single latent variable to represent the state of the scene, we use a set of latent random variables $H_{1:K}$ to represent the state of the objects $h^*_{1:K}$. We use the term *object* to refer to h^*_k, which is part of the physical world, and the term *entity* to refer to H_k, which is part of our model of the physical world. The generative distribution

[1] A discussion of the advantages and disadvantages of using an attention-based entity disambiguation method, which MONet and COBRA use, versus an iterative refinement method, which IODINE [131] and OP3 use, is discussed in Greff et al. [131].

of observations $X^{(0:T)}$ and latent entities $H_{1:K}^{(0:T)}$ from taking T actions $a^{(0:T-1)}$ is modeled as:

$$p\left(X^{(0:T)}, H_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right) = p\left(H_{1:K}^{(0)}\right) \prod_{t=1}^{T} p\left(H_{1:K}^{(t)} \;\middle|\; H_{1:K}^{(t-1)}, a^{(t-1)}\right) \prod_{t=0}^{T} p\left(X^{(t)} \;\middle|\; H_{1:K}^{(t)}\right) \tag{3.1}$$

where $p(X^{(t)} \mid H_{1:K}^{(t)})$ and $p(H_{1:K}^{(t)} \mid H_{1:K}^{(t-1)}, A^{(t-1)})$ are the observation and dynamics distribution respectively shared across all timesteps t. Our goal is to build a model that, from simply observing raw observations of random interactions, can generalize to solve novel compositional object manipulation problems that the learner was never trained to do, such as building various block towers during test time from only training to predict how blocks fall during training time.

When all tasks follow the same dynamics we can achieve such generalization with a planning algorithm if given a sequence of actions we could compute $p(X^{(T+1:T+d)} \mid X^{(0:T)}, A^{(0:T+d-1)})$, the posterior predictive distribution of observations d steps into the future. Approximating this predictive distribution can be cast as a variational inference problem (Appdx. B.2) for learning the parameters of an approximate observation distribution $\mathcal{G}(X^{(t)} \mid H_{1:K}^{(t)})$, dynamics distribution $\mathcal{D}(H_{1:K}^{(t)} \mid H_{1:K}^{(t-1)}, A^{(t-1)})$, and a time-factorized recognition distribution $\mathcal{Q}(H_{1:K}^{(t)} \mid H_{1:K}^{(t-1)}, X^{(t)}, A^{(t-1)})$ that maximize the evidence lower bound (ELBO), given by $\mathcal{L} = \sum_{t=0}^{T} \mathcal{L}_{\mathrm{r}}^{(t)} - \mathcal{L}_{\mathrm{c}}^{(t)}$, where

$$\mathcal{L}_{\mathrm{r}}^{t} = \mathbb{E}_{h_{1:K}^{t} \sim q\left(H_{1:K}^{t} \mid h_{1:K}^{0:t-1}, x^{1:t}, a^{0:t-1}\right)} \left[\log \mathcal{G}\left(x^{t} \mid h_{1:K}^{t}\right)\right]$$
$$\mathcal{L}_{\mathrm{c}}^{t} = \mathbb{E}_{h_{1:K}^{t-1} \sim q\left(H_{1:K}^{t-1} \mid h_{1:K}^{1:t-2}, x^{1:t-1}, a^{0:t-2}\right)} \left[D_{KL}\left(\mathcal{Q}\left(H_{1:K}^{t} \mid h_{1:K}^{t-1}, x^{t}, a^{t-1}\right) || \mathcal{D}\left(H_{1:K}^{t} \mid h_{1:K}^{t-1}, a^{t-1}\right)\right)\right].$$

The ELBO pushes $\mathcal{Q}$ to produce states of the entities $H_{1:K}$ that contain information useful for not only reconstructing the observations via $\mathcal{G}$ in $\mathcal{L}_{\mathrm{r}}^{(t)}$ but also for predicting the entities' future states via $\mathcal{D}$ in $\mathcal{L}_{\mathrm{c}}^{(t)}$. Sec. 3.4 will next offer our method for incorporating entity abstraction into modeling the generative distribution and optimizing the ELBO.

3.4 Object-Centric Perception, Prediction, and Planning (OP3)

The *entity abstraction* is derived from an assumption about symmetry: that the problem of modeling a dynamic scene of multiple entities can be reduced to the problem of (1) modeling a single entity and its interactions with an *entity-centric* function and (2) applying this function to every entity in the scene. Our choice to represent a scene as a set of entities exposes an avenue for directly encoding such a prior about symmetry that would otherwise not be straightforward with a global state representation.

As shown in Fig. 3.1, a function F that respects the entity abstraction requires two ingredients. The first ingredient (Sec. 3.4) is that $F(H_{1:K})$ is expressed in part as the higher-order operation $\texttt{map}(f, H_{1:K})$ that broadcasts the same entity-centric function $f(H_k)$ to every entity variable H_k. This yields the benefit of automatically transferring learned knowledge

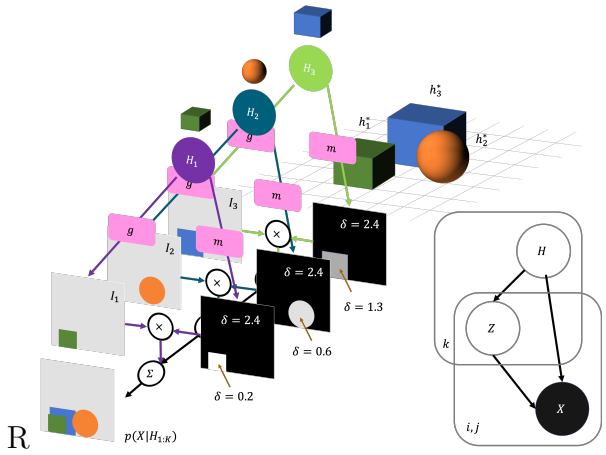


Figure 3.3: **(a)** The observation model $\mathcal{G}$ models an observation image as a composition of sub-images weighted by segmentation masks. The shades of gray in the masks indicate the depth δ from the camera of the object that the sub-image depicts. **(b)** The graphical model of the generative model of observations, where k indexes the entity, and i, j indexes the pixel. Z is the indicator variable that signifies whether an object's depth at a pixel is the closest to the camera.

for modeling an individual entity to all entities in the scene rather than learn such symmetry from data. As f is a function that takes in a single generic entity variable H_k as argument, the second ingredient (Sec. 3.4) should be a mechanism that binds information from the raw observation X about a particular object h_k^* to the variable H_k.

Entity Abstraction in the Observation and Dynamics Models

The functions of interest in model-based RL are the observation and dynamics models $\mathcal{G}$ and $\mathcal{D}$ with which we seek to approximate the data-generating distribution in equation 3.1.

Observation Model: The observation model $\mathcal{G}(X \mid H_{1:K})$ approximates the distribution $p(X \mid H_{1:K})$, which models how the observation X is caused by the combination of entities $H_{1:K}$. We enforce the entity abstraction in $\mathcal{G}$ (in Fig. 3.1g) by applying the same entity-centric function $g(X \mid H_k)$ to each entity H_k, which we can implement using a mixture model at each pixel (i, j):

$$\mathcal{G}\left(X_{(ij)} \mid H_{1:K}\right) = \sum_{k=1}^{K} m_{(ij)}\left(H_k\right) \cdot g\left(X_{(ij)} \mid H_k\right), \tag{3.2}$$

where g computes the mixture components that model how each individual entity H_k is independently generated, combined via mixture weights m that model the entities' relative depth from the camera, the derivation of which is in Appdx. B.1.

Dynamics Model: The dynamics model $\mathcal{D}(H'_{1:K} \mid H_{1:K}, A)$ approximates the distribution $p(H'_{1:K} \mid H_{1:K}, A)$, which models how an action A intervenes on the entities $H_{1:K}$ to produce their future values $H'_{1:K}$. We enforce the entity abstraction in $\mathcal{D}$ (in Fig. 3.1f) by applying the same entity-centric function $d(H'_k \mid H_k, H_{[\neq k]}, A)$ to each entity H_k, which reduces the

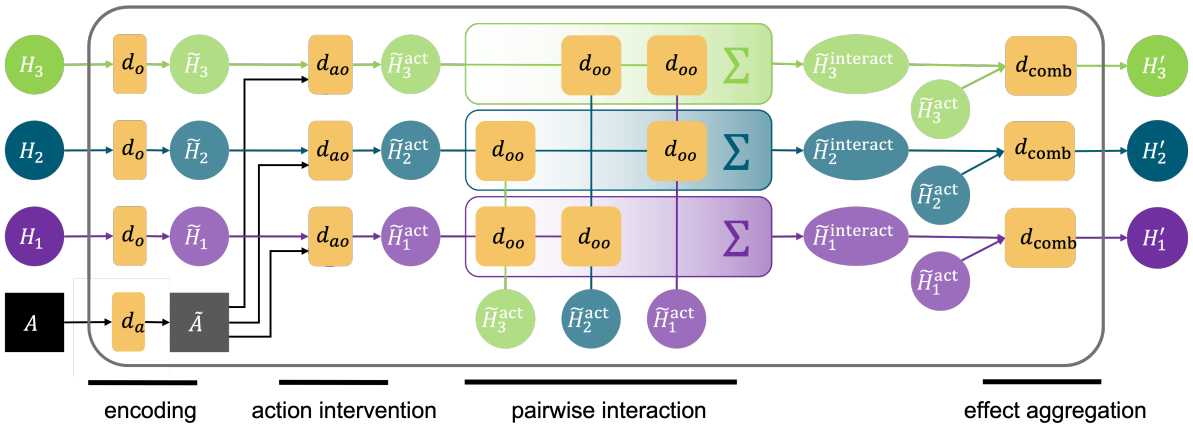


Figure 3.4: The **dynamics model** $\mathscr{D}$ models the time evolution of every object by symmetrically applying the function $\mathscr{d}$ to each object. For a given object, d models the individual dynamics of that object (d_o), embeds the action vector (d_a), computes the action's effect on that object (d_{ao}), computes each of the other objects' effect on that object (d_{oo}), and aggregates these effects together (d_{comb}).

problem of modeling how an action affects a scene with a combinatorially large space of object configurations to the problem of simply modeling how an action affects a single *generic* entity H_k and its interactions with the list of other entities $H_{[\neq k]}$. Modeling the action as an finer-grained intervention on a *single* entity rather than the entire scene is a benefit of using local representations of entities rather than global representations of scenes.

However, at this point we still have to model the combinatorially large space of *interactions* that a single entity could participate in. Therefore, we can further enforce a *pairwise* entity abstraction on $\mathscr{d}$ by applying the same *pairwise* function $\mathscr{d}_{oo}(H_k, H_i)$ to each entity pair (H_k, H_i), for $i \in [\neq k]$. Omitting the action to reduce clutter (the full form is written in Appdx. B.6), the structure of the $\mathscr{D}$ therefore follows this form:

$$\mathscr{D}\left(H'_{1:K} \mid H_{1:K}\right) = \prod_{k=1}^{K} \mathscr{d}\left(H'_k \mid H_k, H_k^{\text{interact}}\right), \text{ where } H_k^{\text{interact}} = \sum_{i \neq k}^{K} d_{oo}\left(H_i, H_k\right). \tag{3.3}$$

The entity abstraction therefore provides the flexibility to scale to modeling a variable number of objects by solely learning a function $\mathscr{d}$ that operates on a single generic entity and a function $\mathscr{d}_{oo}$ that operates on a single generic entity pair, both of which can be re-used for across all entity instances.

Interactive Inference for Binding Object Properties to Latent Variables

For the observation and dynamics models to operate from raw pixels hinges on the ability to bind the properties of specific physical objects $h^*_{1:K}$ to the entity variables $H_{1:K}$. For latent variable models, we frame this variable binding problem as an inference problem: binding

information about $h^*_{1:K}$ to $H_{1:K}$ can be cast as a problem of inferring the parameters of $p(H^{(0:T)} \mid x^{(0:T)}, a^{(0:T-1)})$, the posterior distribution of $H_{1:K}$ given a sequence of interactions. Maximizing the ELBO in Sec. 3.3 offers a method for learning the parameters of the observation and dynamics models while simultaneously learning an approximation to the posterior $q(H^{(0:T)} \mid x^{(0:T)}, a^{(0:T-1)}) = \prod_{t=0}^{T} \mathcal{Q}(H^{(t)}_{1:K} \mid H^{(t-1)}_{1:K}, x^{(t)}, a^{(t)})$, which we have chosen to factorize into a per-timestep recognition distribution $\mathcal{Q}$ shared across timesteps. We also choose to enforce the entity abstraction on the process that computes the recognition distribution $\mathcal{Q}$ (in Fig. 3.1e) by decomposing it into a recognition distribution q applied to each entity:

$$\mathcal{Q}\left(H^{(t)}_{1:K} \mid h^{(t-1)}_{1:K}, x^{(t)}, a^{(t)}\right) = \prod_{k=1}^{K} q\left(H^{(t)}_{k} \mid h^{(t-1)}_{k}, x^{(t)}, a^{(t)}\right). \tag{3.4}$$

Whereas a neural network encoder is often used to approximate the posterior [145, 340, 184], a single forward pass that computes q in parallel for each entity is insufficient to break the symmetry for dividing responsibility of modeling different objects among the entity variables [353] because the entities do not have the opportunity to communicate about which part of the scene they are representing.

We therefore adopt an *iterative inference* approach [214] to compute the recognition distribution $\mathcal{Q}$, which has been shown to break symmetry among modeling objects in static scenes [131]. Iterative inference computes the recognition distribution via a *procedure*, rather than a single forward pass of an encoder, that iteratively refines an initial guess for the posterior parameters $\lambda_{1:K}$ by using gradients from how well the generative model is able to predict the observation based on the current posterior estimate. The initial guess provides the noise to break the symmetry.

For scenes where position and color are enough for disambiguating objects, a static image may be sufficient for inferring q. However, in interactive environments disambiguating objects is more underconstrained because what constitutes an object depends on the goals of the agent. We therefore incorporate actions into the amortized varitional filtering framework [215] to develop an *interactive inference* algorithm (Appdx. B.4 and Fig. 3.5) that uses temporal continuity and interactive feedback to disambiguate objects. Another benefit of enforcing entity abstraction is that preserving temporal consistency on entities comes for free: information about each object remains bound to its respective H_k through time, mixing with information about other entities only through explicitly defined avenues, such as in the dynamics model.

Training at Different Timescales

The variational parameters $\lambda_{1:K}$ are the interface through which the neural networks f_g, f_d, f_q that respectively output the distribution parameters of $\mathcal{G}$, $\mathcal{D}$, and $\mathcal{Q}$ communicate. For a *particular* dynamic scene, the execution of interactive inference optimizes the variational parameters $\lambda_{1:K}$. *Across* scene instances, we train the weights of f_g, f_d, f_q by backpropagating the ELBO through the entire inference procedure, spanning multiple timesteps. OP3 thus learns at three different timescales: the variational parameters learn (1) across M steps of

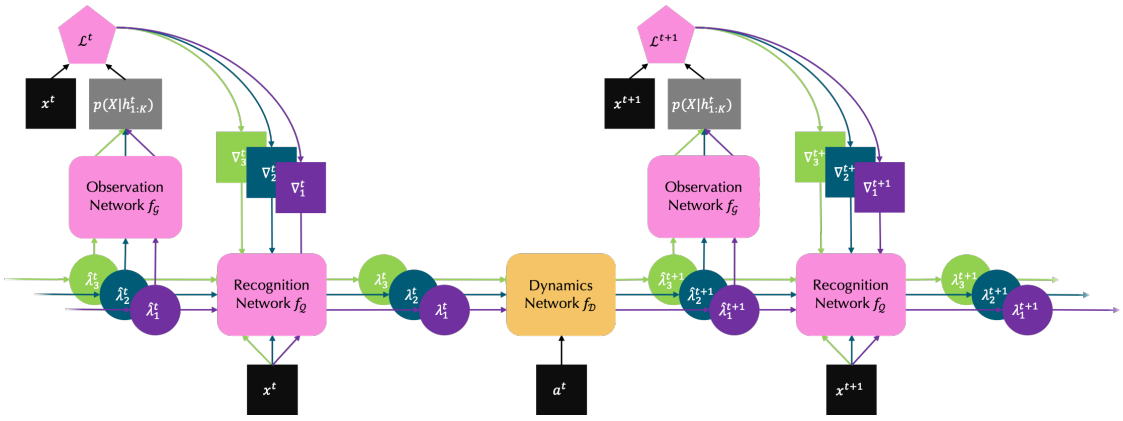


Figure 3.5: **Amortized interactive inference** alternates between refinement (pink) and dynamics (orange) steps, iteratively updating the belief of $\lambda_{1:K}$ over time. $\hat{\lambda}$ corresponds to the output of the dynamics network, which serves as the initial estimate of λ that is subsequently refined by $f_{\mathcal{G}}$ and $f_{\mathcal{Q}}$. ∇ denotes the feedback used in the refinement process, which includes gradient information and auxiliary inputs (Appdx. B.4).

inference within a single timestep and (2) across T timesteps within a scene instance, and the network weights learn (3) across different scene instances.

Beyond next-step prediction, we can directly train to compute the posterior predictive distribution $p(X^{(T+1:T+d)} \mid x^{(0:T)}, a^{(0:T+d)})$ by sampling from the approximate posterior of $H_{1:K}^{(T)}$ with $\mathcal{Q}$, rolling out the dynamics model $\mathcal{D}$ in latent space from these samples with a sequence of d actions, and predicting the observation $X^{(T+d)}$ with the observation model $\mathcal{G}$. This approach to action-conditioned video prediction predicts future observations directly from observations and actions, but with a bottleneck of K time-persistent entity-variables with which the dynamics model $\mathcal{D}$ performs symbolic relational computation.

Object-Centric Planning

OP3 rollouts, computed as the posterior predictive distribution, can be integrated into the standard visual model-predictive control [95] framework. Since interactive inference grounds the entities $H_{1:K}$ in the actual objects $h_{1:K}^{*}$ depicted in the raw observation, this grounding essentially gives OP3 access to a *pointer* to each object, enabling the rollouts to be in the space of entities and their relations. These pointers enable OP3 to not merely predict in the space of entities, but give OP3 access to an *object-centric action space*: for example, instead of being restricted to the standard `(pick_xy, place_xy)` action space common to many manipulation tasks, which often requires biased picking with a scripted policy [198, 170], these pointers enable us to compute a mapping (Appdx. B.7) between `entity_id` and `pick_xy`, allowing OP3 to automatically use a `(entity_id, place_xy)` action space without needing a scripted policy.

Generalization to Various Tasks

We consider tasks defined in the same environment with the same physical laws that govern appearance and dynamics. Tasks are differentiated by goals, in particular goal configurations of objects. Building good cost functions for real world tasks is generally difficult [104] because the underlying state of the environment is always unobserved and can only be modeled through modeling observations. However, by representing the environment state as the state of its entities, we may obtain finer-grained goal-specification without the need for manual annotations [83]. Having rolled out OP3 to a particular timestep, we construct a cost function to compare the predicted entity states $H^{(P)}_{1:K}$ with the entity states $H^{(G)}_{1:K}$ inferred from a goal image by considering pairwise distances between the entities, another example of enforcing the pairwise entity abstraction. Letting S' and S denote the set of goal and predicted entities respectively, we define the form of the cost function via a composition of the task specific distance function c operating on entity-pairs:

$$\mathcal{C}\left(H^{(G)}_{1:K}, H^{(P)}_{1:K}\right) = \sum_{a \in S'} \min_{b \in S} c\left(H^{(G)}_{a}, H^{(P)}_{b}\right), \tag{3.5}$$

in which we pair each goal entity with the closest predicted entity and sum over the costs of these pairs. Assuming a single action suffices to move an object to its desired goal position, we can greedily plan each timestep by defining the cost to be $\min_{a \in S', b \in S} \; c(H^{(G)}_a, H^{(P)}_b)$, the pair with minimum distance, and removing the corresponding goal entity from further consideration for future planning.

3.5 Experiments

Our experiments aim to study to what degree entity abstraction improves generalization, planning, and modeling. Sec. 3.5 shows that from only training to predict how objects fall, OP3 generalizes to solve various novel block stacking tasks with two to three times better accuracy than a state-of-the-art video prediction model. Sec. 3.5 shows that OP3 can plan for multiple steps in a difficult multi-object environment. Sec. 3.5 shows that OP3 learns to ground its abstract entities in objects from real world videos.

Combinatorial Generalization without Object Supervision

We first investigate how well OP3 can learn object-based representations without additional object supervision, as well as how well OP3's factorized representation can enable combinatorial generalization for scenes with many objects.

Domain: In the MuJoCo [313] block stacking task introduced by Janner et al. [164] for the O2P2 model, a block is raised in the air and the model must predict the steady-state effects of dropping the block on a surface with multiple objects, which implicitly requires modeling the effects of gravity and collisions. The agent is never trained to stack blocks, but is tested on a suite of tasks where it must construct block tower specified by a goal image. Janner et al. [164] showed that an object-centric model with access to *ground truth* object

SAVP	O2P2	OP3 (ours)
24%	76%	**82%**

Table 3.1: Accuracy (%) of block tower builds by the SAVP baseline, the O2P2 oracle, and our approach. O2P2 uses image segmentations whereas OP3 uses only raw images as input.

# Blocks	SAVP	OP3 (xy)	OP3 (entity)
1	54%	73%	**91%**
2	28%	55%	**80%**
3	28%	41%	**55%**

Table 3.2: Accuracy (%) of multi-step planning for building block towers. (xy) means `(pick_xy, place_xy)` action space while (entity) means `(entity_id, place_xy)` action space.

segmentations can solve these tasks with about 76% accuracy. We now consider whether OP3 can do better, but *without any supervision on object identity.*

Setup: We train OP3 on the same dataset and evaluate on the same goal images as Janner et al. [164]. While the training set contains up to five objects, the test set contains up to nine objects, which are placed in specific structures (bridge, pyramid, etc.) not seen during training. The actions are optimized using the cross-entropy method (CEM) [264], with each sampled action evaluated by the greedy cost function described in Sec. 3.4. Accuracy is evaluated using the metric defined by Janner et al. [164], which checks that all blocks are within some threshold error of the goal.

Results: The two baselines, SAVP [194] and O2P2, represent the state-of-the-art in video prediction and symmetric object-centric planning methods, respectively. SAVP models objects with a fixed number of convolutional filters and does not process entities symmetrically. O2P2 does process entities symmetrically, but requires access to ground truth object segmentations. As shown in Table 3.1, OP3 achieves better accuracy than O2P2, even without any ground truth supervision on object identity, possibly because grounding the entities in the raw image may provide a richer contextual representation than encoding each entity separately without such global context as O2P2 does. OP3 achieves three times the accuracy of SAVP, which suggests that symmetric modeling of entities is enables the flexibility to transfer knowledge of dynamics of a single object to novel scenes with different configurations heights, color combinations, and numbers of objects than those from the training distribution. Fig. B.1 and Fig. B.2 in the Appendix show that, by grounding its entities in objects of the scene through inference, OP3's predictions isolates only one object at a time without affecting the predictions of other objects.

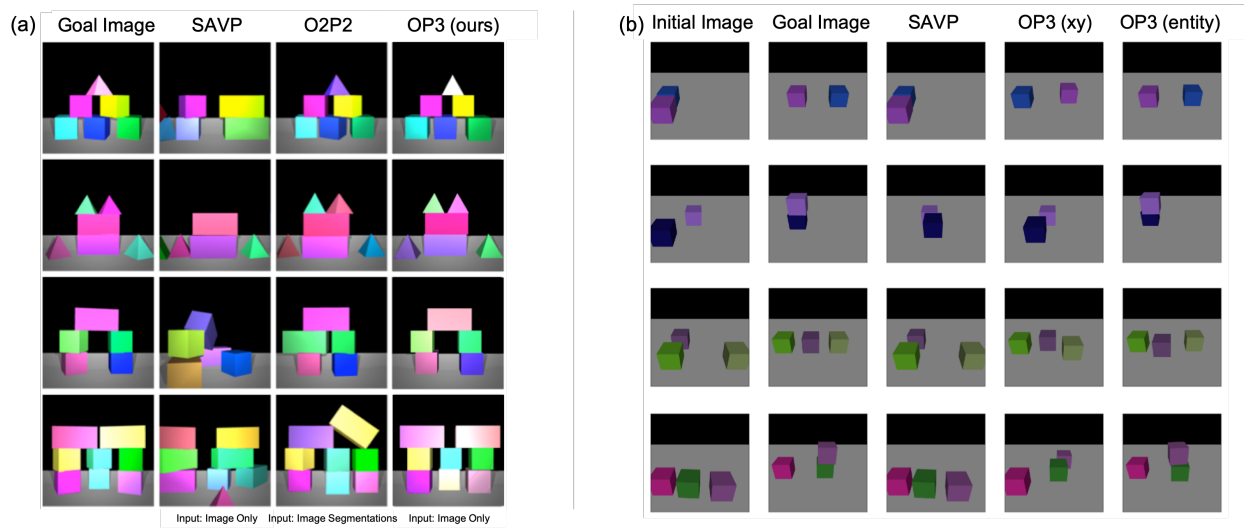


Figure 3.6: **(a)** In the block stacking task from [164] with single-step greedy planning, OP3 generalizes better than both O2P2, an oracle model with access to image segmentations, and SAVP, which does not enforce entity abstraction. **(b)** OP3 exhibits better multi-step planning with objects already present in the scene. By planning with MPC using random pick locations (SAVP and OP3 (xy)), the sparsity of objects in the scene make it rare for random pick locations to actually pick the objects. Because OP3 has access to pointers to the latent entities, we can use these to automatically bias the pick locations to be at the object location, without any supervision (OP3 (entity)).

Multi-Step Planning

The goal of our second experiment is to understand how well OP3 can perform multi-step planning by manipulating objects already present in the scene. We modify the block stacking task by changing the action space to represent a picking and dropping location. This requires reasoning over extended action sequences since moving objects out of place may be necessary.

Goals are specified with a goal image, and the initial scene contains all of the blocks needed to build the desired structure. This task is more difficult because the agent may have to move blocks out of the way before placing other ones which would require multi-step planning. Furthermore, an action only successfully picks up a block if it intersects with the block's outline, which makes searching through the combinatorial space of plans a challenge. As stated in Sec. 3.4, having a pointer to each object enables OP3 to plan in the space of entities. We compare two different action spaces `(pick_xy, place_xy)` and `(entity_id, place_xy)` to understand how automatically filtering for pick locations at actual locations of objects enables better efficiency and performance in planning. Details for determining the `pick_xy` from `entity_id` are in appendix B.7.

Results: We compare with SAVP, which uses the `(pick_xy, place_xy)` action space. With this standard action space (Table 3.2) OP3 achieves between 1.5-2 times the accuracy of SAVP. This performance gap increases to 2-3 times the accuracy when OP3 uses the

`(entity_id, place_xy)` action space. The low performance of SAVP with only two blocks highlights the difficulty of such combinatorial tasks for model-based RL methods, and highlights the both the generalization and localization benefits of a model with entity abstraction. Fig. 3.6b shows that OP3 is able to plan more efficiently, suggesting that OP3 may be a more effective model than SAVP in modeling combinatorial scenes. Fig. 3.7a shows the execution of interactive inference during training, where OP3 alternates between four refinement steps and one prediction step. Notice that OP3 infers entity representations that decompose the scene into coherent objects and that entities that do not model objects model the background. We also observe in the last column ($t = 2$) that OP3 predicts the appearance of the green block even though the green block was partially occluded in the previous timestep, which shows its ability to retain information across time.

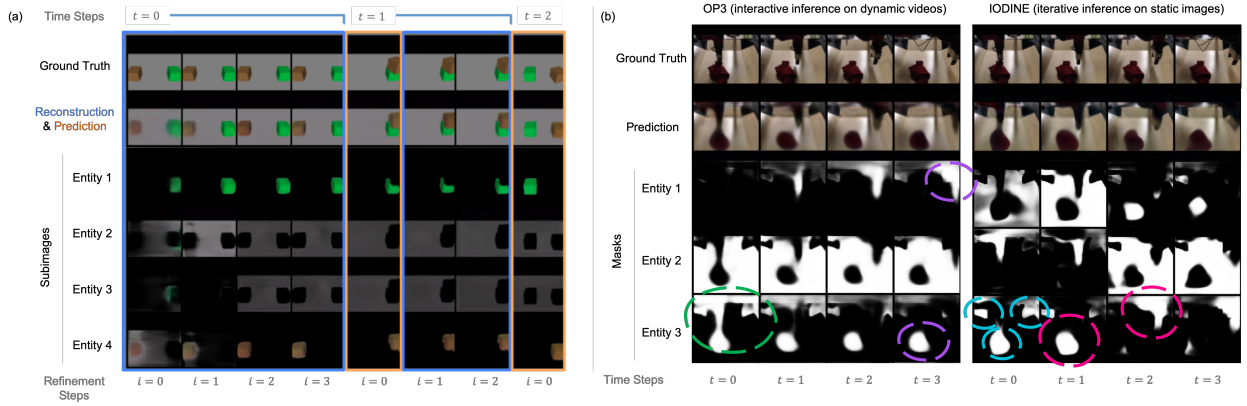


Figure 3.7: Visualization of interactive inference for block-manipulation and real-world videos [82]. Here, OP3 interacts with the objects by executing pre-specified actions in order to disambiguate objects already present in the scene by taking advantage of temporal continuity and receiving feedback from how well its prediction of how an action affects an object compares with the ground truth result. **(a)** OP3 does four refinement steps on the first image, and then 2 refinement steps after each prediction. **(b)** We compare OP3, applied on dynamic videos, with IODINE, applied independently to each frame of the video, to illustrate that using a dynamics model to propagate information across time enables better object disambiguation. We observe that initially, both OP3 (green circle) and IODINE (cyan circles) both disambiguate objects via color segmentation because color is the only signal in a static image to group pixels. However, we observe that as time progresses, OP3 separates the arm, object, and background into separate latents (purple) by using its currently estimates latents predict the next observation and comparing this prediction with the actually observed next observation. In contrast, applying IODINE on a per-frame basis does not yield benefits of temporal consistency and interactive feedback (red).

Real World Evaluation

The previous tasks used simulated environments with monochromatic objects. Now we study how well OP3 scales to real world data with cluttered scenes, object ambiguity, and occlusions. We evaluate OP3 on the dataset from Ebert et al. [82] which contains videos of a robotic arm moving cloths and other deformable and multipart objects with varying textures.

We evaluate qualitative performance by visualizing the object segmentations and compare against vanilla IODINE, which does not incorporate an interaction-based dynamics model into the inference process. Fig. 3.7b highlights the strength of OP3 in preserving temporal continuity and disambiguating objects in real world scenes. While IODINE can disambiguate monochromatic objects in static images, we observe that it struggles to do more than just color segmentation on more complicated images where movement is required to disambiguate objects. In contrast, OP3 is able to use temporal information to obtain more accurate segmentations, as seen in Fig. 3.7b where it initially performs color segmentation by grouping the towel, arm, and dark container edges together, and then by observing the effects of moving the arm, separates these entities into different groups.

3.6 Discussion

We have shown that enforcing the entity abstraction in a model-based reinforcement learner improves generalization, planning, and modeling across various compositional multi-object tasks. In particular, enforcing the entity abstraction provides the learner with a pointer to each entity variable, enabling us to define functions that are local in scope with respect to a particular entity, allowing knowledge about an entity in one context to directly transfer to modeling the same entity in different contexts. In the physical world, entities are often manifested as objects, and generalization in physical tasks such as robotic manipulation often may require symbolic reasoning about objects and their interactions. However, the general difficulty with using purely symbolic, abstract representations is that it is unclear how to continuously update these representations with more raw data. OP3 frames such symbolic entities as random variables in a dynamic latent variable model and infers and refines the posterior of these entities over time with neural networks. This suggests a potential bridge to connect abstract symbolic variables with the noisy, continuous, high-dimensional physical world, opening a path to scaling robotic learning to more combinatorially complex tasks.

Part II

Transformations

Chapter 4

Representing Physical Transformations

4.1 Introduction

The power of an abstraction depends on its usefulness for solving new problems. Object rearrangement [26] offers an intuitive setting for studying the problem of learning reusable abstractions. Solving novel rearrangement problems requires an agent to not only infer object representations without supervision, but also recognize that the same action for moving an object between two locations can be reused for different objects in different contexts.

We study the simplest setting in simulation with pick-and-move action primitives that move one object at a time. Even such a simple setting is challenging because the space of object configurations is combinatorially large, resulting in long-horizon combinatorial task spaces. We formulate rearrangement as an offline goal-conditioned reinforcement learning (RL) problem, where the agent is pretrained on a experience buffer of sensorimotor interactions and is evaluated on producing actions for rearranging objects specified in the input image to satisfy constraints depicted in a goal image.

Offline RL methods [199] that do not infer factorized representations of entities struggle to generalize to problems with more objects. But planning with object-centric methods that do infer entities [319] is also not easy because the difficulties of long-horizon planning with learned parametric models [165] are exacerbated in combinatorial spaces.

Instead of planning with parametric models, our work takes inspiration from non-parametric planning methods that have shown success in combining neural networks with graph search to generate long-horizon plans. These methods [342, 350, 205, 86] explicitly construct a transition graph from the experience buffer and plan by searching through the actions recorded in this transition graph with a learned distance metric. The advantage of such approaches is the ability to stitch different path segments from offline data to solve new problems. The disadvantage is that the non-parametric nature of such methods requires transitions that will be used for solving new problems to have already been recorded in the buffer, making conventional methods, which store entire observations monolithically, ill-suited for combinatorial generalization. Fig. 4.2b shows that the same state transition can manifest

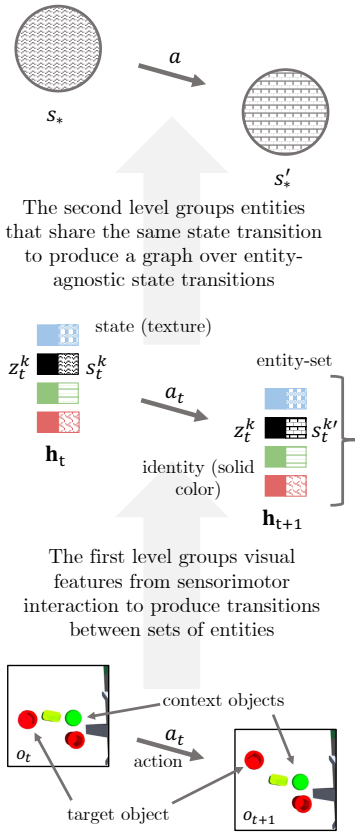


Figure 4.1: NCS uses a two-level hierarchy to abstract sensorimotor interactions into a graph of learned state transitions. The affected entity is in black.

for different objects and in different contexts, but monolithic non-parametric methods are not constrained to recognize that all scenarios exhibit the same state transition at an abstract level. This induces an blowup in the number of nodes of the search graph. To overcome this problem, we devise a method that explicitly exploits the similarity among state transitions in different contexts.

Our method, **Neural Constraint Satisfaction** (NCS), marries the strengths of non-parametric planning with those of object-centric representations. Our main contribution is to show that factorizing the traditionally monolithic entity representation into action-invariant features (its **type**) and action-dependent features (its **state**) makes it possible during planning and control to reuse action representations for different objects in different contexts, thereby addressing the core combinatorial challenge in object rearrangement. To implement this factorization, NCS constructs a two-level hierarchy (Fig. 4.1) to abstract the experience buffer into a graph over state transitions of individual entities, separated from other contextual entities (Fig. 4.3). To solve new rearrangement problems, NCS infers what state transitions

can be taken given the current and goal image observations, re-composes sequences of state transitions from the graph, and translates these transitions into actions.

In §4.3 we introduce a problem formulation that exposes the combinatorial structure of object rearrangement tasks by explicitly modeling the independence, symmetry, and factorization of latent entities. This reveals two challenges in object rearrangement which we call the **correspondence problem** and **combinatorial problem**. In §4.4 we present NCS, a method for controlling an agent that plans over and acts with emergent learned entity representations, as a unified method for tackling both challenges. We show in §4.5 that NCS outperforms both state-of-the-art offline RL methods and object-centric shooting-based planning methods in simulated rearrangement problems.

4.2 Related Work

The problem of discovering re-composable representations is generally motivated by combinatorial task spaces. The traditional approach to enforcing this compositional inductive bias is to compactly represent the task space with MDPs that human-defined abstractions of entities, such as factored MDPs [35, 34, 137], relational MDPs [322, 138, 108], and object-oriented MDPs [76, 1]. Approaches building off of such symbolic abstractions [57, 28, 346, 24, 350] do not address the problem of how such entity abstractions arise from raw data. Our work is one of the first to learn compact representations of combinatorial task spaces directly from raw sensorimotor data.

Recent object-centric methods [130, 317, 132, 129, 208, 177, 356, 290] do learn entity representations, as well as their transformations [119, 120], from sensorimotor data, but only do so for modeling images and video, rather than for taking actions. Instead, we study *how well entity-representations can reused for solving tasks.* Kulkarni et al. [184] considers how object representations improve exploration, but we consider the offline setting which requires zero-shot generalization. Veerapaneni et al. [319] also considers on control tasks, but their shooting-based planning method in suffers from compounding errors as other learned single-step models do [165], while our hierarchical non-parametric approach enables us to plan for longer horizons.

Non-parametric approaches have recently become popular for long horizon planning [342, 350, 205, 86, 351], but the drawback of these approaches is they represent the entire scenes monolithically, which causes a blowup of nodes in combinatorial task spaces, making it infeasible for these methods to be applied in rearrangement tasks that require generalizing to novel object configurations with different numbers of objects. Similar to our work, Huang et al. [157] also tackles rearrangement problems by searching over a constructed latent task graph, but they require a demonstration during deployment time, whereas NCS does not because it reuses context-agnostic state transitions that were constructed during training. [351] conducts non-parametric planning directly on abstract subgoals rather than object-centric states — while similar, the downside of using subgoals rather than abstract states is that those subgoals are not used to represent equivalent states and therefore cannot generalize to new states at

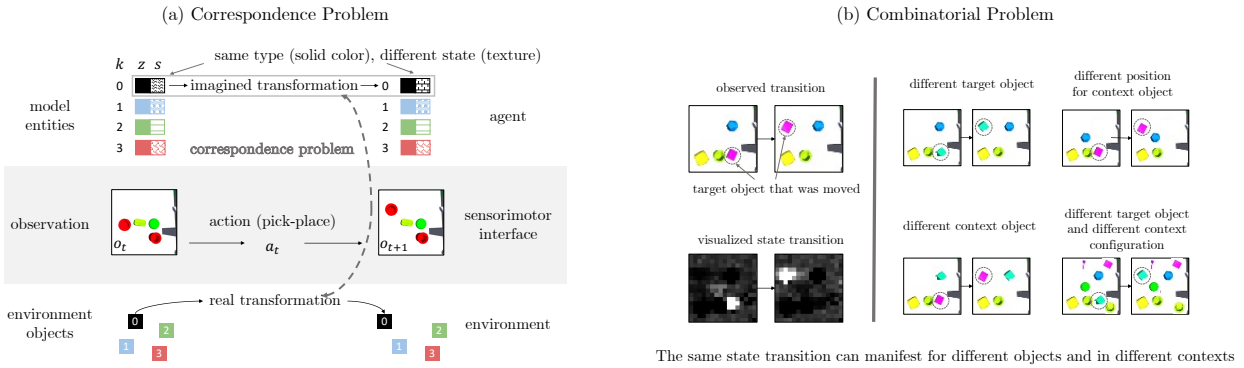


Figure 4.2: **Solving object rearrangement requires solving two challenges**. (a) The **correspondence problem** is the problem of abstracting raw sensorimotor signal into representations of entities such that there is a correspondence between how an agent intervenes on an entity and how its action affects an object in the environment. k denotes the index of the entity, z denotes its type (shown with solid colors), and s denotes its state (shown with textures). The entity representing the moved object is in black. (b) The **combinatorial problem** is the problem of representing the combinatorial task space in a way that enables an agent to transfer knowledge of a given state transition (indicated by the dotted circle) to different contexts.

test time. Our method, NCS, captures both reachability between known states and new, unseen states that can be mapped to the same abstract state.

4.3 Goal-Conditioned Reinforcement Learning with Entities

This section introduces a set of modifications to the standard goal-conditioned partially observed Markov decision process (POMDP) problem formulation that explicitly expose the combinatorial structure of object rearrangement tasks of the following kind: "Sequentially move a subset (or all) of the objects depicted in the current observation o_1 to satisfy the constraints depicted in the goal image o_g." We assume an offline RL setting, where the agent is trained on a buffer of transitions $\{(o_1, a_1, ...a_{T-1}, o_T)\}_{n=1}^{N}$ and evaluated on tasks specified as (o_1, o_g).

The standard POMDP problem formulation assumes an observation space $\mathcal{O}$, action space $\mathcal{A}$, latent space $\mathcal{H}$, goal space $\mathcal{G}$, observation function $E : \mathcal{H} \to \mathcal{O}$, transition function $P : \mathcal{H} \times \mathcal{A} \to \mathcal{H}$, and reward function $R : \mathcal{H} \times \mathcal{G} \to \mathbb{R}$. Monolithically modeling the latent space this way does not expose commonalities among different scenes, such as scenes that contain objects in the same location or scenes with multiple instances of the same type of object, which prevents us from designing control algorithms that exploit these commonalities to collapse the combinatorial task space.

To overcome this issue, we introduce structural assumptions of independence, symmetry,

and factorization to the standard formulation. The *independence* assumption encodes the intuitive property that objects can be acted upon without affecting other objects. This is implemented by decomposing the latent space into independent subspaces as $\mathcal{H} = \mathcal{H}^1 \times ... \times \mathcal{H}^K$, one for each independent degree of freedom (e.g. object) in the scene. The *symmetry* assumption encodes the property that the the same physical laws apply to all objects. This is implemented by constraining the observation function E, transition function P and reward function R to be shared across all subspaces, thereby treating $\mathcal{H}^1 = ... = \mathcal{H}^K$. We define an **entity**[1] $h \in \mathcal{H}^k$ as a member of such a subspace, and an **entity-set** as the set of entities $\mathbf{h} = (h^1, ..., h^K)$ that explain an observation, similar to Diuk, Cohen, and Littman [76] and Weld [328]. Lastly, the *factorization* assumption encodes that each subspace can be decomposed as $\mathcal{H}^k = \mathcal{Z} \times \mathcal{S}$, where the $z \in \mathcal{Z}$ represents the entity's action-invariant features like appearance, and $s \in \mathcal{S}$ represents its action-dependent features like location. We call z the **type** and s the **state**.

Introducing these assumptions solves the problem of modeling the commonalities among different scenes stated above. It allows us to describe scenes that contain objects in the same location by assigning entities in different scenes to share the same state s. It allows us to describe a scene with multiple instances of the same type of object by assigning multiple entities in the scene to share the same type z. This formulation also makes it natural to express goals as a set of constraints $\mathbf{h}_g = (h_g^1, ..., h_g^k)$. To solve a task is to take actions that transform the subset of entities in the initial observation o_1 whose types are given by $\mathbf{z}_g$ to new states specified by $\mathbf{s}_g$.

Exposing this structure in our problem formulation enables us to exploit it by designing methods that represent entities in an independent, symmetric, and factorized way and that use these three properties to collapse the combinatorial task space. To do so involves solving two problems: the **correspondence problem** of learning to represent entities in this way and the **combinatorial problem** of using these properties to make planning tractable. The correspondence problem is hard because it assumes no human supervision of what the entities are. It also goes beyond problems solved by existing object-centric methods for images and videos because it involves action: it requires representing entities such that there is a correspondence between how the agent models how its actions affect entities and how its actions actually affect objects in the environment. The combinatorial problem goes beyond problems solved by methods for solving object-oriented MDPs, relational MDPs, and factorized MDPs because it requires the agent to recognize whether and how previously observed state transitions can be used for new problems, using learned, not human-defined, entity representations. The natural evaluation criterion for both problems is to test to what extent an agent can zero-shot-generalize to solve rearrangement tasks involving new sets of object configurations that aree disjoint from the configurations observed in training, assuming that the training configurations have collectively covered $\mathcal{Z}$ and $\mathcal{S}$. Our experiments in §4.5 test exactly this.

[1]We use "object" to refer to an independent degree of freedom in the environment, and "entity" to refer to the agent's representation of the object.

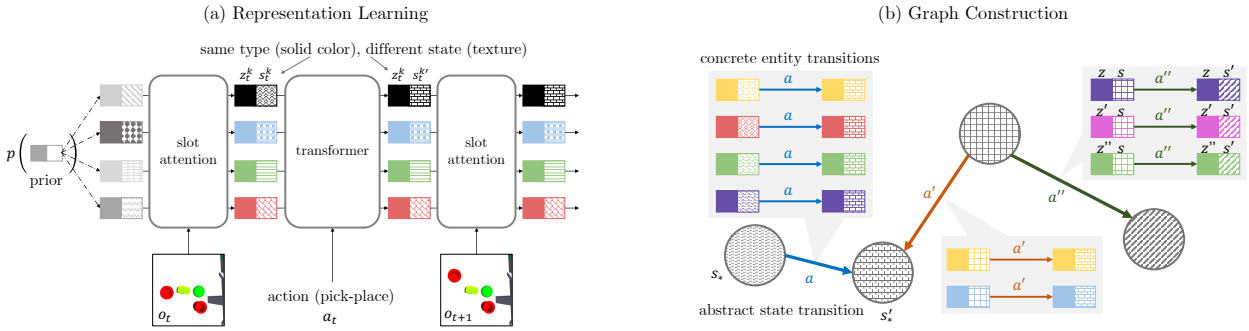


Figure 4.3: **Modeling**. NCS constructs a two-level abstraction hierarchy to model transitions in the experience buffer. (a) **Level 1:** NCS learns to infer a set of entities from sensorimotor transitions with pick-and-move actions, in which one entity is moved per transition. We enforce that the type z (shown with solid colors) of an entity remains unchanged between time-steps. The GPT dynamics model learns to sparsely predict the states s (shown with textures) of the entities at the next time-step. *This addresses the correspondence problem by forcing the network to use predict and reconstruct observations through the entity bottleneck.* (b) **Level 2:** NCS abstracts transitions over entity-sets into transitions over states of individual entities, constructing a graph where states are nodes and transitions between them are edges. This is done by clustering entity transitions that share similar initial states and final states. *This addresses the combinatorial problem by making it possible for state transitions to reused for different entity types and with different context entities.*

Simplifying assumptions To focus on the combinatorial nature of rearrangement, we are not interested in low-level manipulation, so we represent each action as $(w, \Delta w)$, where w are Cartesian coordinates $w = (x, y, z)$. We assume actions sparsely affect one entity at a time and how an action affects an object's state does not depend on its identity. We are not interested in handling occlusion, so we assume that objects are constrained to the xy plane or xz plane and are directly visible to the camera. Following prior work [147, 47], we make a *bisimulation* assumption that the state space can be partitioned into a finite set of equivalence classes, and that there is one action primitive that transitions between each pair of equivalence classes. Lastly, we assume objects can be moved independently. Preliminary experiments suggest that NCS can be augmented to support tasks like block-stacking that involve dependencies among objects, but how to handle these dependencies would warrant a standalone treatment in future work.

4.4 Neural Constraint Satisfaction

In §4.3 we introduced a structured problem formulation for object rearrangement and reduced it to solving the correspondence and combinatorial problems. We now present our method, Neural Constraint Satisfaction (NCS) as a method for controlling an agent that plans over and acts with a state transition graph constructed from learned entity representations. This section is divided into two parts: modeling and control. The modeling part is further divided

into two parts: representation learning and graph construction. The representation learning part addresses the correspondence problem, while the graph construction and control parts address the combinatorial problem.

Modeling

The modeling component of NCS abstracts the experience buffer into a factorized state transition graph that can be reused across different rearrangement problems. Below we describe how we first train an object-centric world model to infer entities that are independent, symmetric, and factorized and then construct the state transition graph by clustering entities with similar state transitions. These two steps comprise a two-level abstraction hierarchy over the raw sensorimotor transitions.

Level 1: representation learning The first level concerns the unsupervised learning of entity representations that factorizes into their action-invariant features (their **type**) and their action-dependent features (their **state**). Concretely our goal is to model a video transition $o_t, a_t \rightarrow o_{t+1}$ as a transition over entity-sets $\mathbf{h}_t, a_t \rightarrow \mathbf{h}_{t+1}$, where each entity h^k is factorized as a pair $h^k = (z^k, s^k)$. Given our setting where an action moves only a single object in the environment at a time, successful representation learning implies three criteria: (1) the world model properly identifies the individual entity h^k corresponding to the moved object, (2) only the state s^k of that entity should change, while its type z^k should remain unaffected, and (3) other entity representations $h^{\neq k}$ should also remain unaffected. Criteria (1) and (3) rule out standard approaches that represent an entire scene with a monolithic representation, so we need an object-centric world model instead of a monolithic world model. But criterion (2) rules out standard object-centric world models (e.g. [319, 85, 291]), which do not decompose entity representations into action-invariant and action-dependent features.

Because the parameters of a mixture model are independent and symmetric by construction, we propose to construct our factorized object-centric world model as an equivariant sequential Bayesian filter with a mixture model as the latent state, where entity representations are the parameters of the mixture components. Recall that a filter consists of two major components, latent estimation and latent prediction. We implement latent estimation with the state-of-the-art slot attention (SA) [207], based on the connection Chang, Griffiths, and Levine [53] between mixture components and SA slots. We implement latent prediction with the transformer decoder (TFD) architecture [318] because TFD is equivariant with respect to its inputs. We denote the SA `slots` as $\boldsymbol{\lambda}$ and SA `attn` masks as $\boldsymbol{\alpha}$. We split each `slot` $\lambda \in \mathbb{R}^n$ into two halves $\lambda^z \in \mathbb{R}^{\frac{n}{2}}$ and $\lambda^s \in \mathbb{R}^{\frac{n}{2}}$. Given observations o and actions a, we embed the actions as $\tilde{a}$ with an feedforward network and implement the filter as:

$$\hat{\boldsymbol{\lambda}}_1 \sim \text{Gaussian} \qquad \hat{\boldsymbol{\lambda}^s}_{t+1} = TFD\left(\text{queries} = \boldsymbol{\lambda}^s_t, \text{keys/values} = [\boldsymbol{\lambda}^s, \tilde{a}_t]\right)$$

$$\boldsymbol{\lambda}_t, \boldsymbol{\alpha}_t = SA\left(\hat{\boldsymbol{\lambda}}_t, o_t\right) \qquad \hat{\boldsymbol{\lambda}}_{t+1} = \left[\boldsymbol{\lambda}^z_t, \hat{\boldsymbol{\lambda}^s}_{t+1}\right]$$

where $[\cdot, \cdot]$ is the concatenation operator, $\hat{\boldsymbol{\lambda}}$ is the output of the latent prediction step, and $\boldsymbol{\lambda}$ is the output of the latent estimation step. We embed this filter inside the SLATE backbone [289] and call this implementation **dynamic SLATE** (dSLATE).

By constructing $\hat{\boldsymbol{\lambda}}^z_{t+1}$ as a copy of $\boldsymbol{\lambda}^z_t$, dSLATE enforces the information contained $\boldsymbol{\lambda}^z$ to be action-invariant, hence we treat $\boldsymbol{\lambda}^z$ as dSLATE's representation of the entities' types. As for the entities' states, either the action-dependent part of the slots $\boldsymbol{\lambda}^s$ or the attention masks α can be used. Using α may be sufficient and more intuitive to analyze if all objects looks similar and there is no occlusion, while $\boldsymbol{\lambda}^s$ may be more suitable in other cases, and we provide an example of each in the experiments. To simplify notation going forward and connect with the notation in §4.3, we use $\mathbf{h}$ to refer to $(\boldsymbol{\lambda}, \boldsymbol{\alpha})$, use $\boldsymbol{z}$ to refer to $\boldsymbol{\lambda}^z$, and use $\boldsymbol{s}$ to refer to $\boldsymbol{\lambda}^s$ or $\boldsymbol{\alpha}$. Thus by construction dSLATE satisfies criterion (2). Empirically we observe that it satisfies criterion (1) as well as SLATE does, and that TFD learns to sparsely edit $\boldsymbol{\lambda}^s_t$, thereby satisfying criterion (3).

Algorithm 1 Building the Graph

input `model`, `buffer`
for $\{(o_t, a_t, o_{t+1})\}_n$ in `buffer` **do**
    # infer entities from transition
    $\{(\mathbf{h}_t, a_t, \mathbf{h}_{t+1})\}_n \leftarrow$ `model` $(\{o_t, a_t, o_{t+1}\}_n)$.
    # identify which entity changed in transition
    $\{(h^k_t, a_t, h^k_{t+1})\}_n \leftarrow$ `isolate` $(\{(\mathbf{h}_t, a_t, \mathbf{h}_{t+1})\}_n)$
end for
# partition transitions by clustering entities
$\{s_*\}^M_{m=1} \leftarrow$ `cluster` $\left(\{(s^k_t, a_t, s^k_{t+1})\}^N_{n=1}\right)$
transitions between clusters are edges
initialize `graph` with nodes $s^{[m]}_*$, for $m \in [1 : M]$
for each $\{(h^k_t, a_t, h^k_{t+1})\}_n$ **do**
    # infer cluster assignments
    $[i], [j] \leftarrow$ `bind` $\left(h^k_t\right)$, `bind` $\left(h^k_{t+1}\right)$
    # tag edge with action a_t
    `graph.edges`$[i, j] \leftarrow$ `create-edge` $\left(s^{[i]}_* \xrightarrow{a_t} s^{[j]}_*\right)$
end for
return `graph`

Level 2: graph construction Having produced from the first level a buffer of entity-set transitions $\{\mathbf{h}_t, a_t \rightarrow \mathbf{h}_{t+1}\}^N_{n=1}$, the goal of the second level (Fig. 4.3b) is to use this buffer to construct a factorized state transition graph. The key to solving the combinatorial problem is to construct the edges of this graph to represent not state transitions of entire entity-sets (i.e. $\mathbf{s}_t, a_t \rightarrow \mathbf{s}_{t+1}$) as prior work does [350], but state transitions of *individual entities* (i.e. $s^k_t, a_t \rightarrow s^k_{t+1}$). Constructing edges over transitions for individual entities rather than entity sets enables the same transition to be reused with different context entities

present. Constructing edges over state transitions instead of entity transitions enables the same transition to be reused across entities with different types. This would enable the agent to recompose sequences of previously encountered state transitions for solving new rearrangement problems with different entities in different contexts. Henceforth our use of "state" refers specifically to the state of individual entity unless otherwise stated.

Given our bisimulation assumption that states can be partitioned into a finite number of groups, we construct our graph such that nodes represent equivalence classes among individual states and the edges represent actions that transform a state from one equivalence class to another. To implement this we cluster state transitions of individual entities in the buffer, which reduces to clustering the states of individual entities before and after the transition. We treat each cluster centroid as a node in the graph, and an edge between nodes is tagged with the single action that transforms one node's state to another's. The algorithm for constructing the graph is shown in Alg. 1 and involves three steps: (1) isolating the state transition of an individual entity from the state transition of the entity-set, (2) creating graph nodes from state clusters, and (3) tagging graph edges with actions.

The first step is to identify which object was moved in each transition, i.e. identifying the entity h^k that dSLATE predicted was affected by a_t in the transition $(\mathbf{h}_t, a_t, \mathbf{h}_{t+1})$. We implement a function `isolate` that achieves this by solving $k = \text{argmax}_{k' \in \{1,\dots,K\}} d(s_t^{k'}, s_{t+1}^{k'})$ to identify the index of the entity whose state has most changed during the transition, where $d(\cdot, \cdot)$ is a distance function, detailed in Table C.2 of the Appendix. This converts the buffer of transitions over entity-sets $\mathbf{h}_t, a_t \to \mathbf{h}_{t+1}$ into a buffer of transitions over individual entities $h_t^k, a_t \to h_{t+1}^k$.

The second step is to cluster the states before and after each transition. We implement a function `cluster` that uses K-means to returns graph nodes as the centroids $\{s_*\}_{m=1}^M$ of these state clusters.

The third step is to connect the nodes with edges that record actions that actually were taken in the buffer to transform one state to the next. We implement a function `bind` that, given entity h^k, returns the index $[i]$ of the centroid s_* that is the nearest neighbor to the entity's state s^k. For each entity transition (h_t^k, a_t, h_{t+1}^k) we `bind` entity h_t^k and h_{t+1}^k to their associated nodes $s_*^{[i]}$ and $s_*^{[j]}$ and create an edge between $s_*^{[i]}$ and $s_*^{[j]}$ tagged with action a, overwriting previous edges based on the assumption that with a proper clustering there should only be one action per pair of nodes.

In our experiments both `cluster` and `bind` use the same distance metric (see Table C.1 in the Appendix), but other clustering algorithms and distance metrics can also be used. Our experiments (Fig. C.5) also show that it is also possible to have more than one action primitive per pair of nodes as long as these actions all map between states bound to the same pair of nodes.

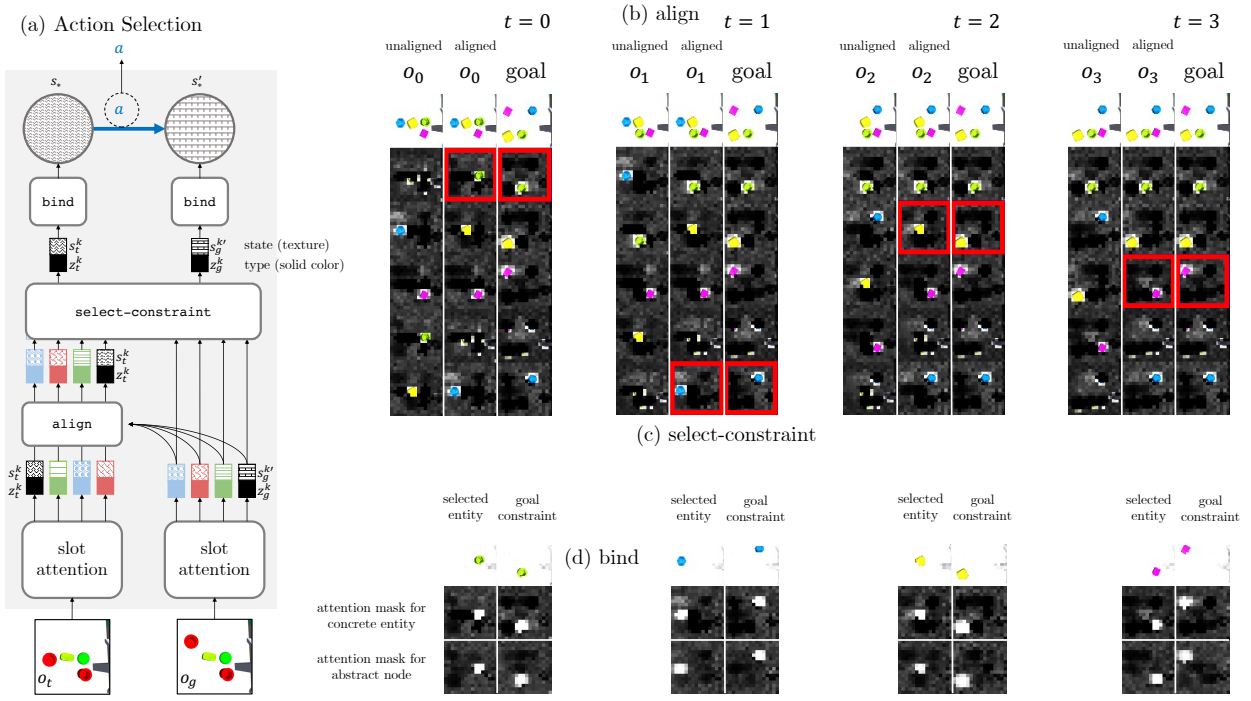


Figure 4.4: **Planning and control**. Given a rearrangement problem specified only by the current and goal observations (o_0, o_g), NCS decomposes the rearrangement problem into one subproblem (o_t, o_g) per entity. (a) shows the computations NCS uses to solve each subproblem and (b-d) show these steps in context. For each subproblem (o_t, o_g), NCS infers entities from both the current and goal observations. The states of the goal entities indicate constraints on the desired locations of the current entities. (b) NCS aligns the indices of the current entities to those of the goal entities with corresponding types. (c) It selects the index k of the next goal constraint s_g^k to satisfy, as indicated by the red box. The selected goal constraint and current entity are also colored black in (a), and note that their types are the same but states are different; we want to choose the action to transform the state of the current entity to the state of the goal constraint. (d) It binds the selected goal constraint and its corresponding current entity to nodes s_* and s'_* in the transition graph. Lastly, it identifies the edge connecting those two nodes and executes the action tagged to that edge in the environment.

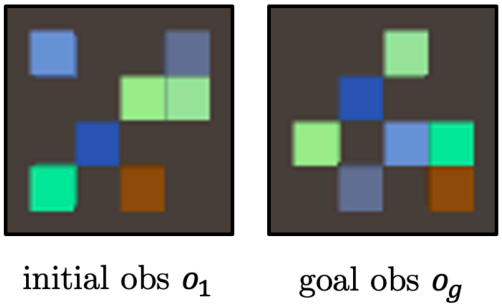
Control

To solve new rearrangement problems, we re-compose sequences of state transitions from the graph. Specifically, the agent decomposes the rearrangement problem into a set of per-entity subproblems (e.g. initial and goal positions for individual objects), searches the transition graph for a transition that transforms the current entity's state to its goal state, and executes the action tagged with this transition in the environment. This problem decomposition is possible because the transitions in our graph are constructed to be agnostic to type and context, enabling different rearrangement problems to share solutions to the same subproblems. The core challenge in deciding which transitions to compose is in determining which transitions are *possible* to compose. That is, the agent must determine which nodes in the graph correspond to the given goal constraints and which nodes correspond to the entities in the current observation, but the current entities $\mathbf{h}_t$ and goal constraints $\mathbf{h}_g$ must themselves be inferred from the current and goal observations o_t and o_g, requiring the agent to infer both what to do and how to do it purely from its sensorimotor interface.

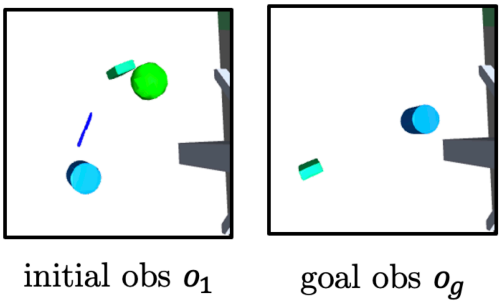
Algorithm 2 Action Selection

given `model`, `graph`
input goal o_g, observation o_t
infer goal constraints and current entities
$\mathbf{h_g}, \mathbf{h_t} \leftarrow \texttt{model}\left(o_g\right), \texttt{model}\left(o_t\right)$
align entity indices of $\mathbf{h_t}$ with those of $\mathbf{h_g}$
$\pi \leftarrow \texttt{align}\left(\mathbf{h_t}, \mathbf{h_g}\right)$
permute indices of $\mathbf{h_t}$ according to π
$\mathbf{h_t} \leftarrow (h_t^{\pi[1]}, ..., h_t^{\pi[K]})$
identify kth goal constraint to satisfy next
$k \leftarrow \texttt{select-constraint}\left(\mathbf{h_t}, \mathbf{h_g}\right)$
infer cluster assignments
$[i], [j] \leftarrow \texttt{bind}\left(h_t^k\right), \texttt{bind}\left(h_g^k\right)$
action that transforms node $[i]$ to node $[j]$
return `graph.edges`$[i, j]$`.action`

Our approach takes four steps, summarized in Alg. 2 and Fig. 4.4. In the first step, we use dSLATE to infer $\mathbf{h}_t$ and $\mathbf{h}_g$ from o_t and o_g (e.g. the positions and types of all objects in the initial and goal images). In the second step (Fig. 4.4b), because of the permutation symmetry among entities, we find a bipartite matching that matches each entities in h_g^j with a corresponding entity in h_t^k that shares the same type and permute the indices k of $\mathbf{h}_t$ to match those of $\mathbf{h}_g$. We implement a function `align` that uses the Hungarian algorithm to perform this matching over $(z_t^1, ...z_t^K)$ and $(z_g^1, ...z_g^K)$, with Euclidean distance as the matching cost. The third step selects which goal constraint h_g^k to satisfy next (Fig. 4.4c). W implement this `select-constraint` procedure by determining which constraint h_g^k has the highest difference in state with its counterpart h_t^k, which reduces to solving the same argmax problem as in



(a) *block-rearrange*



(b) *robogym-rearrange*

Figure 4.5: Our environments are *block-rearrange* and *robogym-rearrange*. Fig. 4.5a shows a complete specification of goal constraints; Fig. 4.5b shows a partial specification that only specifies the desired locations for two objects.

`isolate` with the same distance function used in `isolate`. The last step chooses an action given the chosen goal constraint h_g^k and its counterpart h_t^k, by `bind`ing h_t^k and h_g^k to the graph based on their state components and returning the action tagged to the edge between their respective nodes (Fig. 4.4d). If an edge does not exist between the inferred nodes, then we simply take a random action.

4.5 Experiments

We have proposed NCS as a solution to the object rearrangement problem that addresses two challenges: NCS addresses the correspondence problem by learning a factorized object-centric world model with dSLATE and it addresses the combinatorial problem by abstracting entity representations into a queryable state transition graph. Now we test NCS's efficacy in solving both problems.

The key question is whether NCS is better than state-of-the-art offline RL algorithms in generalizing over combinatorially-structured task spaces from perceptual input. As stated in §4.3, the crucial test for answering this question is to evaluate all methods on solving new rearrangement problems with a disjoint set of object configurations from those encountered during training. The most straightforward way to find a disjoint subset of the combinatorial space is to evaluate with a novel number of objects. We compare NCS to several offline RL baselines and ablations on two rearrangement environments and find a significant gap in performance between our method and the next best method.

Environments. In *block-rearrange* (Fig. 4.5a), all objects are the same size, shape, and orientation. $\mathcal{S}$ covers 16 locations in a grid. $\mathcal{Z}$ is the continuous space of red-green-blue values from 0 to 1. *robogym-rearrange* (Fig. 4.5b) is adapte from the OpenAI [239] rearrange environment and removes the assumptions from *block-rearrange* that all objects have the same size, shape, and orientation. The objects are uniformly sampled from a set of 94

meshes consisting of the YCB object set [44] and a set of basic geometric shapes, with colors sampled from a set of 13. Although locations are not pre-defined in *robogym-rearrange* as in *block-rearrange*, in practice there is a limit to the number of ways to arrange objects on the table to still be visible to the camera, which makes the bisimulation still a reasonable assumption here. For *block-rearrange* we use the SA attention mask α as the state s, and for *robogym-rearrange* we use the action-dependent part of the SA slot $\boldsymbol{\lambda}^s$ as the state s.

Experimental setup. We evaluate two settings: *complete* and *partial*. In the *complete* setting, the goal image shows all objects in new locations. The *partial* setting is underspecified: only a subset of objects have associated goal constraints (Fig. 4.5b). In *block-rearrange*, all constraints are unsatisfied in the start state. In *robogym-rearrange*, four constraints are unsatisfied in the start state. Our metric is the *fractional success rate*, the average change in the number of satisfied constraints divided by the number of initially unsatisfied constraints.

The experiences buffer consists of 5000 trajectories showing 4 objects. We evaluate on 4-7 objects for 100 episodes across 10 seeds. Even if we assume full access to the underlying state space, the task spaces are enormous: with $|S|$ object locations and k objects, the number of possible trajectories over object configurations of t timesteps is $\binom{|S|}{k} \times (k \times (|S| - k))^t$, which amounts to searching over more than 10^{16} possible trajectories for the complete specification setting of *block-rearrange* with $k = 7$ objects (see Appdx. C.5 for derivation). Our setting of assuming access to only pixels makes the problem even harder.

Baselines. The claim of this chapter are that, for object rearrangement, (1) object-centric methods fare better than monolithically-structured offline RL methods (2) non-parametric graph search fares better than parametric planning for object rearrangement and (3) a factorized graph search over state transitions of individual entities fares better than a non-factorized graph search over state transitions over entire entity-sets. To test (1), we compare with state-of-the-art pixel-based behavior cloning (BC) and implicit Q-learning (IQL) implementations based off of [183]. To test (2), we compare against a version of object-centric model predictive control (MPC) [319] that uses the cross entropy method over dSLATE rollouts. To test (3), we compare against an ablation (abbrv. NF, for "non-factorized") that constructs a graph with state transitions of entity-sets than of individual states. Our last baseline just takes random actions (Rand).

Results

Figure 4.1 shows that NCS performs significantly better than all baselines (about a 5-10x improvement), thereby refuting the alternatives to our claims. Most of the baselines perform no better or only slightly better than random. We observe that it is indeed difficult to perform shooting-based planning with an entity-centric world model trained to predict a single step forward [165]: the MPC baseline performs poorly because its rollouts are poor, and it is significantly more computationally expensive to run (11 hours instead of 20 minutes). We also observe that the NF ablation performs poorly, showing the importance of factorizing the non-parametric graph search. Additional results are in the Appendix.

Table 4.1: This table compares NCS with various baselines in the complete and partial evaluation settings of *block-rearrange* and *robogym-rearrange*. The methods were trained on 4 objects and evaluated on generalizing to 4, 5, 6, and 7 objects. We report the fractional success rate, with a standard error computed over 10 seeds.

(a) *block-rearrange*, complete specification.

Method	4	5	6	7
NCS (ours)	**0.94** ± 0.01	**0.93** ± 0.00	**0.93** ± 0.00	**0.89** ± 0.00
Rand	0.06 ± 0.02	0.07 ± 0.03	0.07 ± 0.03	0.08 ± 0.03
MPC	0.16 ± 0.06	0.12 ± 0.04	0.11 ± 0.04	0.10 ± 0.03
NF	0.07 ± 0.03	0.06 ± 0.02	0.07 ± 0.02	0.08 ± 0.03
IQL	0.07 ± 0.01	0.03 ± 0.00	0.02 ± 0.00	0.02 ± 0.00
BC	0.03 ± 0.00	0.02 ± 0.00	0.01 ± 0.00	0.01 ± 0.00

(b) *block-rearrange*, complete specification.

Method	4	5	6	7
NCS (ours)	**0.89** ± 0.01	**0.86** ± 0.01	**0.78** ± 0.01	**0.70** ± 0.01
Rand	0.06 ± 0.02	0.08 ± 0.03	0.08 ± 0.03	0.08 ± 0.03
MPC	0.13 ± 0.05	0.11 ± 0.04	0.10 ± 0.04	0.08 ± 0.03
NF	0.06 ± 0.03	0.07 ± 0.03	0.08 ± 0.03	0.07 ± 0.03
IQL	0.01 ± 0.01	0.07 ± 0.01	0.05 ± 0.01	0.05 ± 0.00
BC	0.05 ± 0.01	0.04 ± 0.00	0.03 ± 0.00	0.03 ± 0.00

(c) *robogym-rearrange*, complete specification.

Method	4	5	6	7
NCS (ours)	**0.64** ± 0.01	**0.47** ± 0.01	**0.49** ± 0.01	**0.41** ± 0.01
Rand	0.01 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
MPC	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
NF	0.01 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
IQL	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
BC	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

(d) *robogym-rearrange*, partial specification.

Method	4	5	6	7
NCS (ours)	**0.47** ± 0.01	**0.33** ± 0.01	**0.27** ± 0.01	**0.22** ± 0.01
Rand	0.005 ± 0.001	0.001 ± 0.00	0.002 ± 0.001	0.001 ± 0.00
MPC	0.00 ± 0.00	0.001 ± 0.001	0.00 ± 0.00	0.00 ± 0.00
NF	0.005 ± 0.001	0.001 ± 0.00	0.002 ± 0.001	0.001 ± 0.00
IQL	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
BC	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

Analysis

Having quantitatively shown the relative strength of NCS in combinatorial generalization from pixels, we now examine how our key design choices of (1) factorizing entity representations into action-invariant and action-dependent features and (2) querying a state transition graph constructed from action-dependent features contribute to NCS's behavior and performance. Is copying the entity type during latent prediction as dSLATE does sufficient for disentangling the location and appearance of objects into the state and type respectively? Does dSLATE learn to sparsely modify only the entity that corresponds to the moved object in the sensorimotor transition, such that the nodes of the state transition graph meaningfully can be reused across entities? These are nontrivial capabilities because NCS is self-supervised on only the experience buffer.

Fig. 4.4b, which visualizes the `align`, `select-constraint`, and `bind` functions of NCS on *robogym-rearrange*, suggests that, at least for the simplified setting we consider, the answer to both questions is yes. NCS has learned to represent different objects in different slots and construct a graph whose nodes capture location information. Fig. 4.6 shows a t-SNE [315] plot that clusters entities inferred from the *robogym* environment. Because we have not provided supervision on what states should represent, we observe there are multiple cluster indices that map onto similar groups of points. This reveals that multiple different regions of $\mathcal{S}$ appear to be modeling similar states. We also tried merging redundant clusters, but found that this did not improve quantitative performance.

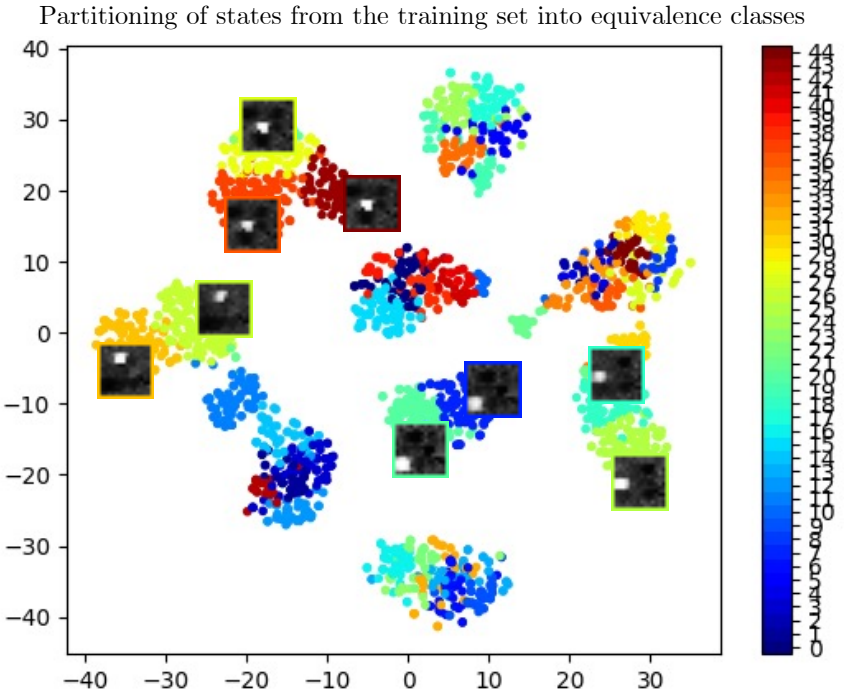


Figure 4.6: **Nodes as equivalent classes over states.** We show a clustering of states inferred for *robogym-rearrange*, where each cluster centroid is treated as a node in our transition graph. A subset of clusters are labeled with an attention mask computed by averaging the slot attention masks for the entities associated with the cluster.

4.6 Discussion

Object rearrangement offers an intuitive setting for studying how an agent can learn reusable abstractions from its sensorimotor experience. This chapter takes a first step toward connecting the world of symbolic planning with human-defined abstractions and the world of representation learning with deep networks by introducing NCS. NCS is a method for controlling an agent that plans over and acts with state transition graph constructed with entity representations learned from raw sensorimotor transitions, without any other supervision. We showed that factorizing the entity representation into action-invariant and action-dependent features are important for solving the correspondence and combinatorial problems that make the object rearrangement difficult, and enable NCS to significantly outperform existing methods on combinatorial generalization in object rearrangement. The implementation of NCS provides a proof-of-concept for how learning reusable abstractions might be done, which we hope inspires future work to engineer methods like NCS for real-world settings.

Chapter 5

Representing Virtual Transformations

5.1 Introduction

This chapter seeks to tackle the question of how to build machines that leverage prior experience to solve more complex problems than they have previously encountered. How does a learner represent prior experience? How does a learner apply what it has learned to solve new problems? Motivated by these questions, this chapter aims to formalize the idea of, as well as to develop an understanding of the machinery for, *compositional generalization* in problems that exhibit compositional structure. The solutions for such problems can be found by composing in sequence a small set of reusable partial solutions, each of which tackles a subproblem of a larger problem. The central contributions of this chapter are to frame the shared structure across multiple tasks in terms of a *compositional problem graph*, propose *compositional generalization* as an evaluation scheme to test the degree a learner can apply previously learned knowledge to solve new problems, and introduce the *compositional recursive learner*, a domain-general framework[1] for sequentially composing representation transformations that each solve a subproblem of a larger problem.

The key to our approach is recasting the problem of generalization as a problem of learning algorithmic procedures over representation transformations. A solution to a (sub)problem is a transformation between its input and output representations, and a solution to a larger problem composes these subsolutions together. Therefore, representing and leveraging prior problem-solving experience amounts to learning a set of reusable primitive transformations and their means of composition that reflect the structural properties of the problem distribution.

This chapter introduces the compositional recursive learner (CRL), a framework for learning both these transformations and their composition together with sparse supervision, taking a step beyond other approaches that have assumed either pre-specified transformation or composition rules (Sec. 5.5). CRL learns a modular recursive program that iteratively re-represents the input representation into more familiar representations it knows how to compute with. In this framework, a transformation between representations is encapsulated

[1] `https://github.com/mbchang/crl`

into a *computational module*, and the overall program is the sequential combination of the inputs and outputs of these modules, whose application are decided by a *controller*.

What sort of training scheme would encourage the spontaneous specialization of the modules around the compositional structure of the problem distribution? First, exposing the learner to a diverse distribution of compositional problems helps it pattern-match across problems to distill out common functionality that it can capture in its modules for future use. Second, enforcing that each module have only a local view of the global problem encourages task-agnostic functionality that prevents the learner from overfitting to the empirical training distribution; two ways to do this are to constrain the model class of the modules and to hide the task specification from the modules. Third, training the learner with a curriculum encourages the learner to build off old solutions to solve new problems by re-representing the new problem into one it knows how to solve, rather than learning from scratch.

How should the learner learn to use these modules to exploit the compositional structure of the problem distribution? We can frame the decision of which computation to execute as a reinforcement learning problem in the following manner. The application of a sequence of modules can be likened to the execution trace of the program that CRL automatically constructs, where a computation is the application of a module to the output of a previous computation. The automatic construction of the program can be formulated as the solution to a sequential decision-making problem in a meta-level Markov decision process (MDP) [148], where the state space is the learner's internal states of computation and the action space is the set of modules. Framing the construction of a program as a reinforcement learning problem allows us to use techniques in deep reinforcement learning to implement loops and recursion, as well as decide on which part of the current state of computation to apply a module, to re-use sub-solutions to solve a larger problem.

Our experiments on solving multilingual arithmetic problems and recognizing spatially transformed MNIST digits [193] show that the above proposed training scheme prescribes a type of *reformulation*: re-representing a new problem in terms of other problems by implicitly making an *analogy* between their solutions. We also show that our *meta-reasoning* approach for deciding what modules to execute achieves better generalization to more complex problems than monolithic learners that are not explicitly compositional.

5.2 Compositional Generalization

> Solving a problem simply means representing it so as to make the solution transparent.
>
> Simon [286]

Humans navigate foreign cities and understand novel conversations despite only observing a tiny fraction of the true distribution of the world. Perhaps they can extrapolate in this

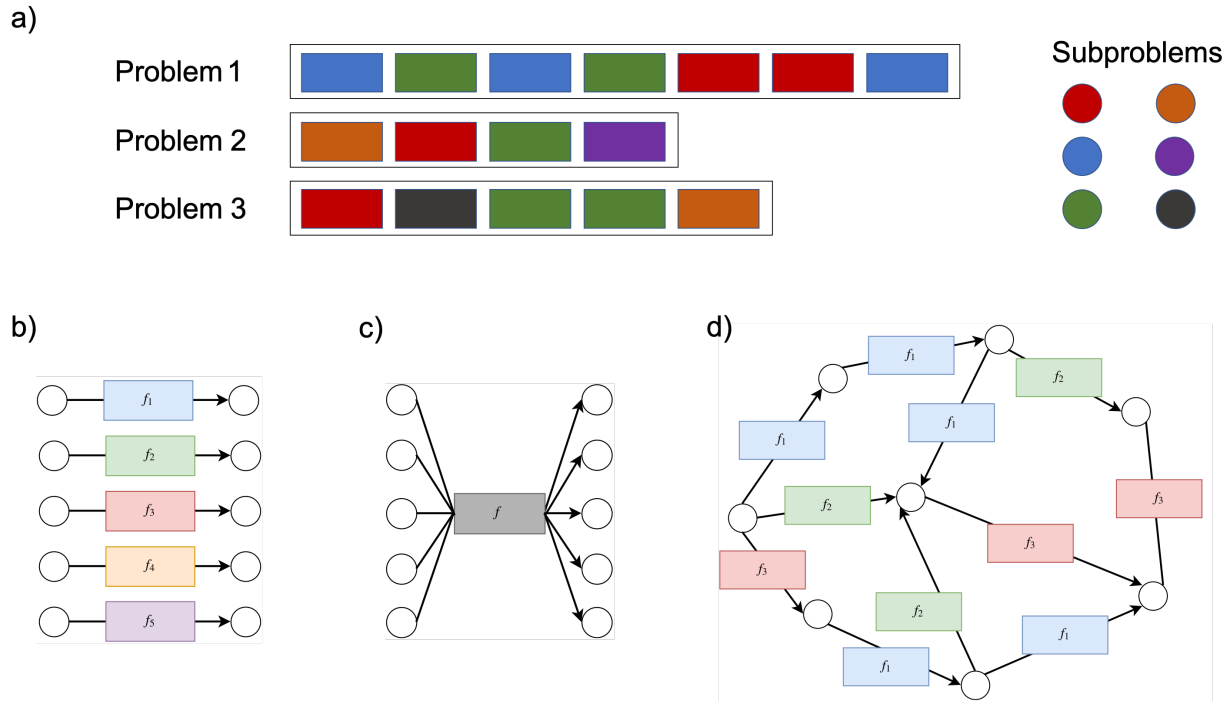


Figure 5.1: **(a)** Consider a multitask family of problems, whose subproblems are shared within and across problems. Standard approaches either **(b)** train a separate learner per task or **(c)** train a single learner for all tasks. Both have difficulty generalizing to longer compositional problems. **(d)** Our goal is to re-use previously learned sub-solutions to solve new problems by composing computational modules in new ways.

way because the world contains compositional structure, such that solving a novel problem is possible by composing previously learned partial solutions in a novel way to fit the context.

With this perspective, we propose the concept of *compositional generalization*. The key assumption of compositional generalization is that harder problems are composed of easier problems. The problems from the training and test sets share the same primitive subproblems, but differ in the manner and complexity with which these subproblems are combined. Therefore, problems in the test set can be solved by combining solutions learned from the training set in novel ways.

Definition. Let a *problem* P be a pair (X_{in}, X_{out}), where X_{in} and X_{out} are random variables that respectively correspond to the input and output representations of the problem. Let the distribution of X_{in} be r_{in} and the distribution of X_{out} be r_{out}. To solve a particular problem $P = p$ is to transform $X_{in} = x_{in}$ into $X_{out} = x_{out}$. A composite problem $p_a = p_b \circ p_c$ is that for which it is possible to solve by first solving p_c and then solving p_b with the output of p_c as input. p_b and p_c are subproblems with respect to p_a. The space of compositional problems form a *compositional problem graph*, whose nodes are the representation distributions r. A problem is described as pair of nodes between which the learner must learn to construct an

edge or a path to transform between the two representations.

Characteristics. First, there are many ways in which a problem can be solved. For example, translating an English expression to a Spanish one can be solved directly by learning such a transformation, or a learner could make an *analogy* with other problems by first translating English to French, and then French to Spanish as intermediate subproblems. Second, sometimes a useful (although not only) way to solve a problem is indicated by the *recursive* structure of the problem itself: solving the arithmetic expression $3 + 4 \times 7$ modulo 10 can be decomposed by first solving the subproblem $4 \times 7 = 8$ and then $3 + 8 = 1$. Third, because a problem is just an (input, output) pair, standard problems in machine learning fit into this broadly applicable framework. For example, for a supervised classification problem, the input representation can be an image and the output representation a label, and intermediate subproblems can be transforming some intermediate representations to other intermediate representations. Sec. 5.4 demonstrates CRL on all three of the above examples.

Broad Applicability. Problems in supervised, unsupervised, and reinforcement learning can all be viewed under the framework of transformations between representations. What we gain from the compositional problem graph perspective is a methodological way to relate together different problems of various forms and complexity, which is especially useful in a lifelong learning setting: the knowledge required to solve one problem is composed of the knowledge required to solve subproblems seen in the past in the context of different problems. For example, we can view latent variable reinforcement learning architectures such as [143, 228] as simultaneously solving an image reconstruction problem and an action prediction problem, both of which share the same subproblem of transforming a visual observation into a latent representation. Lifelong learning, then, can be formulated as not only modifying the connections between nodes in the compositional problem graph but also continuing to make more connections between nodes, gradually expanding the frontier of nodes explored. Sec. 5.4 describes how CRL takes advantage of this compositional formulation in a multi-task zero-shot generalization setup to solve new problems by re-using computations learned from solving past problems.

Evaluation. To evaluate a learner's capacity for compositional generalization, we introduce two challenges. The first is to generalize to problems with different subproblem combinations from what the learner has seen. The second is to generalize to problems with longer subproblems combinations than the learner has seen. Evaluating a learner's capability for compositional generalization is one way to measure how readily old knowledge can be reused and hence built upon.

5.3 A Learner That Programs Itself

This chapter departs from the popular *representation-centric* view of knowledge [31] and instead adopts a *computation-centric* view of knowledge: our goal is to encapsulate useful functionality shared across tasks into specialized *computational modules* – atomic function operators that perform transformations between representations. This section introduces the

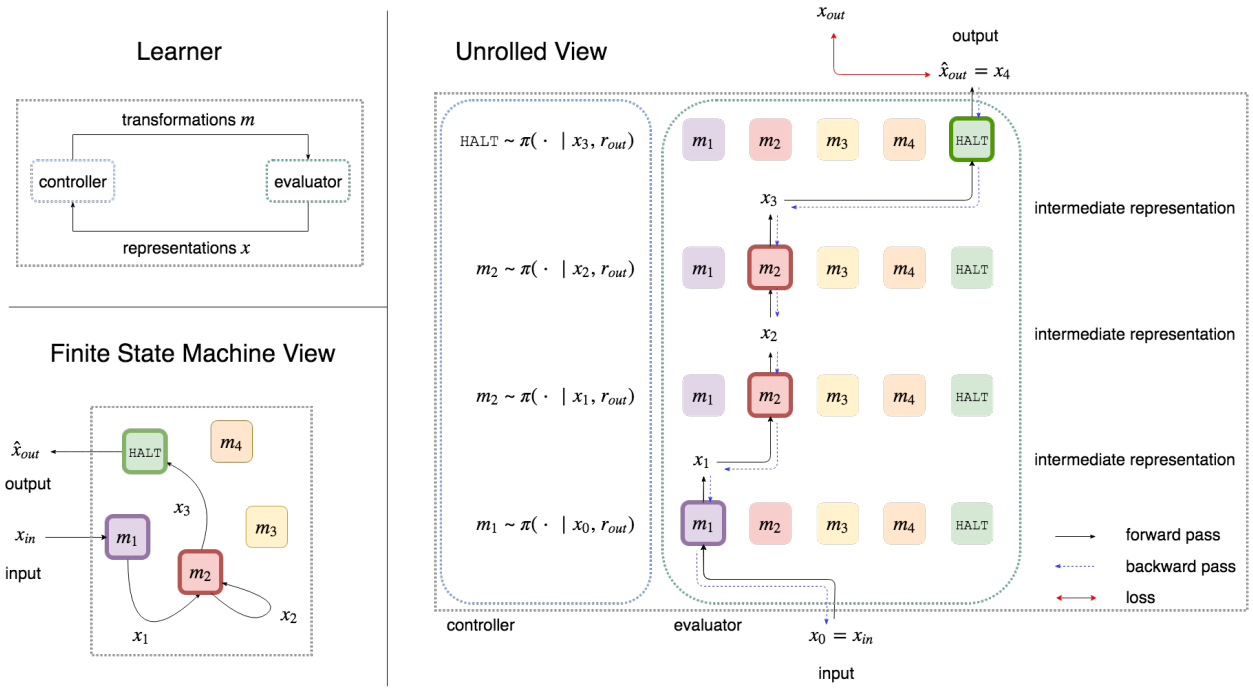


Figure 5.2: **Compositional recursive learner (CRL):** *top-left*: CRL is a symbiotic relationship between a controller and evaluator: the controller selects a module m given an intermediate representation x and the evaluator applies m on x to create a new representation. *bottom-left*: CRL learns dynamically learns the structure of a program customized for its problem, and this program can be viewed as a finite state machine. *right*: A series of computations in the program is equivalent to a traversal through a Meta-MDP, where module can be reused across different stages of computation, allowing for recursive computation.

compositional recursive learner (CRL), a framework for training modules to capture primitive subproblems and for composing together these modules as subproblem solutions to form a path between nodes of the compositional problem graph.

Compositional Recursive Learner

The CRL framework consists of a controller π, a set of modules $m \in M$, and an evaluator E. Training CRL on a diverse compositional problem distribution produces a modular recursive program that is trained to transform the input X_{in} into its output X_{out}, the corresponding samples of which are drawn from pairs of nodes in the compositional problem graph. In this program, the controller looks at the current state x_i of the program and chooses a module m to apply to the state. The evaluator executes the module on that state to produce the next state x_{i+1} of the program. X_{in} is the initial state of the program, $\hat{X}_{out}$ is the last, and the intermediate states X_i of the execution trace correspond to the other representations

produced and consumed by the modules. The controller can choose to re-use modules across different program executions to solve different problems, making it straightforward to re-use computation learned from solving other problems to solve the current one. The controller can also choose to reuse modules several times within the same program execution, which produces recursive behavior.

Deciding Which Computations To Execute

The sequential decision problem that the controller solves can be formalized as a meta-level Markov decision process (meta-MDP) [148], whose state space corresponds to the intermediate states of computation X, whose action space corresponds to the modules M, and whose transition model corresponds to the evaluator E. The symbiotic relationship among these components is shown in Fig. 5.2. In the *bounded-horizon* version of CRL (Sec. 5.4), the meta-MDP has a finite horizon whose length is determined by the complexity of the current problem. In the *infinite-horizon* version of CRL (Sec. 5.4), the program itself determines when to halt when the controller selects the `HALT` signal. When the program halts, in both versions the current state of computation is produced as output $\hat{x}_{out}$, and CRL receives a terminal reward that reflects how $\hat{x}_{out}$ matches the desired output x_{out}. The infinite-horizon CRL also incurs a cost for every computation it executes to encourage it to customize its complexity to the problem.

Note the following key characteristics of CRL. First, unlike standard reinforcement learning setups, the state space and action space can vary in dimensionality across and within episodes because CRL trains on problems of different complexity, reducing more complex problems to simpler ones (Sec. 5.4). Second, because the meta-MDP is internal to CRL, the controller shapes the meta-MDP by choosing which modules get trained and the meta-MDP in turn shapes the controller through its non-stationary state-distribution, action-distribution, and transition function. Thus CRL simultaneously designs and solves reinforcement learning problems "in its own mind," whose dynamics depend just as much on the intrinsic complexity of the problem as well as the current problem-solving capabilities of CRL.

Making Analogies in the Compositional Problem Graph

The solution that we want CRL to discover lies between two extremes, both of which have their own drawbacks. One extreme is where CRL learns a module specialized for every pair of nodes in the compositional problem graph, and the other is where CRL only learns one module for all pairs of nodes. Both extremes yield a horizon-one meta-MDP and are undesirable for compositional generalization: the former does not re-use past knowledge and the latter cannot flexibly continuously learn without suffering from negative transfer.

What is the best solution that CRL could discover? For a given compositional problem graph, an optimal solution would be to recover the original compositional problem graph such that the modules exactly capture the subproblems and the controller composes these modules to reflect how the subproblems were originally generated. By learning both the parameters

of the modules and the controller that composes them, during CRL would construct its own internal representation of the problem graph, where the functionality of the modules produces the nodes of the graph. How can we encourage CRL's internal graph to reflect the original compositional problem graph?

We want to encourage the modules to capture the most primitive subproblems, such that they can be composed as atomic computations for other problems. To do this, we need to enforce that each module only has a *local* view of the global problem. If tasks are distinguished from each other based on the input (see Sec. 5.4), we can use domain knowledge to restrict the representation vocabulary and the function class of the modules. If we have access to a task specification (e.g. goal or task id) in addition to the input, we can additionally give only the controller access to the task specification while hiding it from the modules. This forces the modules to be task agnostic, which encourages that they learn useful functionality that generalizes across problems.

Because the the space of subproblem compositions is combinatorially large, we use a curriculum to encourage solutions for the simpler subproblems to converge somewhat before introducing more complex problems, for which CRL can learn to solve by composing together the modules that had been trained on simpler problems. Lastly, to encourage the controller to generalize to new node combinations it has not seen, we train on a diverse distribution of compositional problems, such that the controller does not overfit to any one problem. This encourages controller to make analogies between problems during training by re-using partial solutions learned while solving other problems. Our experiments show that this analogy-making ability helps with compositional generalization because the controller solves new or more complex subproblem combinations by re-using modules that it learned during training.

5.4 Experiments

The main purpose of our experiments is to test the hypothesis that explicitly decomposing a learner around the structure of a compositional problem distribution yields significant generalization benefit over the standard paradigm of training a single monolithic architecture on the same distribution of problems. To evaluate compositional generalization, we select disjoint subsets of node pairs for training and evaluating the learner. Evaluating on problems distinct from those in training tests the learner's ability to *apply* what it has learned to new problems. To demonstrate the broad applicability of the compositional graph, we consider the structured symbolic domain of multilingual arithmetic and the underconstrained and high-dimensional domain of transformed-MNIST classification. We find that composing representation transformations with CRL achieves significantly better generalization when compared to generic monolithic learners, especially when the learner needs to generalize to problems with longer subproblem combinations than those seen during training.

In our experiments, the controller and modules begin as randomly initialized neural networks. The loss is backpropagated through the modules, which are trained with Adam

(a) Training (b) Different subproblem combinations (c) Longer subproblem combinations

Figure 5.3: **Multilingual Arithmetic (Quantitative).** CRL generalizes significantly better than the RNN, which, even with ten times more data, does not generalize to 10-length multilingual arithmetic expressions. Pretraining the RNN on domain-specific auxiliary tasks does not help the 10-length case, highlighting a limitation of using monolithic learners for compositional problems. By comparing CRL with a version trained without a curriculum ("No Curr": blue), we see the benefit of slowly growing the complexity of problems throughout training, although this benefit does not transfer to the RNN. The vertical black dashed line indicates at which point all the training data has been added when CRL is trained with a curriculum (red). The initial consistent rise of the red training curve before this point shows CRL exhibits forward transfer [209] to expressions of longer length. Generalization becomes apparent only after a million iterations after all the training data has been added. **(b, c)** only show accuracy on the expressions with the maximum length of those added so far to the curriculum. "1e4" and "1e5" correspond to the order of magnitude of the number of samples in the dataset, of which 70% are used for training. 10, 50, and 90 percentiles are shown over 6 runs.

[174]. The controller receives a sparse reward derived from the loss at the end of the computation, and a small cost for each computational step. The model is trained with proximal policy optimization [282].

Multilingual Arithmetic

This experiment evaluates the infinite-horizon CRL in a multi-objective, variable-length input, symbolic reasoning multi-task setting. A task is to simplify an arithmetic expression expressed in a *source* language, encoded as variable-length sequences of one-hot tokens, and produce the answer modulo 10 in a given *target* language. To evaluate compositional generalization, we test whether, after having trained on 46200 examples of 2, 3, 4, 5-length expressions ($2.76 \cdot 10^{-4}$ of the training distribution) involving 20 of the $5 \times 5 = 25$ pairs of five languages, the learner can generalize to 5-length and 10-length expressions involving the other five held-out language pairs (problem space: $4.92 \cdot 10^{15}$ problems). To handle the multiple target languages, the CRL controller receives a one-hot token for the target language at every computational step additional to the arithmetic expression. The CRL modules consist of two types of feedforward networks: reducers and translators, which do not know the target

language and so can only make local progress on the global problem. Reducers transform a consecutive window of three tokens into one token, and translators transform all tokens in a sequence by the same transformation. The CRL controller also selects where in the arithmetic expression to apply a reducer. We trained by gradually increasing the complexity of arithmetic expressions from length two to length five.

Quantitive results in Fig. 5.3 show that CRL achieves significantly better compositional generalization than a recurrent neural network (RNN) baseline [65] trained to directly map the expression to its answer, even when the RNN has been pretrained or receives 10x more data. Fig. D.3 shows that CRL achieves about 60% accuracy for extrapolating to 100-term problems (problem space: $4.29 \cdot 10^{148}$).

The curriculum-based training scheme encourages CRL to designs its own edges and paths to connect nodes in the compositional problem graph, solving harder problems with the solutions from simpler ones. It also encourages its internal representations to mirror the external representations it observes in the problem distribution, even though it has no direct supervision to do so. However, while this is often the case, qualitative results in Fig. 5.5 show that CRL also comes up with its own *internal* language – hybrid representations that mix different external representations together – to construct compositional solutions for novel problems. Rather than learn translators and reducers that are specific to single input and output language pair as we had expected, the modules, possibly due to their nonlinear nature, tended to learn operations specific to the output language only.

Image Transformations

This experiment evaluates the bounded-horizon CRL in a single-objective, latent-structured, high-dimensional multi-task setting. A task is to classify an MNIST digit, where the MNIST digit has been randomly translated (left, right, up, down), rotated (left, right), and scaled (small, big). Suppose CRL has knowledge of what untransformed MNIST digits look like; is it possible that CRL can learn to compose appropriate spatial affine transformations in sequence to convert the transformed MNIST digit into a "canonical" one, such that it can use a pre-trained classifier to classify it? To reformulate a scenario to one that is more familar is characteristic of compositional generalization humans: humans view an object at different angles yet understand it is the same object; they may have an accustomed route to work, but can adapt to a detour if the route is blocked. To evaluate compositional generalization, we test whether, having trained on images produced by combinations of two spatial transformations, CRL can can generalize to different length-2 combinations as well as length-3 combinations. A challenge in this domain is that the compositional structure is latent, rather than apparent in the input for the learner to exploit.

CRL is initialized with four types of modules: a Spatial Transformer Network (STN) [161] parametrized to only rotate, an STN that only scales, an STN that only translates, and an identity function. All modules are initialized to perform the identity transformation, such that symmetry breaking (and their eventual specialization) is due to the stochasticity of the controller.

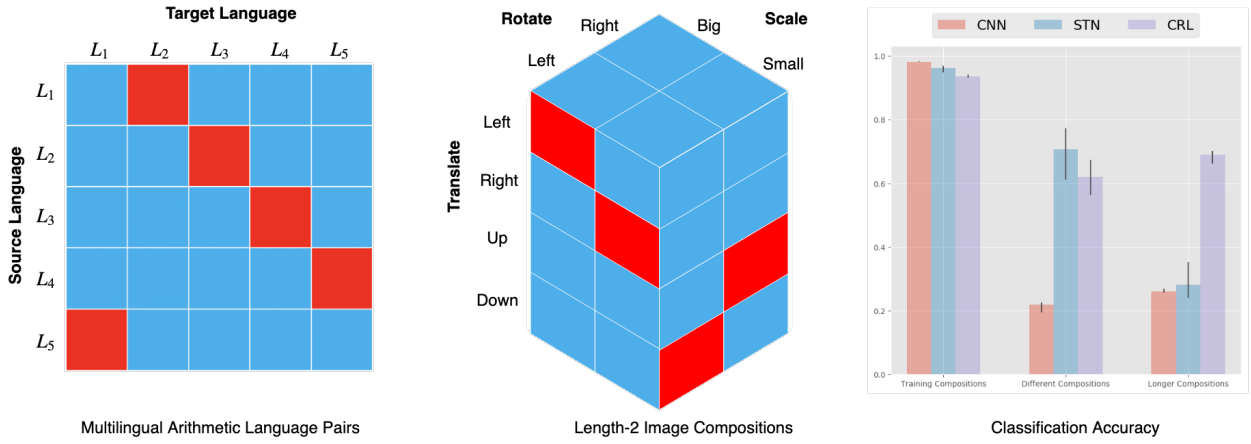


Figure 5.4: **Left:** For multilingual arithmetic, blue denotes the language pairs for training and red denotes the language pairs held out for evaluation in Fig 5.3b,c. **Center:** For transformed MNIST classification, blue denotes the length-2 transformation combinations that produced the input for training, red denotes the length-2 transformation combinations held out for evaluation. Not shown are the more complex length-3 transformation combinations (scale then rotate then translate) we also tested on. **Right:** For transformed MNIST classification, each learner performs better than the others in a different metric: the CNN performs best on the training subproblem combinations, the STN on different subproblem combinations of the same length as training, and CRL on longer subproblem combinations than training. While CRL performs comparably with the others in the former two metrics, CRL's $\sim 40\%$ improvement for more complex image transformations is significant.

Quantitative results in Fig. 5.4 show that CRL achieves significantly better compositional generalization than both the standard practice of finetuning the convolutional neural network [301] pretrained classifier and training an affine-STN as a pre-processor to the classifier. Both baselines perform better than CRL on the training set, and the STN's inductive bias surprisingly also allows it to generalize to different length-2 combinations. However, both baselines achieve only less than one-third of CRL's generalization performance for length-3 combinations, which showcases the value of explicitly decomposing problems. Note that in Fig. 5.6 the sequence of transformations CRL performs are not necessarily the reverse of those that generated the original input, which shows that CRL has learned its own *internal* language for representing nodes in the problem graph.

5.5 Related Work

Several recent and contemporaneous work [190, 206, 210, 19] have tested in whether neural networks exhibit *systematic compositionality* [97, 212, 96, 213, 45] in parsing symbolic data. This chapter draws inspiration from and builds upon research in several areas to propose an approach towards building a learner that exhibits compositional generalization. We hope this

chapter provides a point of unification among these areas through which further connections can be strengthened.

Compositional Generalization

Transformations between representations: Our work introduces a learner that exhibits compositional generalization in some sense by bridging deep learning and *reformulation*, or re-representing a problem to make it easier to solve [155, 287, 13] by making *analogies* [236] to previously encountered problems. Taking inspiration from *meta-reasoning* [266, 148, 146, 124] in humans [133, 43, 202], CRL generalize to new problems by composing representation transformations (analogous to the subprograms in Schmidhuber [278]), an approach for which recent and contemporaneous work [274, 9, 74] provide evidence.

Meta-learning: Our modular perspective departs from recent work in *meta-learning* [312, 275] which assume that the shared representation of monolithic architectures can be shaped by the diversity of tasks in the training distribution as good initializations for future learning [93, 232, 256, 15, 123, 224, 191, 101, 140, 141, 302].

Graph-based architectures: Work in graph-based architectures have studied *combinatorial generalization* in the context of modeling physical systems [28, 57, 27, 271, 270, 317]. Whereas these works focus on factorizing *representations*, we focus on factorizing the *computations* that operate on representations.

Neural program induction:

Just as the motivation behind disentangled representations [329, 185, 62, 311, 31, 151] is to uncover the latent factors of variation, the motivation behind disentangled programs is to uncover the latent organization of a task. Compositional approaches (as opposed to memory-augmented [125, 304, 167, 127, 188, 15, 126] or monolithic [348, 169] approaches for learning programs) to the challenge of discovering reusable primitive transformations and their means of composition generally fall into two categories. The first assumes pre-specified transformations and learns the structure (from dense supervision on execution traces to sparse-rewards) [257, 42, 339, 63, 107, 40, 91, 81, 349, 278]. The second learns the transformations but pre-specifies the structure [14, 259, 203]. These approaches are respectively analogous to our hardcoded-functions and hardcoded-controller ablations in Fig. D.1. The closest works to ours from a program induction perspective are [109, 314], both neurosymbolic approaches for learning differentiable programs integrated in a high-level programming language. Our work complements theirs by casting the construction of a program as a reinforcement learning problem, and we believe that more tightly integrating CRL with types and combinators would be an exciting direction for future work.

Self-organizing learners

Lifelong Learning: CRL draws inspiration from work [275, 71, 279, 277, 84] on learners that learn to design their own primitives and subprograms for solving an increasingly large number of tasks. The simultaneous optimization over the the continuous function parameters and their discrete compositional structure in CRL is inspired by the interplay between abstract and concrete knowledge that is hypothesized to characterize cognitive development: abstract structural priors serve as a scaffolding within which concrete, domain-specific learning takes place [298, 251], but domain-specific learning about the continuous semantics of the world can also provide feedback to update the more discrete structural priors [117, 46].

Hierarchy: Several works have investigated the conditions in which hierarchy is useful for humans [33, 296, 269]; our experiments show that the hierarchical structure of CRL is more useful than the flat structure of monolothic architectures for compositional generalization. Learning both the controller and modules relates CRL to the hierarchical reinforcement learning literature [25], where recent work [18, 186, 101, 320, 227] attempting to learn both lower-level policies as well as a higher-level policy that invokes them.

Modularity: Our idea of selecting different weights at different steps of computation is related to the fast-weights literature [276, 17], but those works are motivated by learning context-dependent associative memory [156, 332, 179, 12, 142] rather than composing representation transformations, with the exception of [273]. CRL can be viewed as a recurrent mixture of experts [160], where each expert is a module, similar to other recent and contemporaneous work [152, 260, 178, 90] that route through a choices of layers of a fixed-depth architecture for multi-task learning. The closest work to ours from an implementation perspective is Rosenbaum, Klinger, and Riemer [260]. However, these works do not address the problem of generalizing to more complex tasks because they do not allow for variable-length compositions of the modules. Parascandolo et al. [240] focuses on a complementary direction to ours; whereas they focus on learning causal mechanisms for a single step, we focus on learning how to compose modules. We believe composing together causal mechanisms would be an exciting direction for future work.

5.6 Discussion

This chapter sought to tackle the question of how to build machines that leverage prior experience to solve more complex problems than they have seen. This chapter makes three steps towards the solution. First, we formalized the compositional problem graph as a language for studying compositionally-structured problems of different complexity that can be applied on various problems in machine learning. Second, we introduced the compositional generalization evaluation scheme for measuring how readily old knowledge can be reused and hence built upon. Third, we presented the compositional recursive learner, a domain-general framework for learning a set of reusable primitive transformations and their means of

composition that reflect the structural properties of the problem distribution. In doing so we leveraged tools from reinforcement learning to solve a program induction problem.

There are several directions for improvement. One is to stabilize the simultaneous optimization between discrete composition and continuous parameters; currently this is tricky to tune. Others are to generate computation graphs beyond a linear chain of functions, and to infer the number of functions required for a family of problems. A major challenge would be to discover the subproblem decomposition without a curriculum and without domain-specific constraints on the model class of the modules.

Griffiths et al. [134] argued that the efficient use cognitive resources in humans may also explain their ability to generalize, and this chapter provides evidence that reasoning about what computation to execute by making analogies to previously seen problems achieves significantly higher compositional generalization than non-compositional monolithic learners. Encapsulating computational modules grounded in the subproblem structure also may pave a way for improving interpretability of neural networks by allowing the modules to be unit-tested against the subproblems we desire them to capture. Because problems in supervised, unsupervised, and reinforcement learning can all be expressed under the framework of transformations between representations in the compositional problem graph, we hope that our work motivates further research for tackling the compositional generalization problem in many other domains to accelerate the long-range generalization capabilities that are characteristic of general-purpose learning machines.

Figure 5.5: **Multilingual Arithmetic (Qualitative).** A randomly selected execution trace for generalizing from length-5 to length-10 expressions. The input is $0-6+1+7\times3\times6-3+7-7\times7$ expressed in Pig Latin. The desired output is *seis*, which is the value of the expression, 6, expressed in Spanish. The purple modules are reducers and the red modules are translators. The input to a module is highlighted and the output of the module is boxed. The controller learns order of operations. Observe that reducer m_9 learns to reduce to numerals and reducer m_{10} to English terms. The task-agnostic nature of the modules forces them to learn transformations that the controller would commonly reuse across problems. Even if the problem may not be compositionally structured, such as translating Pig Latin to Spanish, CRL learns to design a compositional solution (Pig Latin to Numerals to Spanish) from previous experience (Pig Latin to Numerals and Numerals to Spanish) in order to generalize: it first reduces the Pig Latin expression to a numerical evaluation, and then translates that to its Spanish representation using the translator m_6. Note that all of this computation is happening internally to the learner, which computes on softmax distributions over the vocabulary; for visualization we show the token of the distribution with maximum probability.

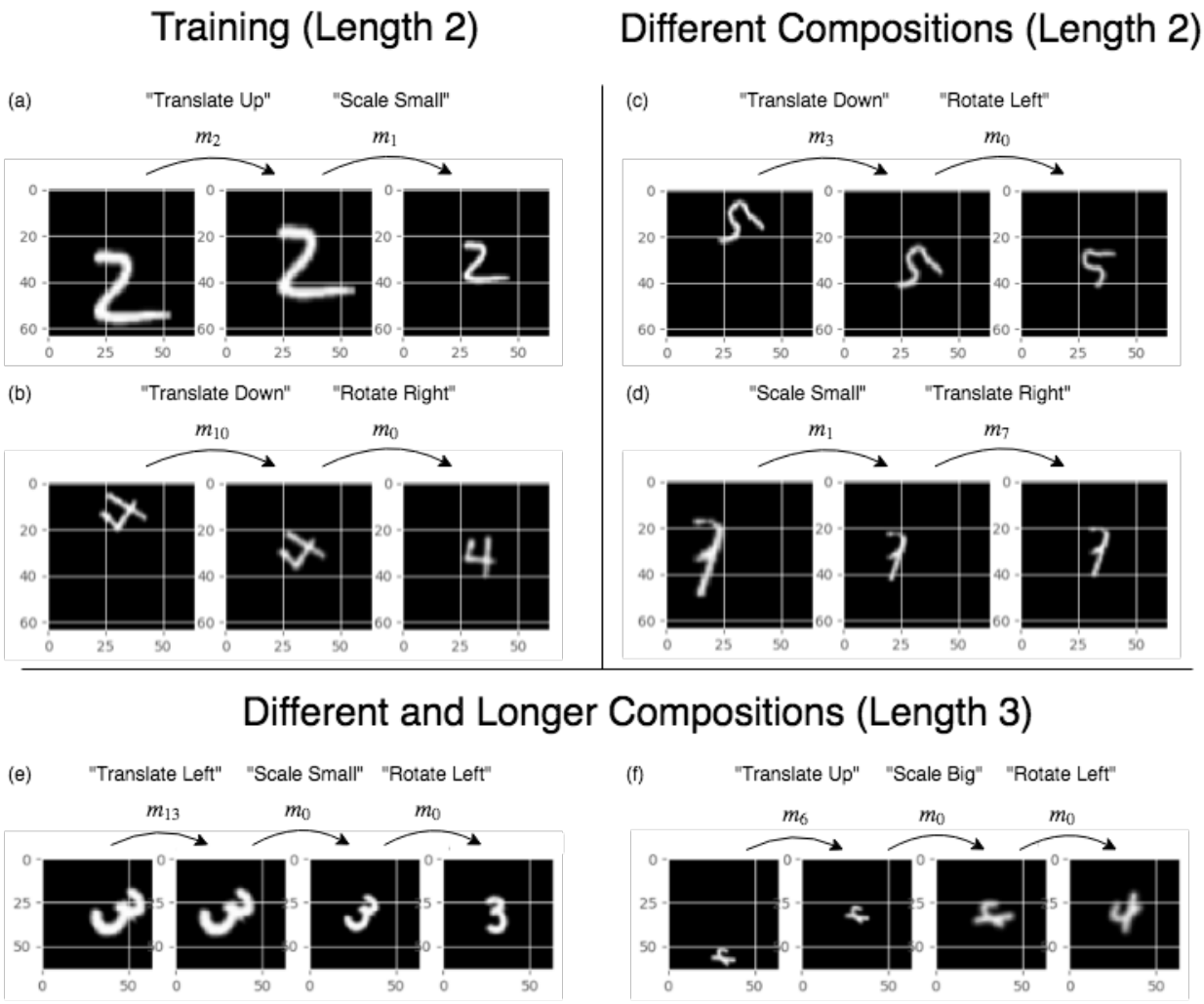


Figure 5.6: **Image Transformations:** CRL reasonably applies a sequence of modules to transform a transformed MNIST digit into canonical position, and generalizes to different and longer compositions of generative transformations. m_0 is constrained to output the sine and cosine of a rotation angle, m_1 is constrained to output the scaling factor, and m_2 through m_{13} are constrained to output spatial translations. Some modules like m_2 and m_6 learn to translate up, some like m_3 and m_{10} learn to translate down, some like m_7 learn to shift right, and some like m_{13} learn to shift left. Consider (d): the original generative transformations were "scale big" then "translate left," so the correct inversion should be "translate right" then "scale small." However, CRL chose to equivalently "scale small" and then "translate right." CRL also creatively uses m_0 to scale, as in (e) and (f), even though its original parametrization of outputting sine and cosine is biased towards rotation.

Part III

Choices

Chapter 6

Representing Policies as Games

> You know that everything you think and do is thought and done by you. But what's a "you"? What kinds of smaller entities cooperate inside your mind to do your work?
>
> Minsky [223]

6.1 Introduction

Biological processes, corporations, and ecosystems – physically decentralized, yet in some sense functionally unified. A corporation, for example, optimizes for maximizing profits as it were a single rational agent. But this agent abstraction is an illusion: the corporation is simply a collection of human agents, each solving their own optimization problems, most not even knowing the existence of many of their colleagues. But the human as the decision-making agent is also simply an abstraction of the trillions of cells making their own simpler decisions. *The society of agents is itself an agent.* What mechanisms bridge between these two levels of abstraction, and under what framework can we develop learning algorithms for studying the self-organizing nature of intelligent societies that pervade so much of the world?

Both the monolithic and the multi-agent optimization frameworks in machine learning offer a language for representing only one of the levels of abstraction but not the relation between both. The monolithic framework, the most commonly used in much of modern machine learning, considers a single agent that optimizes a single objective in an environment, whether it be minimizing classification loss or maximizing return. The multi-agent framework considers multiple agents that each optimize their own independent objective and each constitute each other's learning environments. What distinguishes the multi-agent from the monolithic is the presence of multiple independent optimization problems. The difficulty of interpreting a learner in the monolithic framework as a society of simpler components is that all components are still globally coupled together by the same optimization problem without

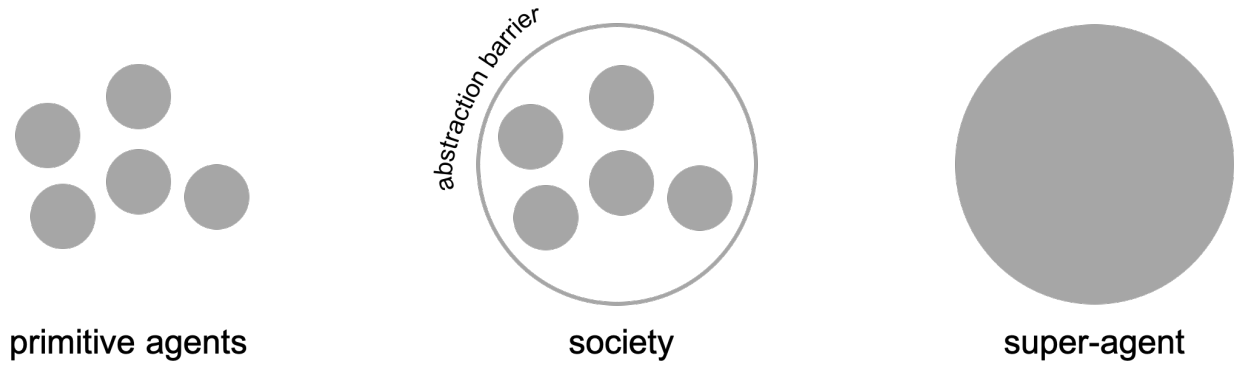


Figure 6.1: We study how a society of primitives agents can be abstracted as a super-agent. The incentive mechanism is the abstraction barrier that relates the optimization problems of the super-agent with those of its constituent primitive agents.

independent local optimization problems themselves, as are the weights in a neural network trained by backpropagation. The difficulty of interpreting a multi-agent system under a global optimization problem is the computational difficulty of computing Nash equilibrium [69], even for general two-player games [61].

To better understand the relationship between the society and the agent, this chapter makes four contributions, each at a different level of abstraction. At the highest level, we define the **societal decision-making** *framework* to relate the local optimization problem of the agent to the global optimization problem of the society in the following restricted setting. Each agent is specialized to transform the environment from one state to another. The agents bid in an auction at each state and the auction winner transforms the state into another state, which it sells to the agents at the next time-step, thereby propagating a series of economic transactions. This framework allows us to ask what are properties of the auction mechanism and of the society that enable the global solution to a Markov decision process (MDP) that the society solves to emerge implicitly as a consequence of the agents optimizing their own independent auction utilities.

At the second level, we present a *solution* to this question by introducing the **cloned Vickrey society** that guarantees that the dominant strategy equilibrium of the agents coincides with the optimal policy of the society. We prove this result by leveraging the truthfulness property of the Vickrey auction [321] and showing that initializing redundant agents makes the primitives' economic transactions robust against market bubbles and suboptimal equilibria.

At the third level, we propose a class of **decentralized reinforcement learning** *algorithms* for optimizing the MDP objective of the society as an emergent consequence of the agents' optimizing their own auction utilities. These algorithms treat the auction utility as optimization objectives themselves, thereby learning a societal policy that is global in space and time using only credit assignment for learnable parameters that is local in space

and time.

At the fourth level, we empirically investigate various *implementations* of the cloned Vickrey society under our decentralized reinforcement learning algorithm and find that a particular set of design choices, which we call the **credit conserving Vickrey** implementation, yields both the best performance at the societal and and agent level.

Finally, we demonstrate that the societal decision making framework, along with its solution, the algorithm that learns the solution, and the implementation of this algorithm, is a broadly applicable perspective on self-organization to not only standard reinforcement learning but also selecting options in semi-MDPs [309] and composing functions in dynamic computation graphs [58]. Moreover, we show evidence that the local credit assignment mechanisms of societal decision-making produce more efficient learning than the global credit assignment mechanisms of the monolithic framework.

6.2 Related Work

Describing an intelligent system as the product of interactions among many individual agents dates as far back as the Republic [252], in which Plato analyzes the human mind via an analogy to a political state. This theme continued into the early foundations of AI in the 1980s and 1990s through cognitive models such as the Society of Mind [223] and Braitenberg vehicles [36] and engineering successes in robotics [37] and in visual pattern recognition [283].

The closest works to ours were the algorithms developed around that same time period that sought as we do to leverage a multi-agent society for achieving a global objective, starting as early as the bucket brigade algorithm [154], in which agents bid in a first-price auction to operate on the state and auction winners directly paid their bid to the winners from the previous step. Prototypical self-referential learning mechanisms [275] improved the bucket brigade by imposing credit conservation in the economic transactions. The neural bucket brigade [280] adapted the bucket brigade to learning neural network weights, where payoffs corresponded to weight changes. Baum [29] observed that the optimal choice for an agent's bid should be equivalent to the optimal Q-value for executing that agent's transformation and developed the Hayek architecture for introducing new agents and removing agents that have gone broke. Kwee, Hutter, and Schmidhuber [189] added external memory to the Hayek architecture.

However, to this date there has been no proof to the best of our knowledge that the bid-updating schemes proposed in these works simultaneously optimize a global objective of the society in a decision-making context. Sutton [306] provides a convergence proof for temporal difference methods that share some properties with the bucket brigade credit assignment scheme, but importantly does not take the competition between the individual agents into account. But it is precisely the competition among agents in multi-agent learning that make their equilibria nontrivial to characterize [218]. Our work offers an alternative auction mechanism for which we prove that the optimal solution for the global objective *does* coincide with a Nash equilibrium of the society. We follow similar motivations to Balduzzi [22], which

investigates incentive mechanisms for training a society of rational discrete-valued neurons. In contrast to other works that decouple the computation graph [303, 122, 247, 244] but optimize a global objective, our work considers optimizing local objectives only. We consider economic transactions between time-steps, as opposed to within a single time-step [237].

6.3 Preliminaries

To set up a framework for societal decision-making, we relate Markov decision processes (MDP) and auctions under a unifying language. We define an **environment** as a tuple that specifies an input space, an output space, and additional parameters for specifying an objective. An **agent** is a function that maps the input space to the output space. An **objective** is a functional that maps the learner to a real number. Given an environment and objective, the **problem** the agent solves is to maximize the value of the objective.

In the MDP environment, the input space is the state space $\mathcal{S}$ and the output space is the action space $\mathcal{A}$. The agent is a policy $\pi : \mathcal{S} \to \mathcal{A}$. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and discount factor γ are additional parameters that specify the objective: the return $J(\pi) = \mathbb{E}_{\tau \sim p^\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t r\left(s_t, a_t\right) \right]$, where $p^\pi(\tau) = p(s_0) \prod_{t=0}^{T} \pi(a_t|s_t) \prod_{t=0}^{T-1} \mathcal{T}(s_{t+1}|s_t, a_t)$. The agent solves the problem of finding $\pi^* = \text{argmax}_\pi J(\pi)$. For any state s, the optimal action for maximizing $J(\pi)$ is $\pi^*(s) = \text{argmax}_a Q^*(s, a)$, where the optimal Q function $Q^*(s, a)$ is recursively defined as $Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{T}(s,a)} \left[r(s, a) + \gamma \max_{a'} Q^*(s', a') | s, a \right]$.

In the auction environments we consider, the input space is a single auction item s and the output space is the bidding space $\mathcal{B}$. Instead of a single agent, each of N agents $\psi^{1:N}$ compete to bid for the auction item via its bidding policy $\psi^i : \{s\} \to \mathcal{B}$. Let $\mathbf{b}$ be the vector of bids produced by $\psi^{1:N}$. The vector $\mathbf{v}_s$ of each agent's valuations for auction item s and the **auction mechanism** – allocation rule $X : \mathcal{B}^N \to [0, 1]^N$ and pricing rule $P : \mathcal{B}^N \to \mathbb{R}^N_{\geq 0}$ – are additional parameters that specify each agent's objective: the utility $U^i_s(\psi^{1:N}) = \mathbf{v}^i_s \cdot X^i(\mathbf{b}) - P^i(\mathbf{b})$, where $X^i(\mathbf{b})$ is the proportion of s allocated to i, and $P^i(\mathbf{b})$ is the scalar price i pays. Each agent i independently solves the problem of finding $\psi^{i*} = \text{argmax}_{\psi^i} U^i_s(\psi^{1:N})$. The independent optimization of objectives distinguishes a multi-agent problem from a single-agent one and makes multi-agent problems generally difficult to analyze when an agent's optimal policy depends on the strategies of other agents.

However, if an auction is dominant strategy incentive compatible (DSIC), bidding one's own valuation is optimal, independent of other players' bidding strategies. That is, truthful bidding is the unique dominant strategy. Notably, the **Vickrey auction** [321], which sets $P^i(\mathbf{b})$ to be the second highest bid $\max_{j \neq i} \mathbf{b}^j$ and $X^i(\mathbf{b}) = 1$ if i wins and 0 and 0 respectively if i loses, is DSIC, which means the dominant strategy equilibrium occurs when every agent bids truthfully, making the Vickrey auction straightforward to analyze. Another attractive property of the Vickrey auction is that the dominant strategy equilibrium automatically maximizes the social welfare $\sum_{i=1}^{N} \mathbf{v}^i \cdot X^i(\mathbf{b})$ [263], which selects the bidder with the highest

valuation as winner. The existence of dominant strategies in the Vickrey auction removes the need for agents to recursively model others, giving the Vickrey auction the practical benefit of running in linear time [263].

6.4 Societal Decision-Making

The perspective of this chapter is that a society of agents can be abstracted as an agent that itself solves an optimization problem at a global level as an emergent consequence of the optimization problems its constituent agents solve at the local level. To make this abstraction precise, we now introduce the **societal decision-making** framework for analyzing and developing algorithms that relate the global decision problem of a society to the local decision problems of its constituent agents. We use **primitive** and **society** to distinguish between the agents at the local and global levels, respectively, which we define in the context of their local and global environments and objectives:

Definition 6.4.1. *A **primitive** ω is a tuple $(\psi, \phi_{\mathcal{T}})$ of a bidding policy $\psi : \mathcal{S} \to \mathcal{B}$ and transformation $\phi_{\mathcal{T}} : \mathcal{S} \to \mathcal{S}$.*

Definition 6.4.2. *A **society** Ω is a set of primitives $\omega^{1:N}$.*

The **global environment** is an MDP that we call the **global MDP**, with state space $\mathcal{S}$ and discrete action space $\mathcal{A} = \{1, ..., N\}$ that indexes the primitives $\omega^{1:N}$. The **local environment** is an auction that we call the **local auction** with auction item $s \in \mathcal{S}$ and bidding space $\mathcal{B} = [0, \infty)$.

The connection between the local and global environments is as follows. Each state in the global MDP is an auction item for a different local auction. The winning primitive $\hat{\omega}$ of the auction at state s transforms s into the next state s' of the global MDP using its transformation $\phi_{\mathcal{T}}$, parameterized by the global MDP's transition function $\mathcal{T}$. For each primitive i at each state s, its **local objective** is the utility $U_s^i(\psi^{1:N})$. Its **local problem** is to maximize $U_s^i(\psi^{1:N})$. The **global objective** is the return $J(\pi_\Omega)$ in the global MDP of the **global policy** π_Ω. The **global problem** for the society is to maximize $J(\pi_\Omega)$. We define the optimal **societal Q function** $Q_\Omega^*(s, \omega)$ as the expected return received from ω invoking its transformation $\phi_{\mathcal{T}}$ on s and the society activating primitives optimally with respect to $J(\pi_\Omega)$ afterward.

Since all decisions made at the societal level are an emergent consequence of decisions made at the primitive level, the societal decision-making framework is a self-organization perspective on a broad range of sequential decision problems. If each transformation $\phi_{\mathcal{T}}$ specifies a literal action, then societal decision-making is a decentralized re-framing of standard reinforcement learning (RL). Societal decision-making also encompasses the decision problem of choosing $\phi_{\mathcal{T}}$s as options in semi-MDPs [309] as well as choosing $\phi_{\mathcal{T}}$s as functions in a computation graph [58, 261, 9].

We are interested in auction mechanisms and learning algorithms for optimizing the global objective as an emergent consequence of optimizing the local objectives. Translating problems from one level of abstraction to another would provide a recipe for engineering a multi-agent system to achieve a desired global outcome and permit theoretical expectations on the nature of the equilibrium of the society, while giving us free choice on the architectures and learning algorithms of the primitive agents. To this end, we next present an auction mechanism for which the dominant strategy equilibrium of the primitives coincides with the optimal policy of the society, which we develop into a class of decentralized RL algorithms in later sections.

6.5 Mechanism Design for the Society

We first observe that to produce the optimal global policy, the optimal bidding strategy for each primitive at each local auction must be to bid their societal Q-value. By defining each primitive's valuation of a state as its optimal societal Q-value at that state, we show that the Vickrey auction ensures the dominant strategy equilibrium profile of the primitives coincides with the optimal global policy. Then we show that a market economy perspective on societal decision-making overcomes the need to assume knowledge of optimal Q-values, although weakens the dominant strategy equilibrium to a Nash equilibrium. Lastly, we explain that adding redundant primitives to the society mitigates market bubbles by enforcing credit conservation. Proofs are in the Appendix.

Optimal Bidding

We state what was observed informally in [29]:

Proposition 6.5.1. *Assume at each state s the local auction allocates $X^i(\mathbf{b}) = 1$ if i wins and $X^i(\mathbf{b}) = 0$ if i loses. Then all primitives ω^i bidding their optimal societal Q-values $Q^*_\Omega(s, \omega^i)$ collectively induce an optimal global policy.*

This proposition makes the problem of self-organization concrete: getting the optimal behavior in the global MDP to emerge from the optimal behavior in the local auctions can be reduced to incentivizing the primitives to bid their optimal societal Q-value at every state.

Dominant Strategies for Optimal Bidding

To incentivize the primitives to bid optimally, we propose to define the primitives' valuations $\mathbf{v}^i_s$ for each state s as their optimal societal Q-values $Q^*_\Omega(s, \omega^i)$ and use the Vickrey auction mechanism for each local auction.

Theorem 6.5.2. *If the valuations $\mathbf{v}^i_s$ for each state s are the optimal societal Q-values $Q^*_\Omega(s, \omega^i)$, then the society's optimal global policy coincides with the primitives' unique dominant strategy equilibrium under the Vickrey mechanism.*

Then, the utility $U_s^i(\psi^{1:N})$ at each state s that induces the optimal global policy, which we refer equivalently as $\hat{U}_s^i(\omega^{1:N})$ for the winning primitive $\hat{\omega}^i$ and $U_s^j(\omega^{1:N})$ for losing primitives ω^j, is given by

$$\hat{U}_s^i(\omega^{1:N}) = Q_\Omega^*(s, \hat{\omega}^i) - \max_{j \neq i} \mathbf{b}_s^j \tag{6.1}$$

and by $U_s^j(\omega^{1:N}) = 0$ for losing primitives.

Economic Transactions for Propagating Q_Ω^*

We have so far defined optimal bidding with respect to societal decision-making and characterized the utilities as functions of Q_Ω^* for which such bidding is a dominant strategy. We now propose to redefine the utilities without knowledge of Q_Ω^* by viewing the society as a market economy.

Monolithic frameworks for solving MDPs, such as directly optimizing the policy $J(\pi)$ with policy gradient methods, are analogous to **command-economies**, in which all production – the transformation of past states s_t into future states s_{t+1} – and wealth distribution – the credit assignment of reward signals to parameters – derive directly from single central authority – the MDP objective. In contrast, under the societal decision-making framework, the optimal global policy does *not* derive directly from the MDP objective, but rather emerges implicitly as the equilibrium of the primitives optimizing their own local objectives. We thus redefine the valuations $\mathbf{v}_s^i$ following the analogy of a **market economy**, in which production and wealth distribution are governed by the economic transactions between the primitives.

Specifically, we couple the local auctions at consecutive time-steps in the same game by defining the valuation $\mathbf{v}_{s_t}^i$ of primitive $\hat{\omega}^i$ for winning the auction item s_t as the revenue it can receive in the auction at the next time-step by selling the product s_{t+1} of executing its transformation $\phi_\mathcal{T}^i$ on s_t. This compensation comes as the environment reward plus the (discounted) winning bid at the next time-step:

$$\underbrace{\hat{U}_{s_t}^i(\omega^{1:N})}_{\text{utility}} = \underbrace{r(s_t, \hat{\omega}^i) + \gamma \cdot \max_k \mathbf{b}_{s_{t+1}}^k}_{\text{revenue, or valuation } \mathbf{v}_{s_t}^i} - \underbrace{\max_{j \neq i} \mathbf{b}_{s_t}^j}_{\text{price}}. \tag{6.2}$$

Analogous to a market economy, the revenue $\hat{\omega}^i$ receives for producing s_{t+1} from s_t depends on the price the winning primitive $\hat{\omega}^k$ at $t+1$ is willing to bid for s_{t+1}. In turn, $\hat{\omega}^k$ sells s_{t+2} to the winning primitive at $t+2$, and so on. Ultimately currency is grounded in the reward. Wealth is distributed based on what future primitives decide to bid for the fruits of the labor of information processing carried out by past primitives transforming one state to another.

Definition 6.5.1. *A **Market MDP** is a global MDP in which all utilities are defined as in Equation 6.2.*

As valuations now depend on the strategies of future primitives, the dominant strategy equilibrium from Theorem 6.5.2 must be weakened to a Nash equilibrium in the general case:

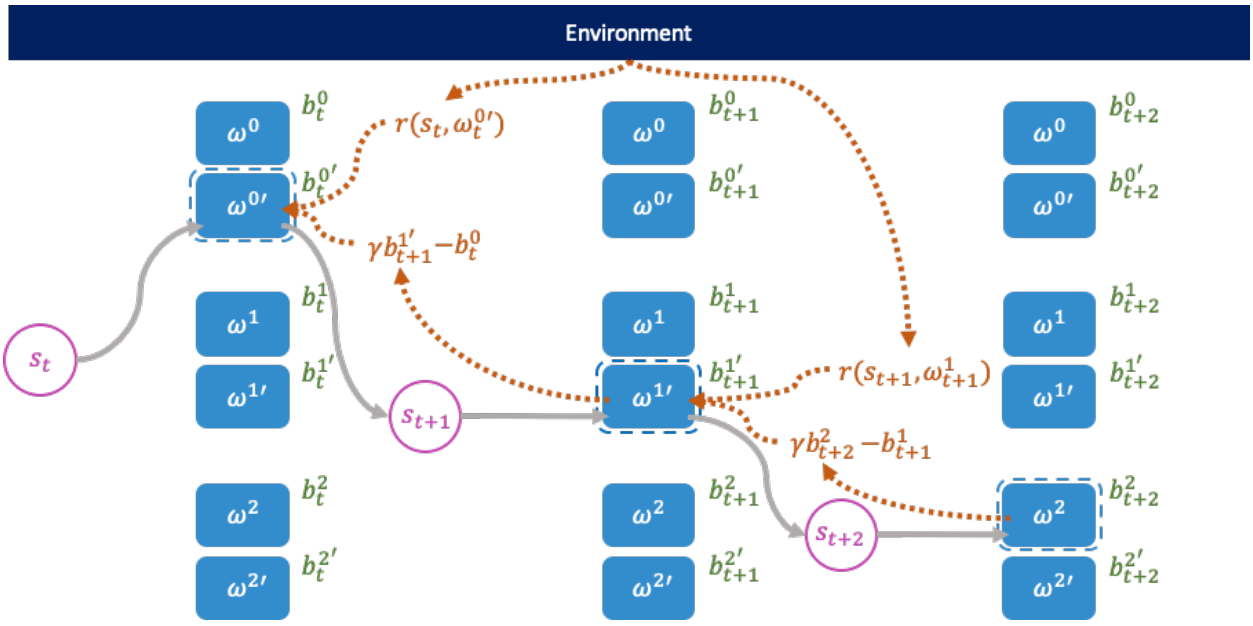


Figure 6.2: **The cloned Vickrey society.** In this market economy of primitive agents, wealth is distributed not directly from the global MDP objective but based on what future primitives decide to bid for the fruits of the labor of information processing carried out by past primitives transforming one state to another. The primitive $\omega_t^{0'}$ that wins the auction at time t receives an environment reward $r(s_t, \omega_t^{0'})$ as well as payment $b_{t+1}^{1'}$ from $\omega_{t+1}^{1'}$ for transforming s_t to s_{t+1}. By the Vickrey auction, the price $\omega_{t+1}^{1'}$ pays to transform s_{t+1} is the second highest bid b_{t+1}^1 at time $t+1$. Because each primitive ω^i and its clone $\omega^{i'}$ have the same valuations, their bids are equivalent and so credit is conserved through time.

Proposition 6.5.3. *In a Market MDP, it is a Nash equilibrium for every primitive to bid $Q_\Omega^*(s, \omega^i)$. Moreover, if the Market MDP is finite horizon, then bidding $Q_\Omega^*(s, \omega^i)$ is the unique Nash equilibrium that survives iterated deletion of weakly dominated strategies.*

Redundancy for Credit Conservation

In general, credit is not conserved in the Market MDP: the winning primitive at time $t-1$ gets paid an amount equal to the highest bid at time t, but the winner at time t only pays an amount equal to the second highest bid at time t. At time t, ω^i could bid arbitrarily higher than $\max_{j \neq i} \mathbf{b}^j$ without penalty, which distorts the valuations for primitives that bid before time t, creating a market bubble.

To prevent market bubbles, we propose a modification to the society, which we will call a **cloned society**, for enforcing credit conservation – for all transformations $\phi_\mathcal{T}$ initialize at least two primitives that share the same $\phi_\mathcal{T}$:

Lemma 6.5.4. *For a cloned society, at the Nash equilibrium specified in Proposition 6.5.3, what the winning primitive $\hat{\omega}^i$ at time t receives from the winning primitive $\hat{\omega}^k$ at $t+1$ is exactly what $\hat{\omega}^k$ pays: $\mathbf{b}^k_{s_{t+1}}$.*

We now state our main result.

Theorem 6.5.5. *Define a **cloned Vickrey society** as a cloned society that solves a Market MDP. Then it is a Nash equilibrium for every primitive in the cloned Vickrey society to bid $Q^*_\Omega(s, \omega^i)$. In addition, the price that the winning primitive pays for winning is equivalent to what it bid.*

The significance of Theorem 6.5.5 is that guaranteeing truthful bidding of societal Q-values decouples the analysis of the local problem within each time-step from that of the global problem across time-steps, which opens the possibility for designing learners to reach a Nash equilibrium that we know exists. Without such a separation of these two levels of abstraction the entire society must be analyzed as a repeated game through time – a non-trivial challenge.

6.6 From Equilibria to Learning Objectives

So far the discussion has centered around the quality of the equilibria, which assumes the primitives know their own valuations. We now propose a class of decentralized RL algorithms for learning optimal bidding behavior without assuming knowledge of the primitives' valuations.

Instead of learning to optimize the MDP objective directly, we propose to train the primitives' parameterized bidding policies to optimize their utilities in Equation 6.2, yielding a class of decentralized RL algorithms for optimizing the global RL objective that is agnostic to the choice of RL algorithm used to train each primitive. By Theorem 6.5.5, truthful bidding for all agents is one global optimum to all agent's local learning problems that also serves as a global optimum for the society as whole. In the special case where the transformation $\phi_\mathcal{T}$ is a literal action, this class of decentralized RL algorithms can serve as an alternative to any standard algorithm for solving discrete-action MDPs. An on-policy learning algorithm is presented in Appendix E.4.

Local Credit Assignment in Space and Time

The global problem requires a solution that is global in space, because the society must collectively work together, and global in time, because the society must maximize expected return across time-steps. But an interesting property of using the auction utility in Equation 6.2 as an RL objective is that given redundant primitives it takes the form of the Bellman equation, thereby implicitly coupling all primitives together in space and time. Thus each primitive need only optimize for its immediate utility at a each time-step without needing to optimize for its own future utilities. This class of decentralized RL algorithms thus implicitly finds a

global solution in space and time using only credit assignment that is *local in space*, because transactions are only between individual agents, and *local in time*, because each primitive need only solve a contextual bandit problem at each time-step.

Redundancy for Avoiding Suboptimal Equilibria

A benefit of casting local auction utilities as RL objectives is the practical development of learning algorithms that need not assume oracle knowledge of valuations, but unless care is taken with each primitives' learning environments, the society may not converge to the globally optimal Nash equilibrium described in Section 6.5. As an example of a suboptimal equilibrium, in a Market MDP with two primitives, even if $\mathbf{v}^1 = 1, \mathbf{v}^2 = 2$, it is a Nash equilibrium for $\mathbf{b}^1 = 100, \mathbf{b}^2 = 0$. Without sufficient competitive pressure to not overbid or underbid, a rogue winner lacks the risk of losing to other primitives with similarly close valuations. Fortunately, the redundancy of a cloned Vickrey society serves a dual purpose of not only preventing market bubbles but also introducing competitive pressure in the primitives' learning environments in the form of other clones.

6.7 Experiments

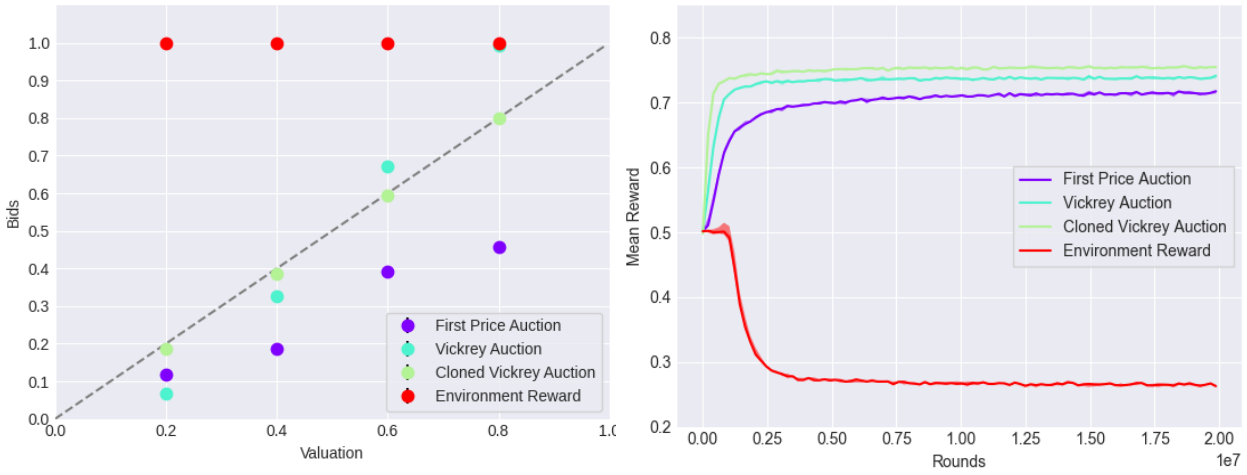
Now we study how well the cloned Vickrey society can recover the optimal societal Q-function as its Nash equilibrium. We compare several implementations of the cloned Vickrey society against baselines across simple tabular environments in Section 6.7, where the transformations $\phi_{\mathcal{T}}$ are literal actions. Then in Section 6.7 we demonstrate the broad applicability of the cloned Vickrey society for learning to select options in semi-MDPs and composing functions in a computation graph. Moreover, we show evidence for the advantages of the societal decision making framework over its monolithic counterpart in transferring to new tasks.

Although our characterization of the equilibrium and the learning algorithm are agnostic to the implementation of the primitive, in our experiments the bidding policy ψ is implemented as a neural network that maps the state to the parameters of a Beta distribution, from which a bid is sampled. We used proximal policy optimization (PPO) [282] to optimize the bidding policy parameters.

Numerical Simulations

In the following simulations we ask: (1) How closely do the bids the primitives learn match their optimal societal Q-values? (2) Does the solution to the global objective emerge from the competition among the primitives? (3) How does redundancy affect the solutions the primitives converge to?

We first consider the *Market Bandit*, the simplest global MDP with one state, and compare the cloned Vickrey society with societies with different auction mechanisms. Next we consider the *Chain* environment, a sparse-reward multi-step global MDP designed to study market



(a) (b)

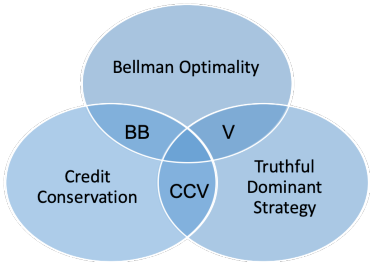
Figure 6.3: **Market Bandits.** We compare the cloned Vickrey society (Cloned Vickrey Auction) against solitary societies that use the first-price auction mechanism (first price auction), the Vickrey auction mechanism (Vickrey Auction), and a mechanism whose utility is only the environment reward (Environment Reward). The dashed line in (a) indicates truthful bidding of valuations. The cloned Vickrey society's bids are closest to the true valuations which also translates into the best global policy (b).

bubbles. Last we consider the *Duality* environment, a multi-step global MDP designed to study suboptimal equilibria. We use *solitary society* to refer to a society without redundant clones.

Market Bandits and Auction Mechanisms

The simulation primarily studies question (1) by comparing the auction mechanism of the cloned Vickrey society against other mechanisms in eliciting bids that match optimal societal Q-values. We compare against *first price auction*, based on Holland [154], a solitary society that uses the first price auction mechanism for the local auction, in which the winning primitive pays a price of their bid, rather the second highest bid; against *Vickrey Auction*, a solitary version of the cloned Vickrey society; and against *Environment Reward*, a baseline solitary society whose utility function uses only the environment reward, with no price term $P^i(\mathbf{b})$.

The environment is a four-armed Market Bandit whose arms correspond to transformations ϕ that deterministically yield reward values of 0.2, 0.4, 0.6, and 0.8. Solitary societies thus would have four primitives, while cloned societies eight. Since we are mainly concerned with question (1), we stochastically drop out a subset of primitives at every round to give each primitive a chance to win and learn its value.



	$\hat{\omega}_t$ Receives	$\hat{\omega}_{t+1}$ Pays
CCV	$\mathbf{b}'_{t+1}$	$\mathbf{b}'_{t+1}$
BB	$\hat{\mathbf{b}}_{t+1}$	$\hat{\mathbf{b}}_{t+1}$
V	$\hat{\mathbf{b}}_{t+1}$	$\mathbf{b}'_{t+1}$

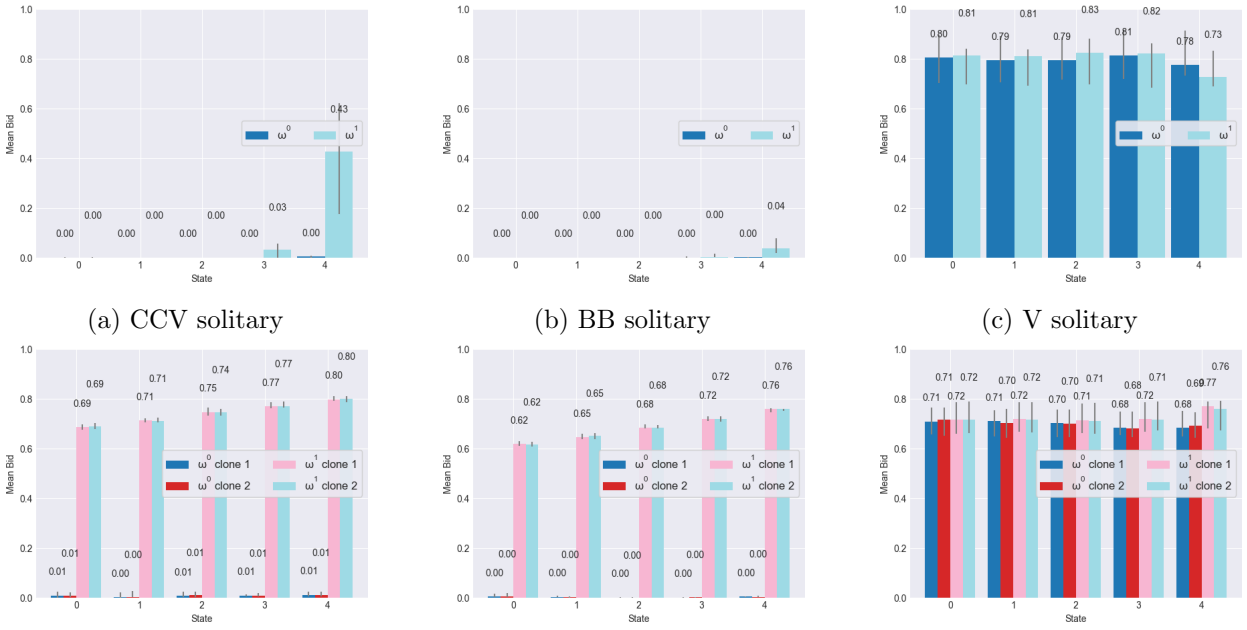
Figure 6.4: **Implementations.** The table shows the bid price that temporally consecutive winners $\hat{\omega}_{t+1}$ and $\hat{\omega}_t$ pay and receive based on three possible implementations of the cloned Vickrey society: *CCV*, *BB*, *V*, with tradeoffs depicted in the Venn diagram. We use $\hat{\mathbf{b}}_{t+1}$ and $\mathbf{b}'_{t+1}$ to denote the highest and second highest bids at time $t+1$ respectively.

In the Market Bandit, the arm rewards directly specify the primitives' valuations, so if the primitives successfully learned their valuations, we would expect them to learn to bid values exactly at 0.2, 0.4, 0.6, and 0.8. Figure 6.3a shows that the primitives in the cloned Vickrey society learn to bid most closely to their true valuations, which also translates into the best global policy in Figure 6.3b, answering question (2). To address question (3), we observe that the solitary Vickrey society's bids are more spread out than those of the cloned Vickrey society because there is no competitive pressure to learn the valuations exactly. This is not detrimental when the global MDP has only one time-step, but the next section shows that the lack of redundancy creates market bubbles that result in a suboptimal global policy.

Market MDPs

Now we consider MDPs that involve multiple time-steps to understand how learning is affected when primitives' valuations are defined by the bids of future primitives. Redundancy theoretically makes what the winner $\hat{\omega}_t$ receives and $\hat{\omega}_{t+1}$ equivalent, but the stochasticity of the bid distribution yields various possible implementations for the cloned Vickrey society in the market MDP– *bucket brigade* (*BB*), *Vickrey* (*V*), and *credit conserving Vickrey* (*CCV*), summarized in Figure 6.4, with different pros and cons. Note that the solitary bucket brigade society is a multiple time-step analog of the *first price auction* in Section 6.7. We aim to empirically test which of the implementations of the cloned Vickrey society yields best local and global performance.

The *Chain* environment and market bubbles. A primary purpose of the *Chain* environment (Figure 6.6a) is to study the effect of redundancy on mitigating market bubbles in the bidding behavior and how that affects global optimality. The *Chain* environment is a sparse reward finite-horizon environment which initializes the society at s_0 on the left, yields a terminal reward of 0.8 if the society enters the goal state s_5 on the right, and ends the episode after 20 steps if the society does not reach the goal. *Chain* thus tests the ability



(a) CCV solitary (b) BB solitary (c) V solitary

(d) CCV clone (e) BB clone (f) V clone

Figure 6.5: **Learned Bidding Strategies for *Chain*.** We organize the analysis by distinguishing between the credit-conserving (CCV and BB) and the non-credit-conserving (V) implementations. The solitary CCV (a) and BB (b) implementations learn to bid very close to 0: CCV because the valuation for a primitive at t is only the second-highest bid at $t+1$, resulting in a rapid decay in the valuations leftwards down the chain; BB because each primitive is incentivized to pay as low of a price for winning as possible. The cloned CCV (d) and BB (e) implementations learn to implement a form of return decomposition [16] that redistributes the terminal reward into a series of positive payoffs back through the chain, each agent getting paid for contributing to moving the society closer to the goal state, where the CCV implementation's bids are closer to the optimal societal Q-value than those of the BB implementation. Because both the solitary (c) and cloned (f) versions of the V implementations do not conserve credit, they learn to bid close to the optimal societal Q-value, but both suffer from market bubbles where the primitive for going left bids higher than the primitive for going right, even though the optimal global policy is to keep moving right.

of the society to propagate utility from future agents to past agents in the absence of an immediate environment reward signal. We compare the bidding behaviors of the BB, V, and CCV implementations of the cloned Vickrey society, as well as those of their solitary counterparts, in Figure 6.6 and the society's global learning curve in Figure 6.8a.

To answer question (1), we observe that redundancy indeed prevents market bubbles, with the cloned CCV implementation bidding closet to the optimal societal Q values. Details are in the caption of Figure 6.5. When we consider the society's global learning curve in Figure 6.8a, the answers to questions (2) and (3) go hand-in-hand: the solitary societies fail to find the globally optimal policy and the cloned CCV implementation has the highest

Figure 6.6: **Multi-Step MDPs.** (a) In the *Chain* environment, the society starts at state s_0 and the goal state is s_5. Only activating primitive ω_1 at state s_4 yields reward. The optimal global policy is to directly move right by continually activating ω_1. Without credit conservation, the society may get stuck going back and forth between s_0 and s_4 without reaching the goal. (b) In the *Duality* environment, the society starts at state s_0. s_{-1} is an absorbing state with perpetual negative rewards. The optimal societal policy is to cycle between s_0 and s_1 to receive unbounded reward, but without redundant primitives, the society may end up in a suboptimal perpetual self-loop at s_1.

sample efficiency.

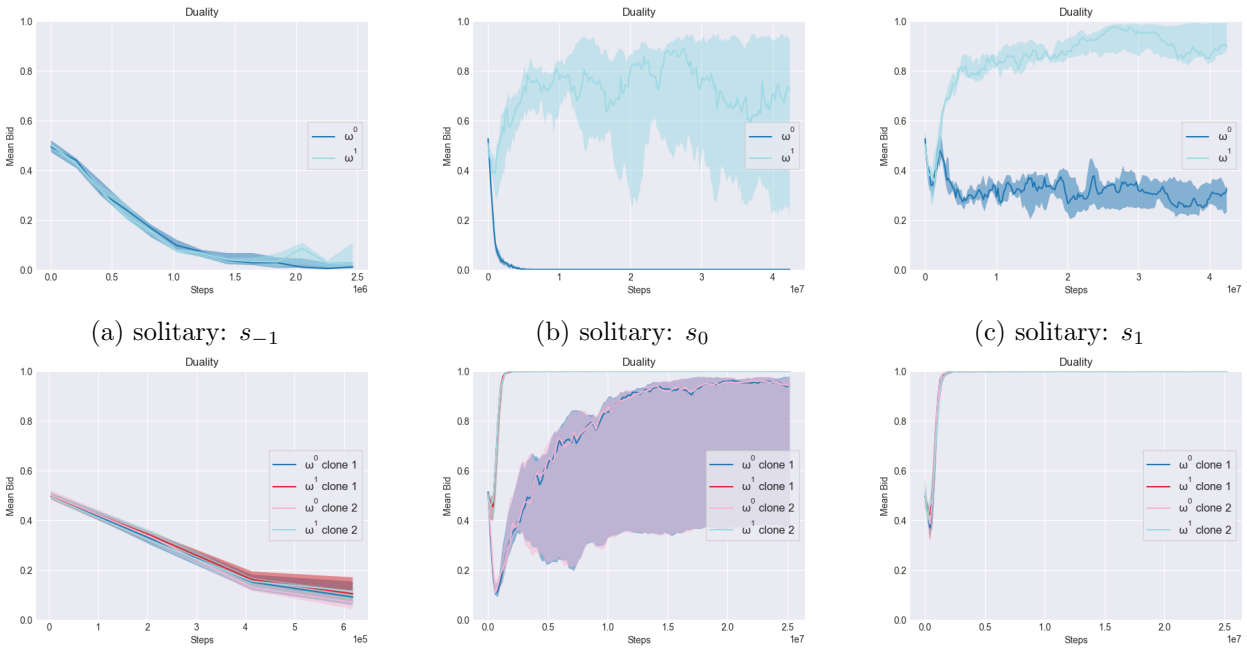
The *Duality* environment and suboptimal equilibria. The *Chain* environment experiments suggested a connection between lack of redundancy and globally suboptimal equilibria, the subtleties of which we explore further in the *Duality* environment (Figure 6.6b). The *CCV* implementation has yielded the best performance so far but does not guarantee Bellman optimality (Figure 6.4) without redundant primitives. We show that in the *Duality* environment, without redundant primitives the dominant strategy equilibrium would lead the society to get stuck in a self-loop at s_1 indefinitely, even though the global optimal solution would be to cycle back and forth between s_0 and s_1. The reasoning for this is explained in Appendix E.6. Figure 6.8b indeed shows that a society with redundant primitives learns a better equilibrium than without.

Semi-MDPs and Computation Graphs

One benefit of framing global decision-making from the perspective of local economic transactions is that the same societal decision-making framework and learning algorithms can be used regardless of the type of transformation $\phi_{\mathcal{T}}$. We now show in the *Two Rooms* environment that the cloned Vickrey society can learn more efficiently than a monolithic counterpart to select among pre-trained options to solve a `gym-minigrid` [64] navigation task that involves two rooms separated by a bottleneck state. We also show in the *Mental Rotation* environment that the cloned Vickrey society can learn to dynamically compose computation graphs of pre-specified affine transformations for classifying spatially transformed MNIST digits. We use the cloned *CCV* society for these experiments.

Transferring with Options in Semi-MDPs

We construct a two room environment, *Two Rooms*, which requires opening the red door and reaching either a green goal or blue goal. The transformations $\phi_{\mathcal{T}}$ are subpolicies that



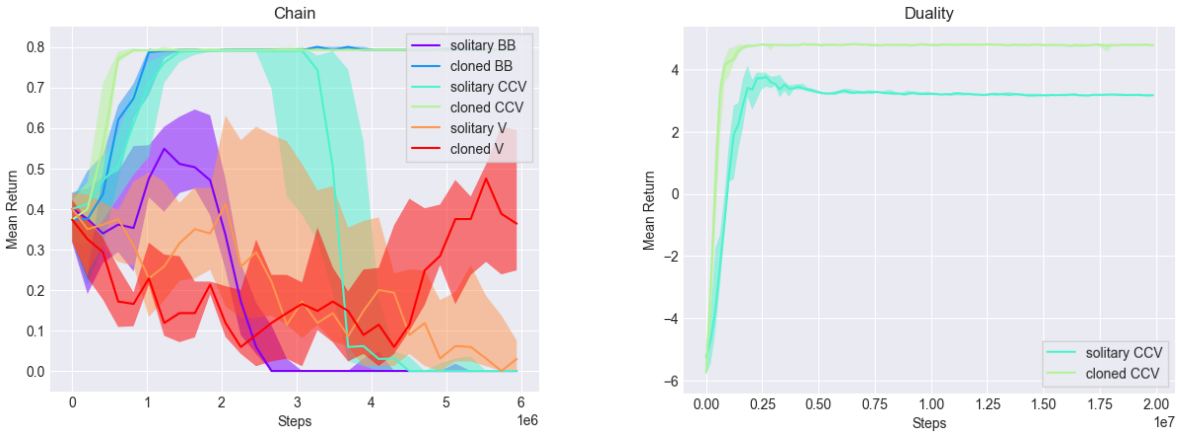
(a) solitary: s_{-1} (b) solitary: s_0 (c) solitary: s_1

(d) cloned: s_{-1} (e) cloned: s_0 (f) cloned: s_1

Figure 6.7: **CCV Bidding Curves for *Duality*.** Each column shows the bidding curves of the solitary (top row) and cloned (bottom row) *CCV* societies for states s_{-1}, s_0, and s_1. Without redundant primitives to force the second-highest and highest valuations to be equal, the dominant strategy of truthful bidding may not coincide with the globally optimal policy because the solitary *CCV* implementation does not guarantee Bellman optimality. The bidding curves in (c) show that ω^1 learns a best response of bidding *higher* than primitive ω^0 at state s_1, even though it would be globally optimal for the society if ω^0 wins at s_1. Adding redundant primitives causes the second-highest and highest valuations to be equal, causing ω^0 to learn to bid highly as well at s_1, which results in a more optimal return as shown in Figure 6.8b.

have been pre-trained to open the red door, reach the green goal, and reach the blue goal. In the pre-training task, only reaching the green goal gives a non-zero terminal reward and reaching the blue goal does not give reward. In the transfer task, the rewards for reaching the green and blue goals are switched. We compared the cloned Vickrey society against a non-hierarchical monolithic baseline that selects among the low-level `gym-minigrid` actions as well as a hierarchical monolithic baseline that selects among the pre-trained subpolicies. Both baseline policies sample from a Categorical distribution and are trained with PPO. The cloned Vickrey society is more sample efficient in learning on the pre-training task and significantly faster in adapting to the transfer task (Figure 6.10).

Our hypothesized explanation for this efficiency is that the local credit assignment mechanisms of a society parallelize learning across space and time, a property that is not true of the global credit assignment mechanisms of a monolithic learner. To test this hypothesis,



(a) Chain

(b) Duality

Figure 6.8: **Multi-Step MDP Global Learning Curves.** We observe that cloned societies are more robust against suboptimal equilibria than solitary societies. Furthermore the cloned *CCV* implementation achieves the best sample efficiency, suggesting that truthful bidding and credit-conservation are important properties to enforce for enabling the optimal global policy to emerge.

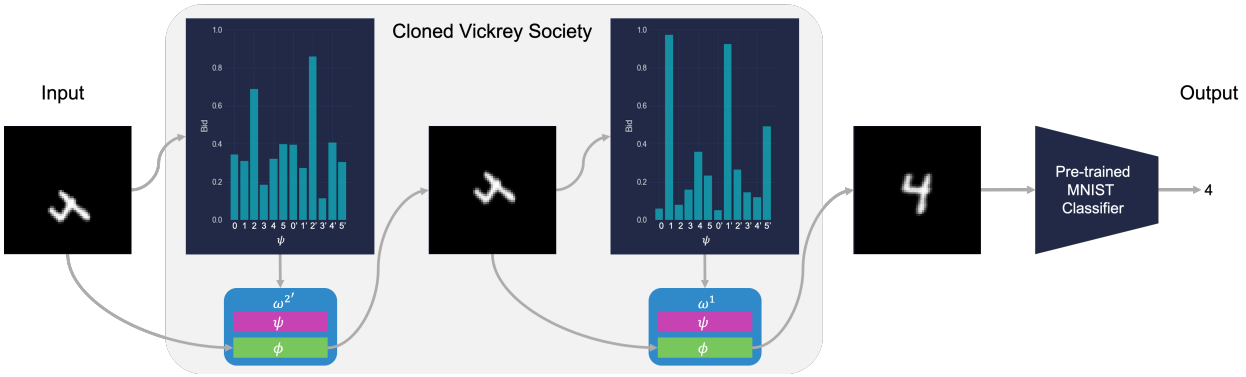


Figure 6.9: **Mental Rotation.** The cloned Vickrey society learns to transform the image into a form that can be classified correctly with a pre-trained classifier by composing two of six possible affine transformations of rotation and translation. Clones are indicated by an apostrophe. In this example, the society activated primitive $\omega^{2'}$ to translate the digit up then primitive ω^1 to rotate the digit clockwise. Though the bidding policies ψ^i and $\psi^{i'}$ of the clones ω^i and $\omega^{i'}$ have the same parameters, their sampled bids may be different because the bidding policies are stochastic.

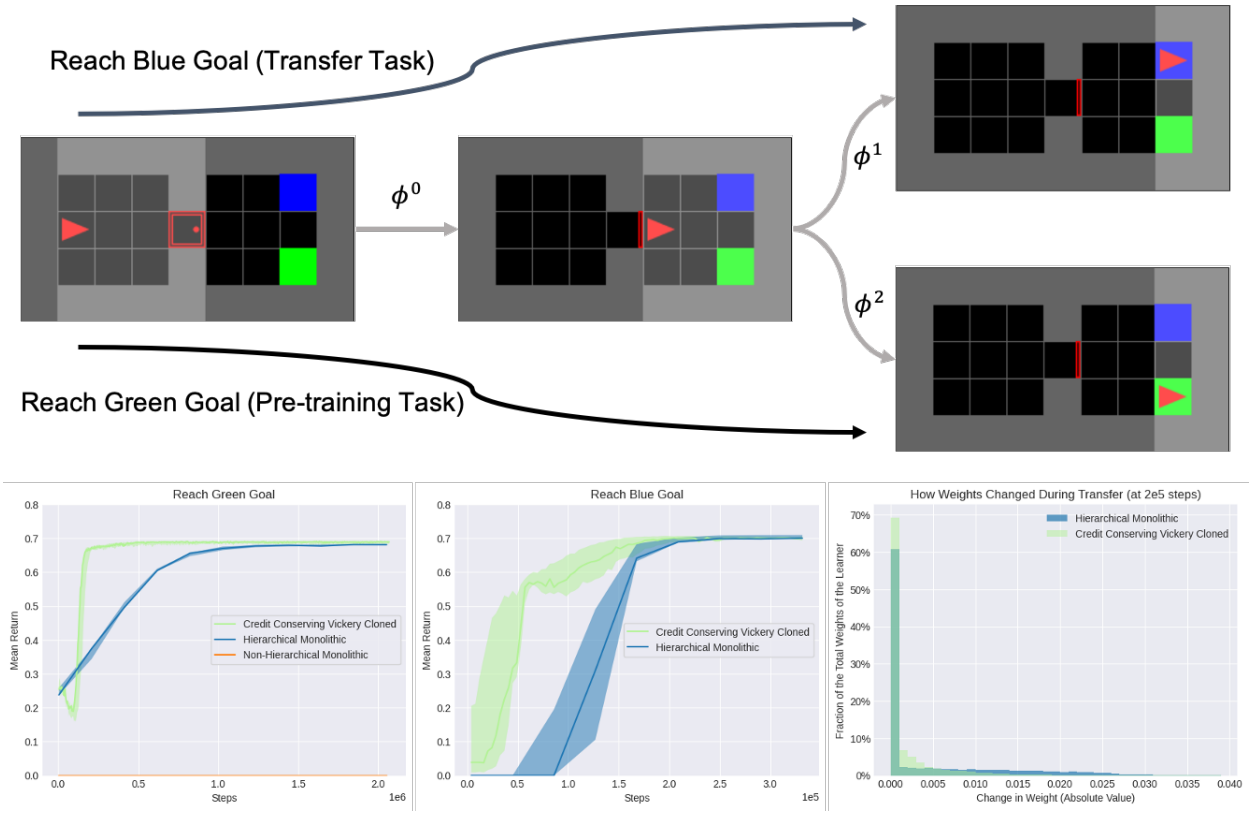


Figure 6.10: **Two Rooms.** The cloned Vickrey society adapts more quickly than the hierarchical monolithic baseline in both the pre-training and the transfer tasks. The bottom-right figure, which is a histogram of the absolute values of how much the weights have shifted from fine-tuning on the transfer task, shows that more weights shift, and to a larger degree, in the hierarchical monolithic baseline than in our method. This seems to suggest that the cloned Vickrey society is a more modular learner than the hierarchical monolithic baseline. The non-hierarchical monolithic baseline does not learn to solve the task from scratch.

we observe in the bottom-right of Figure 6.10 that not only has a higher percentage of the hierarchical monolithic baseline's weights shifted during transfer compared to our method but they also shift to a larger degree, which suggests that the hierarchical monolithic baseline's weights are more globally coupled and perhaps thereby slower to transfer. While this analysis is not comprehensive, it is a suggestive result that motivates future work for further study.

Composing Dynamic Computation Graphs

We adapt the Image Transformations task from Chang et al. [58] as what we call the *Mental Rotation* environment (Figure 6.9), in which MNIST images have been transformed with a composition of one of two rotations and one of four translations. There are $60,000 \times 8 = 240,000$ possible unique inputs, meaning learning an optimal global policy involves training

primitives across $240,000$ local auctions. The transformations $\phi_{\mathcal{T}}$ are pre-specified affine transformations. The society must transform the image into a form that can be classified correctly with a pre-trained classifier, with a terminal reward of 1 if the predicted label is correct and 0 otherwise. The cloned Vickrey society converges to a mean return of 0.933 with a standard deviation of 0.014.

6.8 Discussion

This work formally defines the societal decision-making framework and proves the optimality of a society – the cloned Vickrey society – for solving a problem that was first posed in the AI literature in the 1980s: to specify the incentive structure that causes the global solution of a society to emerge as the equilibrium strategy profile of self-interested agents. For training the society, we further proposed a class of decentralized reinforcement learning algorithms whose global objective decouples in space and time into simpler local objectives. We have demonstrated the generality of this framework for selecting actions, options, and computations as well as its potential advantages for transfer learning.

The generality of the societal decision-making framework opens much opportunity for future work in decentralized reinforcement learning. A society's inherent modular structure suggests the potential of reformulating problems with global credit-assignment paths into problems with much more local credit-assignment paths that offer more parallelism and independence in the training of different components of a learner. Understanding the learning dynamics of multi-agent societies continues to be an open problem of research. It would be exciting to explore algorithms for constructing and learning societies in which the primitives are also societies themselves. We hope that the societal decision-making framework and its associated decentralized reinforcement learning algorithms provide a foundation for future work exploring the potential of creating AI systems that reflect the collective intelligence of multi-agent societies.

Chapter 7

Local Credit Assignment

> It is causality that gives us this modularity, and when we lose causality, we lose modularity.
>
> Judea Pearl [99]

7.1 Introduction

Gusteau's [32] taqueria has a great team for making burritos: Colette heats the tortillas, Remy adds the meat, and Alfredo wraps the burrito in aluminum foil. But today customers fell sick from meat contamination and gave angry reviews. Clearly, Remy should replace meat with tofu or something else. But should credit assignment from the reviews affect the others? Intuitively, no: the feedback signals to the decision of adding meat and to the decisions of heating tortillas and wrapping aluminum foil should be independent. Customer dissatisfaction in burritos are not reflective of the taqueria's quesadillas, for which Colette's tortilla skills and Alfredo's wrapping skills are useful.

The example above expresses the intuition that **modularity**, or the capacity for the mechanisms in a system to be independently modified, enables flexible adaptation. However, using principles of modularity to build flexible learning systems has been difficult because the traditional formalism for precisely describing what modularity means was developed in the context of analyzing **static systems** – systems whose **mechanisms**, or functional components, are assumed fixed. But learning agents are **dynamic systems** composed of mechanisms (i.e. learnable functions) that evolve over the course of learning. Thus, to even express the hypothesis that modularity enables flexibility in learning agents, let alone test it, we first need (1) a formalism that defines what modularity means for dynamic systems, (2) a theory that identifies the conditions under which independent modification of learnable mechanisms is even possible, and (3) a practical criterion for determining when these conditions are met in learning algorithms. This chapter proposes candidate solutions

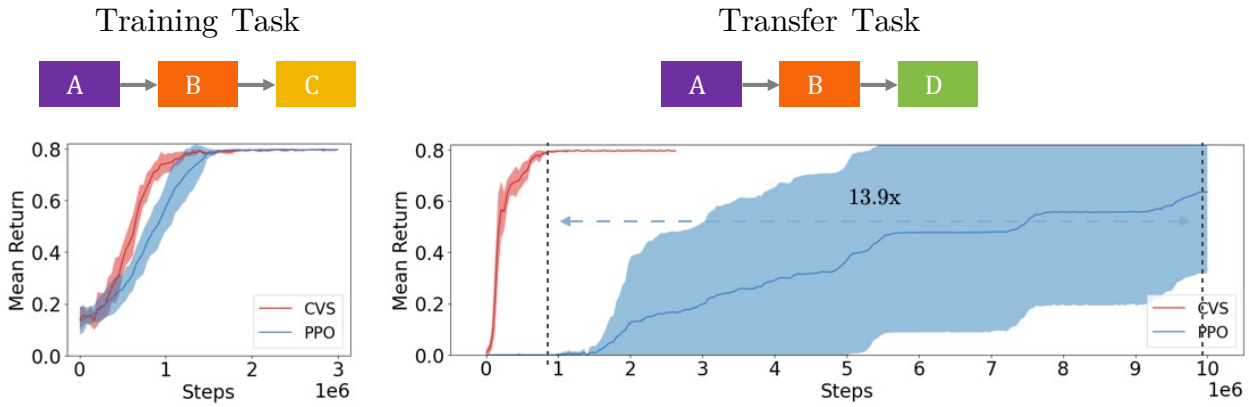


Figure 7.1: **Minimal motivating example.** The optimal action sequence for the training task is $A \rightarrow B \rightarrow C$, and the optimal sequence for the transfer task differs only in the last time-step. Continuing to train an optimal policy from the training task on the transfer task with the cloned Vickrey society (CVS) from Chang et al. [54] transfers 13.9x more efficiently than with PPO [282], even though learning efficiency for both on-policy algorithms during training is comparable. This chapter suggests that this is due to **dynamic modularity**: the algorithmic independence among CVS's learnable mechanisms and among their gradients.

to these problems and applies them to shed new insight on the modularity of discrete-action reinforcement learning (RL) algorithms. The takeaway message of this chapter is that independent modification of mechanisms requires both the mechanisms *and* the feedback signals that update them to be independent: a modular learning algorithm must have a credit assignment mechanism whose algorithmic causal structure makes such independent modification possible.

Janzing and Schölkopf [166] proposed to precisely characterize modularity in static systems as the algorithmic independence of mechanisms in the computational graph used to describe the system. In learning systems, the computational graph in question depicts the forward pass of a learner (e.g. a neural network), but this graph itself evolves over the course of learning because the learnable functions – the mechanisms – get modified. For such dynamic systems, we extend the static notion of modularity to define **dynamic modularity** as the algorithmic independence of mechanisms in the current iteration, conditioned on the graph from the previous iteration of evolution. This addresses problem (1).

Modularity matters when the system needs to be modified for a new context or purpose. In learning systems it is the credit assignment mechanism that performs this modification. Thus dynamic modularity is tied to independence in feedback: for a gradient-based learner, we show that enforcing dynamic modularity requires enforcing gradients to be algorithmically independent as well, which we call the **modularity constraint.** This addresses problem (2).

Algorithmic independence is generally incomputable, which makes the modularity constraint intractable to evaluate. To make this constraint practical for analyzing learning

algorithms, we formally represent the entire learning process as one big causal graph, which we call the **algorithmic causal model of learning** (ACML). Then the modularity constraint translates into an easy-to-inspect criterion, the **modularity criterion**, on d-separation in ACML that enables us to evaluate, without any training, whether a learning algorithm is exhibits dynamic modularity. This addresses problem (3).

Having established a theoretically-grounded formalism for reasoning about modularity in learning systems, we theoretically and empirically analyze discrete-action RL algorithms. The mechanisms of interest are the functions that compute the "bid" (e.g. action probability or Q-value) for each value of the action variable. The Markov decision process (MDP) is too coarse-grained to represent these functions separately, so we use the societal decision-making framework (SDM) from Chang et al. [54], whose computational graph does treat them separately. We prove that certain single-step temporal difference methods satisfy the modularity criterion while all policy gradient methods do not. Empirically, we find that for transfer problems that require only sparse modifications to a sequence of previously optimal decisions, implementations of algorithms that exhibit dynamic modularity transfer more efficiently than their counterparts. All proofs are in the Appendix.

Assumptions and approximations The theory developed in §7.4, §7.5, and §7.6 assume computational graphs over arbitrary strings. Thus we will understand statements about Kolmogorov complexity and algorithmic information in these sections as inherently asymptotic, pertaining to strings of increasing length, where equations that hold up to constant terms are well defined. Similarly, our discussion in §7.6 will pertain not to concrete instantiations of RL algorithms but only to the causal structure of these algorithms in the abstract. As with all explanations in science, there is inevitably a gap between theory and practice. In particular, when considering empirical performance of concrete instantiations of RL algorithms in §7.7 on a specific Turing machine, asympototic statements are not meaningful, but we can still use empirical observation to refute or improve our theory.

7.2 Related Work

The hypothesis that modularity could improve flexibility of learning systems has motivated much empirical work in designing factorized architectures [74, 14, 58, 121, 178, 9, 244] and reinforcement learners [288, 300, 268], but the extent to which the heuristics used in these methods enforce the learnable components to be independently modifiable has yet to be tested. Conversely, other works begin by defining a multi-agent system of independently modifiable components and seek methods to induce their cooperation with respect to a global objective [22, 29, 303, 54, 110, 23], but the precise property of a learning system that characterizes its modularity has not been discussed in these works, as far as we are aware. Recent complementary work has proposed alternative measures of modularity, restricted to deep networks, based on connectivity strength [92] and functional decomposition [68]. In contrast, our work identifies a general property that defines the modularity of a learning

system as the algorithmic independence of learnable mechanisms and of their gradients, and presents a practical method for testing for this property without any training. We build upon the theoretical foundations from Janzing and Schölkopf [166] that have clarified similar notions of "autonomy" and "invariance" that underlie axioms of econometrics [144, 8], causality [246, 248], and computer programming [2]. Yu et al. [345] explored enforcing the linear independence of gradients to improve multi-task learning, and formulating the precise connection between algorithmic and linear independence would be valuable future work.

7.3 Background

Our analysis of the modularity of RL algorithms employs two key ideas: (§7.3) computational graphs can be interpreted as causal graphs and (§7.3) a learnable discrete-action policy can be interpreted as a society of learnable action-specific functions and a fixed selection mechanism.

Algorithmic Causality

We begin by reviewing terms from algorithmic information theory [181, 201, 294][1]. We assume that programs are expressed in a language L and run on a universal Turing machine. Given binary strings x, y, and z, we denote **conditional algorithmic independence** as $x \perp\!\!\!\perp y \mid z$, equivalently $I(x : y \mid z) \stackrel{+}{=} 0$, which reads "given z, knowledge of y does not allow for a stronger compression of x." Let x^* denote the shortest program that produces x. I denotes **conditional algorithmic mutual information**. $\stackrel{+}{=}$ denotes equality up to a constant that depends on L but not the strings on either side of the equality (see §7.1). **Conditional Kolmogorov complexity** of y given x is given by $K(y \mid x)$, the length of the shortest program that generates y from x as input.

Janzing and Schölkopf [166, Post. 6] generalized structural causal models [245] to general programs, allowing us to treat computational graphs as causal graphs.

Definition 7.3.1 (**computational graph**)**.** *Define a* ***computational graph*** $\mathcal{G} = (\boldsymbol{x}, \boldsymbol{f})$ *as a directed acyclic factor graph (DAG) of variable nodes* $\boldsymbol{x} = x_1, ..., x_N$ *and function nodes* $\boldsymbol{f} = \boldsymbol{f}^1, ..., \boldsymbol{f}^N$*. Let each* x_j *be computed by a program* $\boldsymbol{f}^j$ *with length* $O(1)$ *from its parents* $\{pa_j\}$ *and an auxiliary input* n_j*. Assume the* n_j *are jointly independent:* $n_j \perp\!\!\!\perp \{n_{\neq j}\}$*. Formally,* $x_j := \boldsymbol{f}^j(\{pa_j\}, n_j)$*, meaning that the Turing machine computes* x_j *from the input* $\{pa_j\}, n_j$ *using the additional program* $\boldsymbol{f}^j$ *and halts.*

By absorbing the n_j into the functions $\mathtt{f}^j$ we can equivalently assume that $\mathtt{f}^j$ are jointly independent, but not necessarily $O(1)$ Janzing and Schölkopf [166, Post. 6]. If the n_j are interpreted as noise, this DAG represents a probabilistic program [221, 116, 211] that implements a standard causal model. Henceforth we treat all graphs as computational graphs. We define a **mechanism** as the string representation (i.e. source code) of the program that

[1] See the appendix for background.

implements a function `f` and **data** as the string representations of the input/output variables x of `f`. The **algorithmic causal Markov condition** [166, Thm. 4], which states that d-separation implies conditional independence, generalizes the standard Markov condition to general programs:

Theorem 7.3.1 (**algorithmic causal Markov condition**). *Let $\{pa_j\}$ and $\{nd_j\}$ respectively represent concatenation of the parents and non-descendants (except itself) of x_j in a computational graph. Then $\forall x_j$, $x_j \perp\!\!\!\perp \{nd_j\} \mid \{pa_j\}^*$.*

In standard causality it is typical to assume the converse of the Markov condition, known as **faithfulness** [299]. We do the same for algorithmic causality:

Postulate 7.3.2 (**algorithmic faithfulness**). Given sets S, T, R of nodes in a computational graph, $I(S:T|R^*) \stackrel{+}{=} 0$ implies R d-separates S and T.

Societal Decision-Making

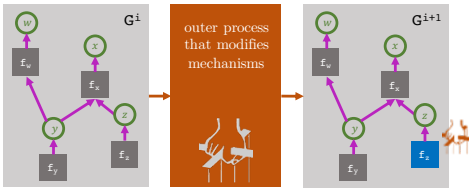
A discrete-action MDP is the standard graph for a sequential decision problem over states S with N discrete actions A, defined with state space $\mathcal{S}$, action space $\{1, ..., N\}$, transition function $\mathtt{T} : \mathcal{S} \times \{1, ..., N\} \to \mathcal{S}$, reward function $\mathtt{R} : \mathcal{S} \times \{1, ..., N\} \to \mathbb{R}$, and discount factor γ. The MDP objective is to maximize the return $\sum_{t=0}^{T} \gamma^t \mathtt{R}(s_t, a_t)$ with respect to a policy $\pi : \mathcal{S} \to \{1, ..., N\}$. We define a **decision** as a value a of A. The MDP abstracts over the mechanisms that control each decision with a single edge in the graph, represented by π, but to analyze the independence of different decisions we are interested in representing these mechanisms as separate edges.

The societal decision-making (SDM) framework [54] offers an alternative graph that does exactly this: it decomposes a discrete-action policy as a society of N agents ω^k that each controls a different decision. Each agent is a tuple (ψ^k, ϕ^k) of a bidder $\psi^k : \mathcal{S} \to \mathcal{B}$ and a fixed transformation $\phi^k : \mathcal{S} \to \mathcal{S}$. In §7.6, we will consider the algorithmic independence of the ψ^n. Recovering a policy π composes two operations: one computes bids $b_s^k := \psi^k(s)$, $\forall k$, and one applies a selection mechanism $\mathtt{S} : \mathcal{B}^N \to \{1, ..., N\}$ on the bids to select decision a. SDM thus curries the transition and reward functions as $\mathtt{T} : \{1, ..., N\} \to [\mathcal{S} \to \mathcal{S}]$ and $\mathtt{R} : \{1, ..., N\} \to [\mathcal{S} \to \mathbb{R}]$.

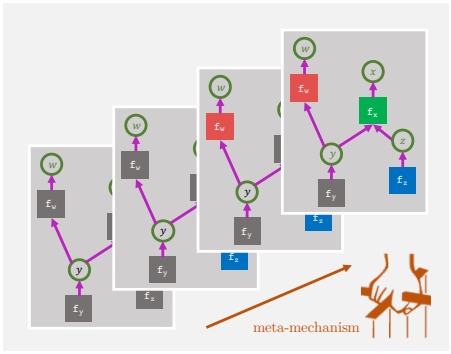
Chang et al. [54] introduced the cloned Vickrey society (CVS) algorithm as an on-policy single-step temporal-difference action-value method. CVS interprets the Bellman optimality equation as an economic transaction between agents seeking to optimize their utilites in a Vickrey auction [321] at each time-step. The Vickrey auction is the selection mechanism that selects the highest bidding agent i, which receives a utility

$$\underbrace{U_{s_t}^i(\omega^{1:N})}_{\text{utility}} = \underbrace{\mathtt{R}^\phi(\omega^i, s_t) + \gamma \cdot \max_k b_{s_{t+1}}^k}_{\text{revenue, or valuation } v_{s_t}} - \underbrace{\max_{j \neq i} b_{s_t}^j}_{\text{price}}, \tag{7.1}$$

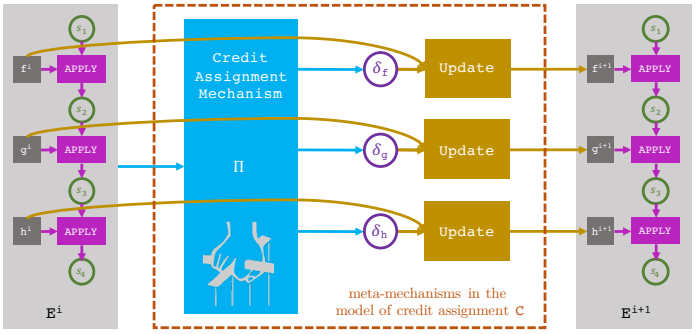
and the rest receive a utility of 0. In CVS each agent bids twice: the highest and second highest bids are produced by the same function parameters. The auction incentivizes each



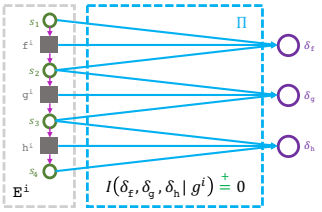
(a) A modification of a mechanism.



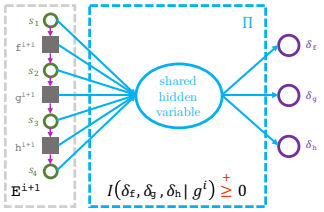
(b) A dynamic system.



(c) Learning algorithms are dynamic systems.



(d) A modular credit assignment structure



(e) A non-modular credit assignment structure.

Figure 7.2: **Key Ideas.** A system can be represented as a algorithmic causal graph $\mathtt{G}$. (a) A modification to a mechanism in graph $\mathtt{G}^i$ generates a new graph $\mathtt{G}^{i+1}$. (b) A dynamic system encompasses an outer process that generates a sequence of graphs via a series of modifications to the mechanisms. (c) Learning algorithms are examples of dynamic systems, where the outer process is the model of credit assignment $\mathtt{C}$, which modifies the mechanisms of the model of execution $\mathtt{E}$, which represents the forward pass of the learner. By flattening the learning algorithm as one algorithmic causal graph, we can determine whether the causal structure of the credit assignment mechanism makes independent modification of learnable mechanisms possible by inspecting whether the gradients are d-separated by the previous graph $\mathtt{C}^i$. A credit assignment mechanism is (d) modular if they are d-separated and (e) not modular if not.

agent to truthfully bid the Q-value of its associated transformation mechanism, independent of the identities and bidding strategies of other agents.

7.4 Dynamic Modularity in Learning Systems

This section extends the definition of modularity in static systems to dynamic systems. We discuss learning algorithms as examples of dynamic systems and the constraints that must be imposed on the credit assignment mechanism for the learning algorithm to be exhibit dynamic modularity.

From Static Modularity to Dynamic Modularity

A system can be described by a computational graph (i.e. an algorithmic causal graph), whose mechanisms represent the system components and whose data represent the information communicated between components. In standard causal analysis [246], mechanisms are generally treated as fixed. In this context, we call the graph a **static graph** that describes a **static system.** Modularity in the static context has been defined as the autonomy [246, §1.3.1], or more precisely, the algorithmic independence [166, Post. 7] of mechanisms:

Definition 7.4.1 (**static modularity**)**.**

$$\forall k \neq j, \quad I\left(\boldsymbol{f}^k : \boldsymbol{f}^j\right) \stackrel{+}{=} 0. \tag{7.2}$$

If mechanisms are algorithmically independent, then one can be modified without an accompanying modification in the others to compensate. This enables the analysis of counterfactual queries, for example, where a human performs a hypothetical modification to a mechanism of a static graph (Fig. 7.2a). To analyze a different query, the mechanism is reset to its original state before the hypothetical modification, and the human then performs a different hypothetical modification on the original static graph. The human is a **meta-mechanism**, a mechanism that modifies mechanisms.

Whereas static graphs are useful for analyzing systems under human control, many systems in the real world (e.g. star systems, whose mechanisms are stars that communicate via forces) are **dynamic systems** (Fig. 7.2b), whose mechanisms evolve through time (e.g. the stars move). We describe these systems with a **dynamic graph**. In the dynamic context, the meta-mechanism is not the human, but symmetric laws that govern the time evolution of the mechanisms (e.g. physical laws governing stars' motion are invariant to change reference frame). Since any snapshot in time of a dynamic system depicts a static system, we naturally extend the traditional definition of modularity to the dynamic context:

Definition 7.4.2 (**dynamic modularity**)**.**

$$\forall k \neq j, \quad I\left(\boldsymbol{f}^{k,i+1} : \boldsymbol{f}^{j,i+1} \;\middle|\; \boldsymbol{x}^i, \boldsymbol{f}^i\right) \stackrel{+}{=} 0. \tag{7.3}$$

Dynamically modularity simply re-interprets static modularity as a snapshot i along the temporal dimension.

Learning Algorithms are Dynamic Systems

We now show that general learning algorithms are examples of dynamic systems and can be analyzed as such. To do so, we need to specify the data and mechanisms of the static computational graph that represents a particular snapshot, as well as the equivariant meta-mechanism that evolves the mechanisms from one iteration to the next.

Let the **model of execution** be the computational graph `E` that represents the forward pass of the learner, generating $\mathbf{x}$ as an execution trace $(x_1, ..., x_t, ..., x_T)$ of the input and

output data of the learnable mechanisms $\mathtt{f}$. For example, with MDPs, the forward pass is a rollout, the trace records its states, actions, and rewards, and the mechanisms, which map parent variables $\{pa\}_t = s_t$ to child variables $x_t = (a_t, s_{t+1}, r_t)$, are instances of the policy at different steps t.

Let the **model of credit assignment** be the computational graph $\mathtt{C}$ that evolves the mechanisms. Each step represents the backward pass of the learner. Here the mechanisms are treated as data for two equivariant meta-mechanisms, the credit assignment mechanism $\Pi(\mathbf{x}, \mathtt{f}) \to \boldsymbol{\delta}$ and the update rule $\mathtt{UPDATE}(\mathtt{f}, \boldsymbol{\delta}) \to \mathtt{f}'$. $\mathtt{C}$ can be viewed as a reward-less MDP with states $\mathtt{f}$ and actions $\boldsymbol{\delta}$, with $\mathtt{UPDATE}$ as the transition function. Then Π is a context-conditioned policy that generates modifications $\boldsymbol{\delta} = (\delta_1, ..., \delta_T)$ to the functions $\mathtt{f}$ of the learner, given $\mathbf{x}$ as context. For a gradient-based learner, δ_t^k is the gradient of the learning objective with respect to the function $\mathtt{f}^k$ that participated at step t of the execution trace (e.g. as we discuss in §7.6, δ_t^k would be the Bellman error of the decision mechanism for action k taken at step i). $\mathtt{UPDATE}$ performs the parallel operation $\mathtt{UPDATE}(\mathtt{f}^k, \sum_t \delta_t^k) \to \mathtt{f}^{k\prime}$ over all mechanisms $\mathtt{f}^k$. The choice of optimizer for gradient descent (e.g. Adam [174]) determines the functional form of $\mathtt{UPDATE}$. Henceforth we assume gradient-based learning, but our results hold more generally given the assumptions that $\mathtt{UPDATE}$ (1) is algorithmically independent of Π and (2) completely factorizes across k.

Modularity Constraint on Credit Assignment

The design of a learning algorithm primarily concerns the credit assignment mechanism Π, whereas the choice of $\mathtt{UPDATE}$ is often assumed. We now present the constraint Π must satisfy for dynamic modularity to hold at every iteration of learning. Given trace $\mathbf{x}$ and previous mechanisms $\mathtt{f}$, we define the **modularity constraint** as that which imposes that the gradients $\delta_1, ..., \delta_T$ be jointly independent:

Definition 7.4.3 (**modularity constraint**)**.**

$$I\left(\delta_1, ..., \delta_T \mid \boldsymbol{x}, \boldsymbol{f}\right) \stackrel{+}{=} 0. \tag{7.4}$$

A **modular credit assignment mechanism** is one that satisfies the modularity constraint. If $\mathtt{E}$ exhibited statically modularity (i.e. its functions were independently initialized) then a modular Π enforces dynamic modularity:

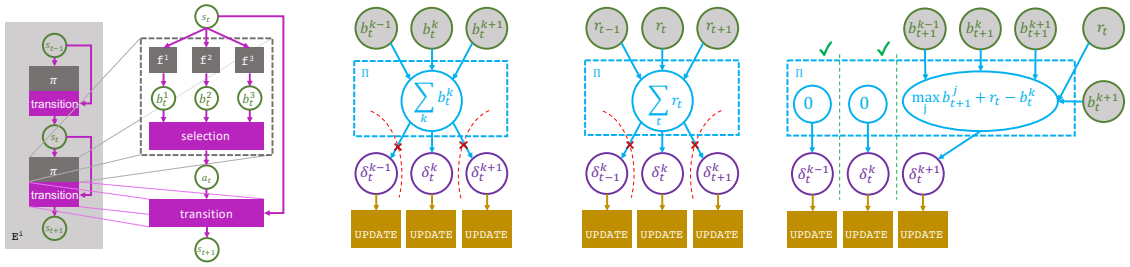
Theorem 7.4.1 (**modular credit assignment**)**.** *Dynamic modularity is enforced at learning iteration i if and only if static modularity holds at iteration $i = 0$ and the credit assignment mechanism satisfies the modularity constraint.*

Initializing different functions with different weights is not sufficient to guarantee dynamic modularity. The gradients produced by Π must be independent as well. If Π were not modular it would be impossible for it to modify a function without simultaneously inducing a dependence with another, other than via non-generic instances where δ_t has a simple

description, i.e. $\delta_t = 0$, which, unless imposed, are unlikely to hold over all iterations of learning.

7.5 An Algorithmic Causal Model of Learning

We can determine the dynamic modularity of a learning algorithm if we can evaluate the modularity constraint, but evaluating it is not practical in its current form because algorithmic information is generally incomputable. This section proposes to bypass this incomputability by translating the constraint into a d-separation criterion on the causal structure of Π, defined as part of one single causal graph of the learning process, which combines both the model of execution and the model of credit assignment. The challenge to constructing this graph is that `f` are treated as functions in `E` but as data in `C`, so it is not obvious how to reconcile the two in the same graph. We solve this by treating the function application operation `APPLY` [2][2], where $\forall \texttt{f}, x,\ \texttt{APPLY}(\texttt{f}, x) := \texttt{f}(x)$, as itself a function in a computational graph, enabling us to treat both `f` and x as variables in the same flattened dynamic graph (Fig. 7.2c).



(a) A policy π decomposes into a set of decision mechanisms `f`.

(b) Policy gradient methods.

(c) N-step temporal difference methods.

(d) Single-step temporal difference methods.

Figure 7.3: **Modularity in RL.** In RL, the forward pass is a rollout in the MDP. (a) The societal decision-making framework exposes the learnable decision mechanisms of the policy as separate components in the model of execution. The bids b represent either action probabilities are estimated action-specific Q-values. The credit assignment mechanisms of (b) policy gradient methods and (c) TD($n > 1$) methods, like using Monte Carlo estimation, contain shared hidden variable and thus do not produce algorithmically independent gradients δ. (d) TD(0) methods have modular credit assignment mechanisms in generic cases. The red crosses indicate a lack of d-separation, whereas the green checkmarks do.

Lemma 7.5.1 (**algorithmic causal model of learning**). *Given a model of execution* `E` *and of credit assignment* `C`*, define the* ***algorithmic causal model of learning*** *(ACML) as a dynamic computational graph* $\mathbb{L}$ *of the learning process. We assume* Π *has its own internal causal structure with internal variable and function nodes. The function nodes of* $\mathbb{L}$ *are* `APPLY`*,* `UPDATE`*, and internal function nodes of* Π*. The variable nodes of* $\mathbb{L}$ *are* x*,* `f`*,* δ*,*

[2]This operation is known in λ-calculus as β-reduction.

and internal variable nodes of Π. `APPLY` *and* `UPDATE` *are assumed to have length* $O(1)$. *The internal function nodes of* Π *jointly independent, and along with the variable nodes of* $\mathbb{L}$, *are assumed to not have length* $O(1)$. *Then these variable nodes satisfy the algorithmic causal Markov condition with respect to* $\mathbb{L}$ *for all steps of credit assignment.*

ACML is the bridge that brings tools from algorithmic causality [166] to bear on analyzing not simply the algorithmic independence of variables, but algorithmic independence of *functions* in general learning algorithms. The learnable mechanisms are no longer considered to have length $O(1)$ as is assumed in the model of execution. With ACML, we define a criterion to test whether the modularity constraint holds by direct inspection:

Theorem 7.5.2 (**modularity criterion**). *If* $\mathbb{L}$ *is faithful, the modularity constraint holds if and only if for all* i, *outputs* δ_t *and* $\delta_{\neq t}$ *of* Π *are d-separated by its inputs* $\boldsymbol{x}$ *and* $\boldsymbol{f}$.

We generally have access to the true computational graph, because the learning algorithm was programmed by us. Thus Thm. 7.5.2 enables us to evaluate, before any training, whether a learning algorithm satisfies the modularity constraint by simply inspecting $\mathbb{L}$ for d-separation (Fig. 7.2d,e), giving us a practical tool to both design and evaluate learning algorithms on the basis of dynamic modularity.

7.6 Modularity in Reinforcement Learning

We now apply the modularity criterion to evaluate the dynamic modularity of two major classes of RL algorithms [308] – action-value and policy-gradient methods. The modularity criterion unlocks the use graphical language for our analysis, which simplifies the proofs. We define a common model of execution for all algorithms within the SDM framework from §7.3 that enables us to compare the causal structures of their different credit assignment mechanisms under ACML. We find that in the general function approximation setting, assuming acyclic decision sequences, the cloned Vickrey society (CVS, §7.3) is the only algorithm to our knowledge so far that produces reinforcement learners that exhibit dynamic modularity.

From Monolithic Policies to Decision Mechanisms

As mentioned in §7.3, and as motivated by our taqueria example, we are interested in analyzing the independence of different decisions, so we need to adapt the model of execution we gave as an example for MDPs in §7.4 to treat the functions that control each decision as separate mechanisms.

We observe from the SDM framework that any discrete-action policy π with N actions can be decomposed into a set of mechanisms computing a "bid" $b^k_{s_t}$ for each **decision** k (i.e., a value of the action variable, recall §7.3) at the given state s_t, and an independent selection mechanism that selects a decision given the bids (Fig. 7.3a). Define a **decision mechanism**

as the function that computes a bid. For policy-gradient methods, a bid corresponds to the action probability for a particular action $p(a = k|\cdot)$, and the selection mechanism is the stochastic sampler for a categorical variable. For action-value methods, a bid corresponds to the estimated Q-value for a particular action, $Q(\cdot, a = k)$, and the selection mechanism could be an ε-greedy sampler or a Vickrey auction [54]. Often decision mechanisms share weights (e.g. DQN [226]) and thus are algorithmically dependent, but for some algorithms they do not, as in CVS. Then, by absorbing the transition function `T` and reward function `R` into `APPLY`, the function nodes `f` of our model of execution are the decision mechanisms, which each take as input s_t, and produce as output the tuple $(b^k_{s_t}, s_{t+1}, r_t, w^k_t)$, where w_t is a binary flag that indicates whether the selection chose its corresponding action. The execution trace $\mathbf{x}$, which we call a **decision sequence**, records the values of these variables in a rollout.

The Modularity of RL Algorithms

We now ask which action-value and policy-gradient methods exhibit dynamic modularity by evaluating whether their credit assignment mechanisms satisfy the modularity criterion and whether their decision mechanisms share weights.

Which RL algorithms satisfy the modularity criterion? The modularity criterion can be violated if there exists a shared hidden variable in the causal structure of Π that couples together the gradients δ, which causes the δ^k_t's to not be d-separated given $\mathbf{x}$ and `f` (Fig. 7.3b-d).

For all policy gradient methods, the gradient into the action probabilities includes a normalization term $\sum_k b^k$ as a shared hidden variable (Fig. 7.3b):

Corollary 7.6.0.1 (**policy gradient**). *All policy gradient methods do not satisfy the modularity criterion.*

We divide action-value methods into single-step and n-step (where $n > 1$) temporal difference methods, abbrv. TD(0) and TD($n > 1$) respectively. For TD($n > 1$) methods, such as those that use Monte Carlo (MC) estimation of returns, TD(λ) [307], or generalized advantage estimation [281], this shared hidden variable is a sum of estimated returns or advantages at different steps of the decision sequence (Fig. 7.3c):

Corollary 7.6.0.2 (**n-step TD**). *All TD($n > 1$) methods do not satisfy the modularity criterion.*

This leaves only TD(0) methods. If the decision mechanism $\mathtt{f}^k$ were selected (i.e. $w^k_t = 1$) at step i, these methods produce, for some function g, gradients as $\delta^k_t := g(b^k_{s_t}, s_t, s_{t+1}, r_t, \mathtt{f})$. Otherwise, $\delta^k_t := 0$. For example, for Q-learning, g is the TD error $[\max_j b^j_{s_{t+1}} + r_t - b^k_{s_t}]$ (Fig. 7.3d), where $[\max_j b^j_{s_{t+1}}]$ is computed from s_{t+1} and `f`. The only hidden variable is $[\max_j b^j_{s_{t+1}}]$. It is only shared when the decision sequence $\mathbf{x}$ contains a cycle where two

states s_t and s'_t transition into the same state s_{t+1}. In this cyclic case, the credit assignment mechanism would not satisfy the modularity criterion. Otherwise it does:

Corollary 7.6.0.3 (**single-step TD**). *TD(0) methods satisfy the modularity criterion for acyclic* $\boldsymbol{x}$.

As cyclic $\mathbf{x}$ are non-generic cases that arise from specific settings of $\mathbf{x}$, we henceforth restrict our analysis to the acyclic case, justifying this restriction similarly to the justification of assuming faithfulness in other causal literature.

Which RL algorithms exhibit dynamic modularity? We have identified TD(0) methods as the class of RL algorithms that satisfy the modularity criterion. By Thm. 7.4.1, whether they satisfy dynamic modularity now depends on whether they satisfied static modularity at initialization ($i = 0$). We assume random initialization of `f`, so the only source of dependence among `f` is if they share parameters.

In the tabular setting, decision mechanisms are columns of the Q-table corresponding to each action. Because these columns do not share parameters, Q-learning [325], SARSA [265], and CVS exhibit dynamic modularity:

Corollary 7.6.-3.1 (**tabular**). *In the tabular setting, Thm. 7.4.1 holds for Q-learning, SARSA, and CVS.*

In the general function approximation setting, static modularity requires decision mechanisms to not share weights, which eliminates DQN [226] and its variants.

Corollary 7.6.-3.2 (**function approximation**). *In the function approximation setting, Thm. 7.4.1 holds for TD(0) methods whose decision mechanisms do not share parameters.*

To our knowledge, CVS is the only proposed TD(0) method with this property, but it is straightforward to make existing TD(0) methods exhibit dynamic modularity by using separate networks for estimating the Q-value of each decision.

Summary. If we want dynamic modularity, then we need the decision mechanisms to not share parameters and the credit assignment mechanism to not contain a shared hidden variable that induced algorithmic dependence among the gradients it outputs. An RL algorithm with dynamic modularity makes it possible for individual decision mechanisms to be modified independently without an accompanying modification to other decision mechanisms.

7.7 Simple Experiments

This chapter is motivated by the hypothesis that modularity enables flexible adaptation. To test this hypothesis requires (1) a method for determining whether a learning algorithm is modular and (2) a metric for evaluating flexible adaptation. The previous sections have

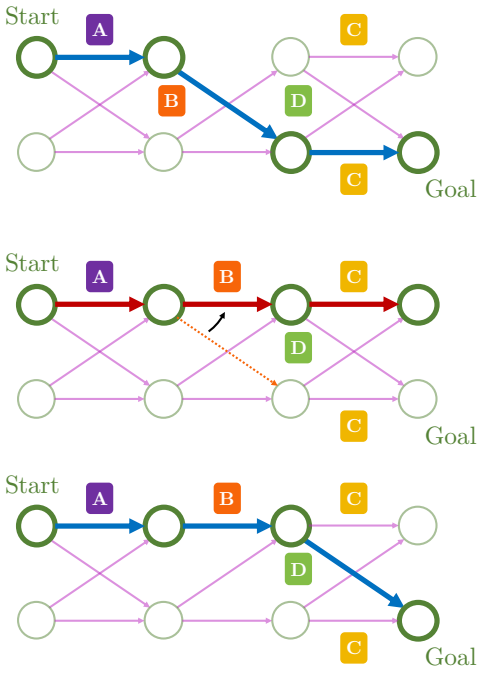


Figure 7.4: **How transfer tasks are generated.** We consider transfer problems where the optimal decision sequence of the transfer task differs from that of the training task by a single decision. As above, the transfer MDP and the training MDP differ in that the effect of action **B**; all other transitions remain the same. The agent must learn to choose action **D** instead of **C** while re-using other previously optimal decisions.

contributed (1). The metric we use for (2) is the comparative transfer efficiency of an algorithm that exhibits dynamic modularity with respect to one that does not. We consider transfer problems that require modifying only one decision in a previously optimal decision sequence needs to be changed, similar to our motivating example with Gusteau's taqueria (§7.1).

Our evaluation focuses on discrete-action on-policy RL algorithms since many factors that influence the learning of off-policy methods are still not well understood [3, 187, 316, 103]. Specifically we compare three algorithms that span the spectrum of action-value and policy-gradient methods. CVS represents a method that exhibits dynamic modularity. PPO [282] represents a method that is not modular at all. PPOF is a modification of PPO whose where each action logit is computed by a different network, and represents a method that exhibits static modularity at initialization but not dynamic modularity during learning.

We designed our experiments to be as minimal as possible to remove confounders. States are represented as binary vectors. The reward is given at the end of the episode and is 1 if the task is solved and 0 otherwise. The relationship between the training and transfer MDP is given by an intervention in the MDP transition function (Fig. 7.4).

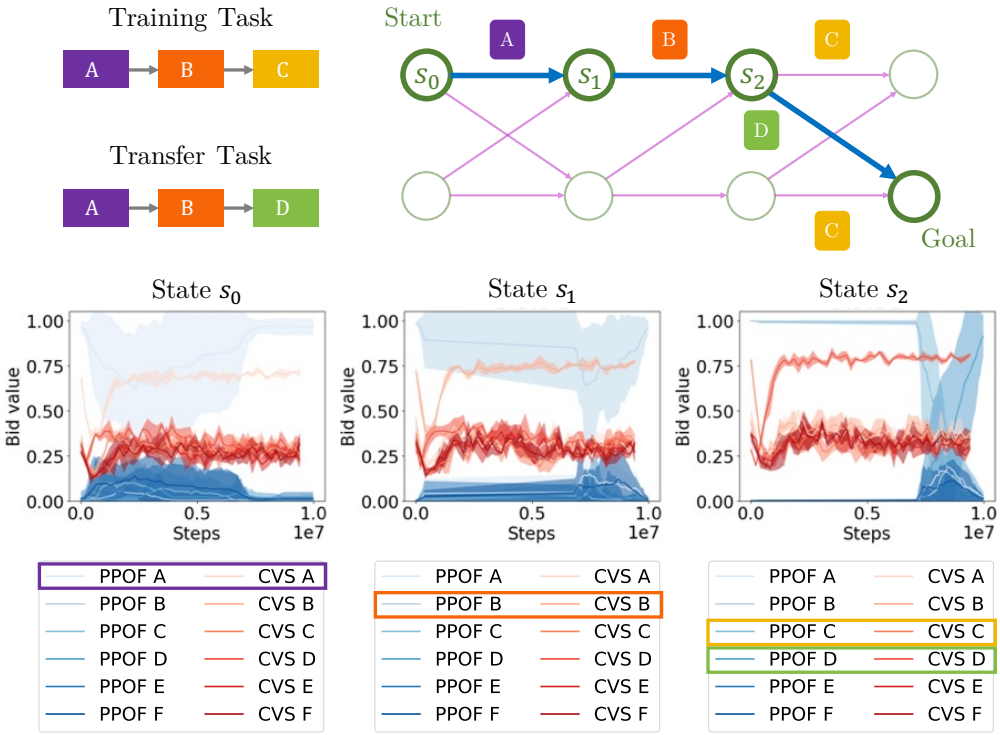


Figure 7.5: **How the decision mechanisms change during transfer.** Shown the three states of the decision sequence. The optimal last decision must change from action C (purple) to action D (green). CVS modifies its bids independently. The bids for PPOF are coupled together across decision mechanisms and across time.

An Enumeration of Transfer Problems

Similar to how analysis of d-separation is conducted with triplets of nodes, we enumerated all possible topologies of triplets of decisions: *linear chain*, *common ancestor*, and *common descendant* (Fig. 7.6, left column). For each topology we enumerated all ways of making an isolated change to an optimal decision sequence. The *common ancestor* and *common descendant* topologies involve multi-task training for two decision sequences of length two, while *linear chain* involves single-task training for one decision sequence of length three. For example, in Fig. 7.6, the optimal decision sequence for the *linear chain* training task is $A \rightarrow B \rightarrow C$. For each topology we have a training task and three independent transfer tasks. Each transfer task represents a different way to modify the MDP of the training task. This single comprehensive task suite (Fig. 7.6) enables us to ask a wide range of questions. The answers to the questions that follow are scoped only to our stated experimental setup.

Does dynamic modularity improve transfer efficiency? Yes, at least in these experiments. For each of the nine transfer settings (rightmost three columns) in Fig. 7.6, CVS (red)

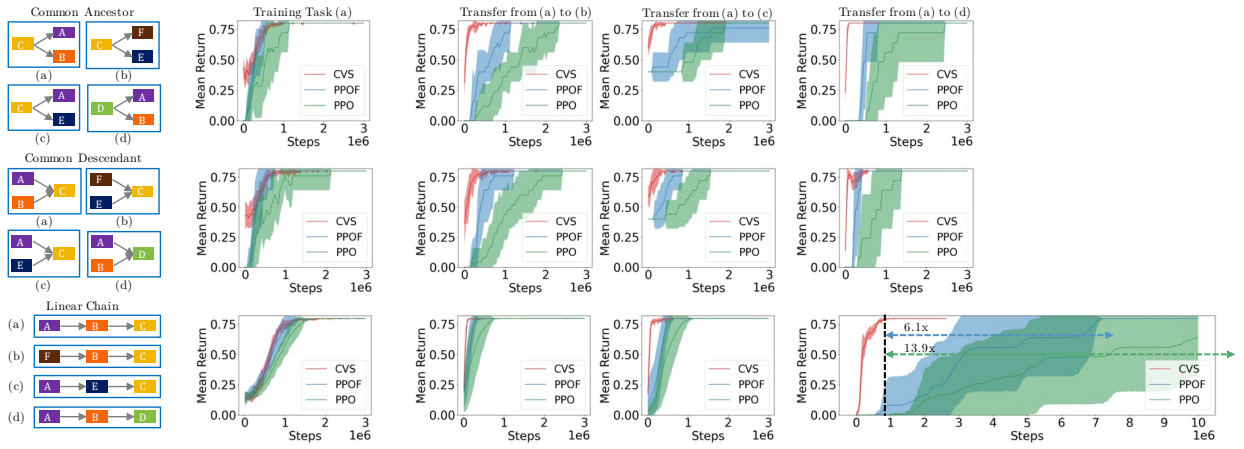


Figure 7.6: **Transfer problems involving triplets of decisions.** For each task topology (leftmost column) we have a training task, labeled (a) and three independent transfer tasks, labeled (b,c,d). Each transfer task is a different way to modify the training task's MDP. CVS consistently exhibits higher sample efficiency than both PPO and PPOF showing that dynamic modularity correlates with more efficient transfer. Notably the gap between CVS and the other methods in the bottom-right (e.g. 13.9x more efficient than PPO) is so wide that we had to extend the chart width. We set the convergence time as the first time after which the return deviates by no more than $\varepsilon = 0.01$ from the optimal return, 0.8, for 30 epochs of training. Shown are runs across ten seeds.

transfers consistently more efficiently than both PPO (green) and PPOF (blue), despite having comparable training efficiency in the training task (second column from left). The variance among the different runs is also lower for CVS.

How does where a decision needs to be modified in the decision sequence affect transfer efficiency? The improvement in transfer efficiency is especially pronounced in the trend shown in the bottom row of Fig. 7.6 for *linear chain*. The later the decision that needs to be modified appears in the decision sequence, the wider the gap between CVS and the other two methods, to the point that we had to widen the plot width. Our theory (Thm. 7.4.1) offers one possible explanation. Considering the bottom-right plot of Fig. 7.6, the transfer task requires modifying the last decision and keeping the previous two the same. But the lack of independent gradients and parameters in PPO and PPOF seems to have affected correct decision mechanisms in the first two steps based on the errors encountered by the decision mechanism in the last step, seemingly causing the previous decision mechanisms to "unlearn" originally optimal behavior, then relearn the correct behavior again, as shown in the plots for "state s_0" and "state s_1" in Fig. 7.5 for PPOF. This slow unlearning and relearning seems to be a reason for the lower transfer efficiency of PPO and PPOF. It is as if Colette in Gusteau's taqueria (§7.1) stopped heating tortillas because of the angry reviews about meat contamination but then realized that she should still be heating tortillas after all.

Does dynamic modularity enable independent modification of decision mechanisms in practice? While theory tells us that decision mechanisms can be modified independently within a single credit assignment update, in practice transfer learning requires multiple credit assignment updates to converge. Across multiple credit assignment updates, the decision mechanisms would no longer be independent, even for algorithms that exhibit dynamic modularity, but it is also expected that the functions of a learner should learn to work together over the course of learning in any case. Nonetheless, Fig. 7.5 shows that the lack of a softmax tying the bids of CVS together enables them to change more independently and rapidly than PPOF.

How much of transfer efficiency is due to modular credit assignment than network factorization? This question pits our theory against a competing explanation: that network factorization alone (represented by PPOF) is responsible for improved transfer efficiency. Though PPOF is more efficient than PPO in training and transfer, PPOF is consistently less efficient than CVS in transfer while being similarly efficient in training. This suggests that network factorization is not a sufficient explanation, leaving our theory of dynamic modularity still standing.

Modularity and Forgetting

A desirable consequence of having the capacity to independently modify learnable mechanisms is the ability to *not* modify mechanisms that need not be modified: we would not want the agent to forget optimal behavior in one context when it trains on a different task in a different context. We now test whether dynamic modularity contributes to this ability. The experimental setup is shown in Fig. 7.7. There are four possible values for the action, A, B, C, D. In task (a), the optimal decision sequence is $A \to C$, starting at state s_0 and passing through state s_2, which has a context bit flipped to 0. In task (b), the optimal decision sequence is $B \to D$, starting at state s_1 and passing through state s_2, which has a context bit flipped to 1. Though the optimal states for task (a) are disjoint from the optimal states for task (b), the decision mechanisms corresponding to A, B, C, D are present for both tasks. We first train on task (a), then transfer from (a) to (b), then transfer back from (b) to (a).

Does dynamic modularity improve the agent's ability to preserve optimal behavior on a previous task after having trained to convergence on a different task? To test this, we compare CVS and PPO's sample efficiency when transferring back from (b) to (a). Fig. 7.7 shows that even when both CVS and PPO have similar sample efficiency when initially training on task (a), CVS is more than ten times more sample efficient than PPO when transferring back from (b) to (a). Our explanation for this phenomenon is that the lack of algorithmic independence in the decision mechanisms of PPO causes the decision mechanisms for actions A and C to be significantly modified when PPO transfers from (a) to

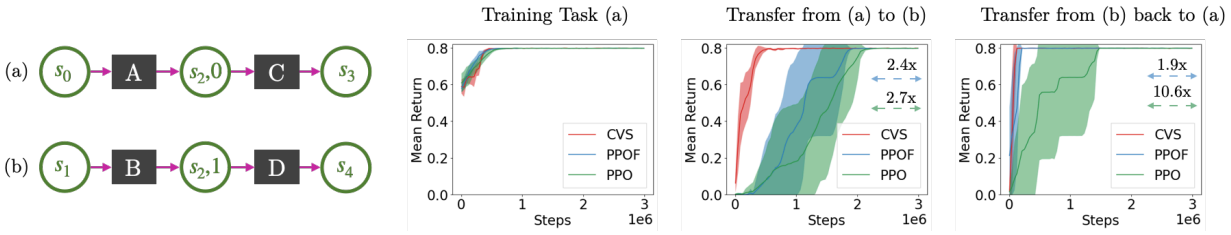


Figure 7.7: **Modularity and forgetting.** The optimal solutions for tasks (a) and (b) involve a disjoint set of decisions: $A \to C$ for task (a) and $B \to D$ for task (b). We first train on task (a), then transfer from (a) to (b), then transfer back from (b) to (a). The purpose of this experiment is to test whether dynamic modularity improve the agent's ability to preserve optimal behavior on a previous task after having trained to convergence on a different task in a different context. While both CVS and PPO have similar sample efficiency when initially training on task (a), CVS is more than ten times more sample efficient than PPO when transferring back from (b) to (a), suggesting that PPO "forgot" the optimal behavior for task (a) when training on task (b), which is not the kind of forgetting we want in learning agents.

(b), even when these actions do not even participate in the optimal decision sequence for task (b). The low sample efficiency when transferring back from (b) to (a) suggests that PPO "forgot" the optimal behavior for task (a) when training on task (b), which is not the kind of forgetting we want in flexibly adaptable agents.

How much of this ability to preserve previously optimal behavior due to modular credit assignment than network factorization? PPOF is similarly inefficient as PPO compared to CVS in transferring from (a) to (b), which is consistent with our findings from §7.7. Interestingly, PPOF seems to be just as efficient at transferring back from (b) to (a) as CVS, which seems to suggest that the primary cause for the forgetfulness of PPO, at least in this experiment, is less due to lack of independent gradients but more to lack of network factorization. This experiment suggests a need for an explanatory theory to identify under which circumstances independent gradients are more influential to flexible adaptation than network factorization, and vice versa, as well as a means for quantifying the degree of influence each has.

7.8 Discussion

The hypothesis that modularity can enable flexible adaptation requires a method for determining whether a learning system is modular. This chapter has contributed the modularity criterion (Thm. 7.5.2) as such a method.

The consistency of how dynamic modularity in on-policy reinforcement learning correlates with higher transfer efficiency in our experiments suggests a need for future work to provide

an explanatory theory for exactly how dynamic modularity contributes to flexible adaptation as well as to test whether the same phenomenon can be observed with other classes of learning algorithms, other transfer problems, and other domains. The modularity criterion is a binary criterion on algorithmic independence or lack thereof, but our experiments also suggest a need for future work in quantifying algorithmic causal influence if we want to relax the criterion to a softer penalty on algorithmic mutual information. Moreover, our work highlights a need for a formalism that allows for meaningful statements about independence among concrete strings within a specific Turing machine, rather than statements that only hold asymptotically.

Learning algorithms are only one example of the more general concept of the dynamic computational graph introduced in this chapter, which Lemma 7.5.1 shows can be used to analyze the algorithmic independence of functions that evolve over time. The connection we have established among credit assignment, modularity, and algorithmic information theory, in particular the link between learning algorithms and algorithmic causality, opens many opportunities for future work, such as new ways of formalizing inductive bias in the algorithmic causal structure of learning systems *and* the learning algorithms that modify them.

Part IV

Conclusion

Our goal in this thesis has been to take a step towards building machines that automatically model and manipulate systems. I have argued that proper modeling and manipulation of systems should exhibit combinatorial generalization: the capacity to generalize across the space of different combinations of entities, transformations, and choices. I have identified the core bottleneck to achieving this to be the capacity to learn reusable abstractions: representations of entities, transformations, and choices that can be independently recombined in new contexts. Machine learning lacks a useful mathematical framework for describing this notion of independence, and I have argued that naively factorizing monolithic networks does not effectively prevent spurious correlations. The transition from electronic circuits to software required solving a similar problem. The solution for achieving combinatorial generalization in computing was to implement the principle of separation of concerns via contextual refinement towards designed specifications. The question this thesis has tackled is whether the method of contextual refinement towards learned specifications can enable neural networks to learn representations that can be similarly reused.

I have investigated this question along three dimensions of system interaction: entities, transformations, and choices. For entities, I showed that implementing contextual refinement as iterative clustering enables neural networks to learn to infer representations of objects that can be reused in the context of other objects. For transformations, I showed that implementing context refinement by restricting neural blocks to only partially modify their input enables these learned functions to be composed with other functions in novel ways. For choices, I showed that training policies as a set of choice functions that optimize auction utility enables these choice functions to be independently updated for more efficient transfer.

I have made a case in this thesis that the future of AI research has much to gain from taking the analogy between electronic circuits and neural circuits seriously. Contextual refinement for encapsulation is an idea invented uniquely for circuits and software, and encapsulation as a notion of independence does not have a rigorous formalization in the mathematics of machine learning. This thesis showed that identifying contextual refinement as the mechanism behind how we implemented encapsulation for enabling combinatorial generalization in software opens the path to implementing the same kind of mechanism for enabling combinatorial generalization in neural networks.

But the insights we get from this analogy does not have to stop there. Most recently, the field has begun to take steps towards a neural variant of von Neumann architecture, where a Transformer interacts with an external data source similarly to how a CPU interacts with the hard drive [173, 52, 112, 222]. We can then use this connection to draw parallels between how the field of computing evolved after the von Neumann machine to give us insight on how the field of deep learning will evolve as well. As an example, we have yet to invent the analogue of Assembly let alone high-level programming languages on top of our neural circuits. The best AI models today are large language models [238] and interacting with them by typing text is like interacting with the early computers of the 1960s through command-line interfaces. We have yet to invent the analogue to the graphical user interface of the deep learning era.

As we begin the transition beyond designing neural circuits to writing neural software, it is worth keeping in mind the key difference between electronic circuits and computer software.

The difference is that software is fundamentally a user interface. Though software ultimately grounds out in circuits, it is a series of abstraction layers on top of circuits that enable humans to model and manipulate systems by writing code. As a user interface, software has also introduced a new set of design criteria beyond those for designing circuits, specifically that software needs to be "safe from bugs, easy to understand, and ready for change" [217]. These design criteria suggest that, though anything that can be achieved by software can be achieved with a complex electronic circuit, the reason why we developed software over the last century rather than stayed with circuits is that problem solving is inherently human-centric: it matters not only what problems software can solve but that it solves it in a way that can be understood, modified, and used by humans.

Understanding the path from circuits to software may give us valuable insights on the future of deep learning and how to address its present limitations. Massive neural networks are similarly inscrutable as complex electronic circuits. The design criteria of "safe from bugs, easy to understand, and ready for change" for good software are also the same criteria for building safe AI systems of the future: aligned, interpretable, and extensible. If we want learning machines to help us solve problems, then we need to similarly design AI systems with the human in mind. We need to go beyond training neural circuits and begin creating neural software abstractions.

Bibliography

[1] David Abel et al. "Goal-Based Action Priors". In: *ICAPS*. AAAI Press, 2015, pp. 306–314.

[2] Harold Abelson and Gerald Jay Sussman. *Structure and interpretation of computer programs*. The MIT Press, 1996.

[3] Joshua Achiam, Ethan Knight, and Pieter Abbeel. "Towards characterizing divergence in deep q-learning". In: *arXiv preprint arXiv:1903.08894* (2019).

[4] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. "A learning algorithm for Boltzmann machines". In: *Cognitive science* 9.1 (1985), pp. 147–169.

[5] Akshay Agrawal et al. "Differentiable convex optimization layers". In: *arXiv preprint arXiv:1910.12430* (2019).

[6] Pulkit Agrawal et al. "Learning to poke by poking: Experiential learning of intuitive physics". In: *Advances in Neural Information Processing Systems*. 2016, pp. 5074–5082.

[7] Anurag Ajay et al. "Combining Physical Simulators and Object-Based Networks for Control". In: *arXiv:1904.06580* (2019).

[8] John Aldrich. "Autonomy". In: *Oxford Economic Papers* 41.1 (1989), pp. 15–34.

[9] Ferran Alet, Tomás Lozano-Pérez, and Leslie P Kaelbling. "Modular meta-learning". In: *arXiv:1806.10166* (2018).

[10] Luis B Almeida. "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment". In: *Artificial neural networks: concept learning*. 1990, pp. 102–111.

[11] Brandon Amos and J Zico Kolter. "Optnet: Differentiable optimization as a layer in neural networks". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 136–145.

[12] James A Anderson and Geoffrey E Hinton. "Models of information processing in the brain". In: *Parallel models of associative memory*. Psychology Press, 2014, pp. 33–74.

[13] John Robert Anderson. "The adaptive character of thought". In: Psychology Press, 1990. Chap. 5, pp. 191–230.

[14] Jacob Andreas et al. "Neural module networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 39–48.

[15] Marcin Andrychowicz et al. "Learning to learn by gradient descent by gradient descent". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3981–3989.

[16] Jose A Arjona-Medina et al. "Rudder: Return decomposition for delayed rewards". In: *Advances in Neural Information Processing Systems*. 2019, pp. 13544–13555.

[17] Jimmy Ba et al. "Using fast weights to attend to the recent past". In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 4331–4339.

[18] Pierre-Luc Bacon, Jean Harb, and Doina Precup. "The Option-Critic Architecture." In: *AAAI*. 2017, pp. 1726–1734.

[19] Dzmitry Bahdanau et al. "Systematic Generalization: What Is Required and Can It Be Learned?" In: *arXiv preprint arXiv:1811.12889* (2018).

[20] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "Deep equilibrium models". In: *arXiv preprint arXiv:1909.01377* (2019).

[21] Shaojie Bai, Vladlen Koltun, and J Zico Kolter. "Stabilizing equilibrium models by Jacobian regularization". In: *arXiv preprint arXiv:2106.14342* (2021).

[22] David Balduzzi. "Cortical prediction markets". In: *arXiv preprint arXiv:1401.1465* (2014).

[23] David Balduzzi et al. "Smooth markets: A basic mechanism for organizing gradient-based learners". In: *arXiv preprint arXiv:2001.04678* (2020).

[24] Victor Bapst et al. "Structured agents for physical construction". In: *arXiv:1904.03177* (2010), pp. 464–474.

[25] Andrew G Barto and Sridhar Mahadevan. "Recent advances in hierarchical reinforcement learning". In: *Discrete event dynamic systems* 13.1-2 (2003), pp. 41–77.

[26] Dhruv Batra et al. "Rearrangement: A challenge for embodied AI". In: *arXiv preprint arXiv:2011.01975* (2020).

[27] Peter Battaglia et al. "Interaction networks for learning about objects, relations and physics". In: *Advances in Neural Information Processing Systems*. 2016, pp. 4502–4510.

[28] Peter W Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[29] Eric B Baum. "Toward a model of mind as a laissez-faire economy of idiots". In: *ICML*. 1996, pp. 28–36.

[30] R Bellmann. "Dynamic programming princeton university press". In: *Princeton, NJ* (1957).

[31] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[32] B. Bird et al. *Ratatouille*. 2007.

[33] Matthew M Botvinick, Yael Niv, and Andrew C Barto. "Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective". In: *Cognition* 113.3 (2009), pp. 262–280.

[34] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. "Stochastic dynamic programming with factored representations". In: *Artif. Intell.* 121.1-2 (2000), pp. 49–107.

[35] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. "Exploiting Structure in Policy Construction". In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'95. Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc., 1995, pp. 1104–1111.

[36] Valentino Braitenberg. *Vehicles: Experiments in synthetic psychology*. 1986.

[37] Rodney A Brooks. "Intelligence without representation". In: *Artificial intelligence* 47.1-3 (1991), pp. 139–159.

[38] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[39] Sébastien Bubeck et al. "Sparks of artificial general intelligence: Early experiments with gpt-4". In: *arXiv preprint arXiv:2303.12712* (2023).

[40] Rudy Bunel et al. "Leveraging grammar and reinforcement learning for neural program synthesis". In: *arXiv:1805.04276* (2018).

[41] Christopher P Burgess et al. "MONet: Unsupervised Scene Decomposition and Representation". In: *arXiv:1901.11390* (2019).

[42] Jonathon Cai, Richard Shin, and Dawn Song. "Making neural programming architectures generalize via recursion". In: *arXiv:1704.06611* (2017).

[43] Frederick Callaway et al. "Mouselab-MDP: A new paradigm for tracing how people plan". In: (2017).

[44] Berk Calli et al. "The ycb object and model set: Towards common benchmarks for manipulation research". In: *2015 international conference on advanced robotics (ICAR)*. IEEE. 2015, pp. 510–517.

[45] Paco Calvo and John Symons. *The Architecture of Cognition: Rethinking Fodor and Pylyshyn's Systematicity Challenge*. MIT Press, 2014.

[46] Susan Carey. "Why Theories of Concepts Should Not Ignore the Problem of Acquisition". In: *The conceptual mind: New directions in the study of concepts* (2015), p. 415.

[47] Pablo Samuel Castro, Prakash Panangaden, and Doina Precup. "Equivalence relations in fully and partially observable Markov decision processes". In: *Twenty-First International Joint Conference on Artificial Intelligence*. Citeseer. 2009.

[48] Augustin-Louis Cauchy. "Résumé d'un mémoir sur la mécanique céleste et sur un nouveau calcul appelé calcul des limites". In: *Oeuvres Complétes d'Augustun Cauchy* 12.48-112 (1831), p. 3.

[49] Gregory J Chaitin. "A theory of program size formally identical to information theory". In: *Journal of the ACM (JACM)* 22.3 (1975), pp. 329–340.

[50] Gregory J Chaitin. "On the length of programs for computing finite binary sequences". In: *Journal of the ACM (JACM)* 13.4 (1966), pp. 547–569.

[51] Chien-Yi Chang et al. "Procedure planning in instructional videos". In: *European Conference on Computer Vision*. Springer. 2020, pp. 334–350.

[52] Michael Chang. *Von Neumann : LLM :: Doug Engelbart : ?* Twitter. Twitter. Feb. 2023. URL: `https://twitter.com/mmmbchang/status/1623392148786712577`.

[53] Michael Chang, Thomas L Griffiths, and Sergey Levine. "Object Representations as Fixed Points: Training Iterative Refinement Algorithms with Implicit Differentiation". In: *arXiv preprint arXiv:2207.00787* (2022).

[54] Michael Chang et al. "Decentralized Reinforcement Learning: Global Decision-Making via Local Economic Transactions". In: *arXiv preprint arXiv:2007.02382* (2020).

[55] Michael Chang et al. "Modularity in Reinforcement Learning via Algorithmic Independence in Credit Assignment". In: *Learning to Learn-Workshop at ICLR 2021*. 2021.

[56] Michael Chang et al. "Neural Constraint Satisfaction: Hierarchical Abstraction for Combinatorial Generalization in Object Rearrangement". In: *arXiv preprint* (2023). eprint: `arXiv:2303.11373`.

[57] Michael B Chang et al. "A compositional object-based approach to learning physical dynamics". In: *arXiv:1612.00341* (2016).

[58] Michael B Chang et al. "Automatically composing representation transformations as a means for generalization". In: *arXiv preprint arXiv:1807.04640* (2018).

[59] Mark Chen et al. "Generative pretraining from pixels". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1691–1703.

[60] Ricky TQ Chen et al. "Neural ordinary differential equations". In: *arXiv preprint arXiv:1806.07366* (2018).

[61] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. "Settling the complexity of computing two-player Nash equilibria". In: *Journal of the ACM (JACM)* 56.3 (2009), pp. 1–57.

[62] Xi Chen et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2172–2180.

[63] Xinyun Chen, Chang Liu, and Dawn Song. "Learning Neural Programs To Parse Programs". In: *arXiv:1706.01284* (2017).

[64] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. *Minimalistic Gridworld Environment for OpenAI Gym*. `https://github.com/maximecb/gym-minigrid`. 2018.

[65] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[66] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv:1412.3555* (2014).

[67] Erwin Coumans and Yunfei Bai. "Pybullet, a python module for physics simulation for games, robotics and machine learning". In: (2016).

[68] Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. "Are Neural Nets Modular? Inspecting Functional Modularity Through Differentiable Weight Masks". In: *arXiv preprint arXiv:2010.02066* (2020).

[69] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. "The complexity of computing a Nash equilibrium". In: *SIAM Journal on Computing* 39.1 (2009), pp. 195–259.

[70] Peter Dayan et al. "The Helmholtz machine". In: *Neural computation* 7.5 (1995), pp. 889–904.

[71] Eyal Dechter et al. "Bootstrap Learning via Modular Concept Discovery." In: *IJCAI*. 2013, pp. 1302–1309.

[72] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22.

[73] Emily L Denton et al. "Unsupervised learning of disentangled representations from video". In: *Advances in neural information processing systems*. 2017, pp. 4414–4423.

[74] Coline Devin et al. "Learning modular neural network policies for multi-task and multi-robot transfer". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 2169–2176.

[75] Coline Devin et al. "Self-supervised goal-conditioned pick and place". In: *arXiv preprint arXiv:2008.11466* (2020).

[76] Carlos Diuk, Andre Cohen, and Michael L. Littman. "An object-oriented representation for efficient reinforcement learning". In: *ICML*. Vol. 307. ACM International Conference Proceeding Series. ACM, 2008, pp. 240–247.

[77] Carl Doersch. "Tutorial on variational autoencoders". In: *arXiv preprint* (2016). eprint: `arXiv:1606.05908`.

[78] Yilun Du and Igor Mordatch. "Implicit generation and generalization in energy-based models". In: *arXiv preprint arXiv:1903.08689* (2019).

[79] Yilun Du and Karthik Narasimhan. "Task-Agnostic Dynamics Priors for Deep Reinforcement Learning". In: *arXiv:1905.04819* (2019).

[80] David Duvenaud, J. Zico Kolter, and Matthew Johnson. "Deep Implicit Layers Tutorial - Neural ODEs, Deep Equilibirum Models, and Beyond". In: *Neural Information Processing Systems Tutorial* (2020).

[81] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. "Relational reinforcement learning". In: *Machine learning* 43.1-2 (2001), pp. 7–52.

[82] Frederik Ebert et al. "Robustness via retrying: Closed-loop robotic manipulation with self-supervised learning". In: *arXiv:1810.03043* (2018).

[83] Frederik Ebert et al. "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control". In: *arXiv:1812.00568* (2018).

[84] Kevin Ellis et al. "Learning Libraries of Subroutines for Neurally–Guided Bayesian Program Induction". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 7805–7815.

[85] Gamaleldin F Elsayed et al. "SAVi++: Towards End-to-End Object-Centric Learning from Real-World Videos". In: *arXiv preprint arXiv:2206.07764* (2022).

[86] Scott Emmons et al. "Sparse graphical memory for robust planning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5251–5262.

[87] The Editors of Encyclopaedia Britannica. *Bombe*. Accessed on April 13, 2023. 2023. URL: https://www.britannica.com/topic/Bombe.

[88] SM Ali Eslami et al. "Attend, infer, repeat: Fast scene understanding with generative models". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3225–3233.

[89] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. "Search on the Replay Buffer: Bridging Planning and Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/5c48ff18e0a47baaf81d8b8ea51eec92-Paper.pdf.

[90] Chrisantha Fernando et al. "Pathnet: Evolution channels gradient descent in super neural networks". In: *arXiv:1701.08734* (2017).

[91] John K Feser et al. "Differentiable Functional Program Interpreters". In: *arXiv preprint* (2016). eprint: arXiv:1611.01988.

[92] Daniel Filan et al. "Clusterability in Neural Networks". In: *arXiv preprint* (2021). eprint: arXiv:2103.03386.

[93] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1126–1135.

[94] Chelsea Finn, Ian Goodfellow, and Sergey Levine. "Unsupervised learning for physical interaction through video prediction". In: *Advances in neural information processing systems*. 2016, pp. 64–72.

[95] Chelsea Finn and Sergey Levine. "Deep visual foresight for planning robot motion". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 2786–2793.

[96] Jerry A Fodor and Ernest Lepore. *The compositionality papers*. Oxford University Press, 2002.

[97] Jerry A Fodor and Zenon W Pylyshyn. "Connectionism and cognitive architecture: A critical analysis". In: *Cognition* 28.1-2 (1988), pp. 3–71.

[98] V. Fomin et al. *High-level library to help with training neural networks in PyTorch*. `https://github.com/pytorch/ignite`. 2020.

[99] Martin Ford. *Architects of Intelligence: The truth about AI from the people building it*. Packt Publishing Ltd, 2018.

[100] Katerina Fragkiadaki et al. "Learning visual predictive models of physics for playing billiards". In: *arXiv:1511.07404* (2015).

[101] Kevin Frans et al. "Meta learning shared hierarchies". In: *arXiv:1710.09767* (2017).

[102] Karl Friston. "The free-energy principle: a unified brain theory?" In: *Nature Reviews Neuroscience* 11.2 (2010), pp. 127–138.

[103] Justin Fu et al. "Diagnosing bottlenecks in deep q-learning algorithms". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2021–2030.

[104] Justin Fu et al. "Variational inverse control with events: A general framework for data-driven reward definition". In: *Advances in Neural Information Processing Systems*. 2018, pp. 8538–8547.

[105] Samy Wu Fung et al. "Fixed point networks: Implicit depth models with Jacobian-free backprop". In: *arXiv preprint arXiv:2103.12803* (2021).

[106] Péter Gács, John T Tromp, and Paul MB Vitányi. "Algorithmic statistics". In: *IEEE Transactions on Information Theory* 47.6 (2001), pp. 2443–2463.

[107] Yaroslav Ganin et al. "Synthesizing Programs for Images using Reinforced Adversarial Learning". In: *arXiv:1804.01118* (2018).

[108] Natalia Gardiol and Leslie Kaelbling. "Envelope-based Planning in Relational MDPs". In: *Advances in Neural Information Processing Systems*. Ed. by S. Thrun, L. Saul, and B. Schölkopf. Vol. 16. MIT Press, 2003. URL: `https://proceedings.neurips.cc/paper/2003/file/4a06d868d044c50af0cf9bc82d2fc19f-Paper.pdf`.

[109] Alexander L Gaunt et al. "Differentiable programs with neural libraries". In: *arXiv preprint* (2016). eprint: `arXiv:1611.02109`.

[110] Ian Gemp et al. "D3C: Reducing the Price of Anarchy in Multi-Agent Learning". In: *arXiv preprint arXiv:2010.00575* (2020).

[111] Zhengyang Geng et al. "On Training Implicit Models". In: *Advances in Neural Information Processing Systems* 34 (2021).

[112] Angeliki Giannou et al. "Looped transformers as programmable computers". In: *arXiv preprint arXiv:2301.13196* (2023).

[113] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[114] Vik Goel, Jameson Weng, and Pascal Poupart. "Unsupervised Video Object Segmentation for Deep Reinforcement Learning". In: *arXiv:1805.07780* (2018).

[115] Tejas Gokhale et al. "Cooking with blocks: A recipe for visual reasoning on image-pairs". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019, pp. 5–8.

[116] Noah D Goodman, Joshua B. Tenenbaum, and The ProbMods Contributors. *Probabilistic Models of Cognition*. `http://probmods.org/v2`. Accessed: 2021-1-29. 2016.

[117] Alison Gopnik and Henry M Wellman. "Reconstructing constructivism: Causal models, Bayesian learning mechanisms, and the theory theory." In: *Psychological bulletin* 138.6 (2012), p. 1085.

[118] Stephen Gould, Richard Hartley, and Dylan Campbell. "Deep declarative networks: A new hope". In: *arXiv preprint arXiv:1909.04866* (2019).

[119] Anirudh Goyal et al. "Neural production systems". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 25673–25687.

[120] Anirudh Goyal et al. "Object files and schemata: Factorizing declarative and procedural knowledge in dynamical systems". In: *arXiv preprint arXiv:2006.16225* (2020).

[121] Anirudh Goyal et al. "Recurrent independent mechanisms". In: *arXiv preprint* (2019). eprint: `arXiv:1909.10893`.

[122] Anirudh Goyal et al. "Reinforcement Learning with Competitive Ensembles of Information Constrained Primitives". In: *arXiv preprint arXiv:1906.10667* (2019).

[123] Erin Grant et al. "Recasting gradient-based meta-learning as hierarchical Bayes". In: *arXiv preprint arXiv:1801.08930* (2018).

[124] Alex Graves. "Adaptive computation time for recurrent neural networks". In: *arXiv preprint arXiv:1603.08983* (2016).

[125] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural turing machines". In: *arXiv preprint* (2014). eprint: `arXiv:1410.5401`.

[126] Alex Graves et al. "Hybrid computing using a neural network with dynamic external memory". In: *Nature* 538.7626 (2016), p. 471.

[127] Edward Grefenstette et al. "Learning to transduce with unbounded memory". In: *Advances in Neural Information Processing Systems*. 2015, pp. 1828–1836.

[128] Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. "Binding via reconstruction clustering". In: *arXiv:1511.06418* (2015).

[129] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. "On the Binding Problem in Artificial Neural Networks". In: *arXiv preprint arXiv:2012.05208* (2020).

[130] Klaus Greff, Sjoerd Van Steenkiste, and Jürgen Schmidhuber. "Neural expectation maximization". In: *arXiv preprint arXiv:1708.03498* (2017).

[131] Klaus Greff et al. "Multi-Object Representation Learning with Iterative Variational Inference". In: *arXiv:1903.00450* (2019).

[132] Klaus Greff et al. "Multi-object representation learning with iterative variational inference". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2424–2433.

[133] Thomas L Griffiths, Falk Lieder, and Noah D Goodman. "Rational use of cognitive resources: Levels of analysis between the computational and the algorithmic". In: *Topics in cognitive science* 7.2 (2015), pp. 217–229.

[134] Thomas L Griffiths et al. "Doing more with less: Meta-reasoning and meta-learning in humans and machines". In: *Current Opinion in Behavioral Sciences* (Jan. 2019).

[135] Oliver Groth et al. "Shapestacks: Learning vision-based physical intuition for generalised object stacking". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 702–717.

[136] Computation Structures Group. *Digital Abstraction*. `https://computationstructures.org/notes/digitalabstraction/notes.html`. [Accessed: April 21, 2023]. 2023.

[137] Carlos Guestrin et al. "Efficient solution algorithms for factored MDPs". In: *Journal of Artificial Intelligence Research* 19 (2003), pp. 399–468.

[138] Carlos Guestrin et al. "Generalizing Plans to New Environments in Relational MDPs". In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI'03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003, pp. 1003–1010.

[139] Siyuan Guo et al. "Causal de Finetti: On the Identification of Invariant Causal Structure in Exchangeable Data". In: *arXiv preprint arXiv:2203.15756* (2022).

[140] Abhishek Gupta et al. "Meta-Reinforcement Learning of Structured Exploration Strategies". In: *arXiv:1802.07245* (2018).

[141] Abhishek Gupta et al. "Unsupervised Meta-Learning for Reinforcement Learning". In: *arXiv:1806.04640* (2018).

[142] David Ha, Andrew Dai, and Quoc V Le. "Hypernetworks". In: *arXiv:1609.09106* (2016).

[143] David Ha and Jürgen Schmidhuber. "World models". In: *arXiv preprint* (2018). eprint: `arXiv:1803.10122`.

[144] Trygve Haavelmo. "The probability approach in econometrics". In: *Econometrica: Journal of the Econometric Society* (1944), pp. iii–115.

[145] Danijar Hafner et al. "Learning Latent Dynamics for Planning from Pixels". In: *arXiv:1811.04551* (2018).

[146] Jessica B Hamrick et al. "Metacontrol for adaptive imagination-based optimization". In: *arXiv:1705.02670* (2017).

[147] Philippe Hansen-Estruch et al. "Bisimulation Makes Analogies in Goal-Conditioned Reinforcement Learning". In: *arXiv preprint arXiv:2204.13060* (2022).

[148] Nicholas Hay et al. "Selecting computations: Theory and applications". In: *arXiv preprint* (2014). eprint: `arXiv:1408.2048`.

[149] Kaiming He et al. "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[150] Martin Heusel et al. "Gans trained by a two time-scale update rule converge to a local nash equilibrium". In: *Advances in neural information processing systems* 30 (2017).

[151] Irina Higgins et al. "Towards a Definition of Disentangled Representations". In: *arXiv preprint arXiv:1812.02230* (2018).

[152] Geoffrey Hinton, Nicholas Frosst, and Sara Sabour. "Matrix capsules with EM routing". In: (2018).

[153] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.

[154] John H Holland. "Properties of the bucket brigade". In: *Proceedings of an International Conference on Genetic Algorithms and their Applications 1-7*. 1985.

[155] Robert C Holte and Berthe Y Choueiry. "Abstraction and reformulation in artificial intelligence". In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 358.1435 (2003), pp. 1197–1204.

[156] John J Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558.

[157] De-An Huang et al. "Neural task graphs: Generalizing to unseen tasks from a single video demonstration". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 8565–8574.

[158] Zhichun Huang, Shaojie Bai, and J Zico Kolter. "Implicit^2: Implicit Layers for Implicit Representations". In: *Advances in Neural Information Processing Systems* 34 (2021).

[159] Kazuki Irie et al. "Going beyond linear transformers with recurrent fast weight programmers". In: *Advances in Neural Information Processing Systems* 34 (2021).

[160] Robert A Jacobs et al. "Adaptive mixtures of local experts". In: *Neural computation* 3.1 (1991), pp. 79–87.

[161] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. "Spatial transformer networks". In: *Advances in neural information processing systems*. 2015, pp. 2017–2025.

[162] Michael Janner, Qiyang Li, and Sergey Levine. "Offline reinforcement learning as one big sequence modeling problem". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 1273–1286.

[163] Michael Janner, Karthik Narasimhan, and Regina Barzilay. "Representation learning for grounded spatial reasoning". In: *Transactions of the Association for Computational Linguistics* 6 (2018), pp. 49–61.

[164] Michael Janner et al. "Reasoning About Physical Interactions with Object-Oriented Prediction and Planning". In: *arXiv:1812.10972* (2018).

[165] Michael Janner et al. "When to trust your model: Model-based policy optimization". In: *Advances in Neural Information Processing Systems* 32 (2019).

[166] Dominik Janzing and Bernhard Schölkopf. "Causal inference using the algorithmic Markov condition". In: *IEEE Transactions on Information Theory* 56.10 (2010), pp. 5168–5194.

[167] Armand Joulin and Tomas Mikolov. "Inferring algorithmic patterns with stack-augmented recurrent nets". In: *Advances in neural information processing systems*. 2015, pp. 190–198.

[168] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[169] Łukasz Kaiser and Ilya Sutskever. "Neural gpus learn algorithms". In: *arXiv:1511.08228* (2015).

[170] Dmitry Kalashnikov et al. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation". In: *arXiv preprint arXiv:1806.10293* (2018).

[171] Pentti Kanerva. "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors". In: *Cognitive computation* (2009).

[172] Ken Kansky et al. "Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics". In: *arXiv:1706.04317* (2017).

[173] Andrej Karpathy. *The analogy between GPTs of today to the CPUs of early days of computing*. Twitter. Twitter. Apr. 2023. URL: `https://twitter.com/karpathy/status/1644183721405464576`.

[174] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv:1412.6980* (2014).

[175] Diederik P Kingma and Max Welling. "Auto-encoding variational Bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[176] Thomas Kipf, Elise van der Pol, and Max Welling. "Contrastive Learning of Structured World Models". In: *arXiv preprint arXiv:1911.12247* (2019).

[177] Thomas Kipf et al. "Conditional Object-Centric Learning from Video". In: *arXiv preprint arXiv:2111.12594* (2021).

[178] Louis Kirsch, Julius Kunze, and David Barber. "Modular Networks: Learning to Decompose Neural Computation". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2414–2423.

[179] Teuvo Kohonen. "Correlation matrix memories". In: *IEEE transactions on computers* 100.4 (1972), pp. 353–359.

[180] Teuvo Kohonen. *Self-organization and associative memory*. Vol. 8. Springer Science & Business Media, 2012.

[181] Andrei N Kolmogorov. "Three approaches to the quantitative definition ofinformation'". In: *Problems of information transmission* 1.1 (1965), pp. 1–7.

[182] Adam R Kosiorek et al. "Sequential Attend, Infer, Repeat: Generative Modelling of Moving Objects". In: *arXiv:1806.01794* (2018).

[183] Ilya Kostrikov. *JAXRL: Implementations of Reinforcement Learning algorithms in JAX*. Oct. 2021. DOI: 10.5281/zenodo.5535154. URL: https://github.com/ikostrikov/jaxrl.

[184] Tejas Kulkarni et al. "Unsupervised Learning of Object Keypoints for Perception and Control". In: *arXiv:1906.11883* (2019).

[185] Tejas D Kulkarni et al. "Deep convolutional inverse graphics network". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2539–2547.

[186] Tejas D Kulkarni et al. "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation". In: *Advances in neural information processing systems*. 2016, pp. 3675–3683.

[187] Aviral Kumar et al. "Implicit Under-Parameterization Inhibits Data-Efficient Deep Reinforcement Learning". In: *arXiv preprint arXiv:2010.14498* (2020).

[188] Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. "Neural random-access machines". In: *arXiv:1511.06392* (2015).

[189] Ivo Kwee, Marcus Hutter, and Jürgen Schmidhuber. "Market-based reinforcement learning in partially observable worlds". In: *International Conference on Artificial Neural Networks*. Springer. 2001, pp. 865–873.

[190] BM Lake and Marco Baroni. "Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks". In: *arXiv:1711.00350* (2017).

[191] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 350.6266 (2015), pp. 1332–1338.

[192] Yann LeCun et al. "A tutorial on energy-based learning". In: *Predicting structured data* 1.0 (2006).

[193] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[194] Alex X Lee et al. "Stochastic adversarial video prediction". In: *arXiv:1804.01523* (2018).

[195] Jan Lemeire. "Conditional independencies under the algorithmic independence of conditionals". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 5252–5271.

[196] Jan Lemeire and Dominik Janzing. "Replacing causal faithfulness with algorithmic independence of conditionals". In: *Minds and Machines* 23.2 (2013), pp. 227–249.

[197] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[198] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436.

[199] Sergey Levine et al. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[200] Simon D Levy and Ross Gayler. "Vector symbolic architectures: A new building material for artificial general intelligence". In: *Conference on Artificial General Intelligence*. 2008.

[201] Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*. Vol. 3. Springer, 2008.

[202] Falk Lieder et al. "Learning to select computations". In: *arXiv:1711.06892* (2017).

[203] Chen-Hsuan Lin and Simon Lucey. "Inverse compositional spatial transformer networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2568–2576.

[204] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[205] Martina Lippi et al. "Latent space roadmap for visual action planning of deformable and rigid object manipulation". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 5619–5626.

[206] Adam Liška, Germán Kruszewski, and Marco Baroni. "Memorize or generalize? searching for a compositional RNN in a haystack". In: *arXiv preprint arXiv:1802.06467* (2018).

[207] Francesco Locatello et al. "Object-Centric Learning with Slot Attention". In: *NeurIPS*. 2020.

[208] Francesco Locatello et al. "Object-centric learning with slot attention". In: *arXiv preprint arXiv:2006.15055* (2020).

[209] David Lopez-Paz et al. "Gradient episodic memory for continual learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6467–6476.

[210] Joao Loula, Marco Baroni, and Brenden M Lake. "Rearranging the Familiar: Testing Compositional Generalization in Recurrent Networks". In: *arXiv preprint* (2018). eprint: arXiv:1807.07545.

[211] Vikash Kumar Mansinghka et al. "Natively probabilistic computation". PhD thesis. Massachusetts Institute of Technology, Department of Brain and Cognitive ..., 2009.

[212] Gary F Marcus. "Rethinking eliminative connectionism". In: *Cognitive psychology* 37.3 (1998), pp. 243–282.

[213] Gary F Marcus. *The algebraic mind: Integrating connectionism and cognitive science*. MIT press, 2018.

[214] Joe Marino, Yisong Yue, and Stephan Mandt. "Iterative amortized inference". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3403–3412.

[215] Joseph Marino, Milan Cvitkovic, and Yisong Yue. "A general method for amortizing variational filtering". In: *arXiv preprint arXiv:1811.05090* (2018), pp. 7857–7868.

[216] Joseph Marino et al. "Iterative amortized policy optimization". In: *arXiv preprint arXiv:2010.10670* (2020).

[217] Massachusetts Institute of Technology. *The goal of 6.031*. [Online; accessed April 11, 2023]. 2022. URL: %5Curl%7Bhttps://web.mit.edu/6.031/www/sp22/classes/01-static-checking/#the_goal_of_6031%7D.

[218] Eric Mazumdar et al. "Policy-gradient algorithms have no guarantees of convergence in continuous action and state multi-agent settings". In: *arXiv preprint arXiv:1907.03712* (2019).

[219] Donella H Meadows. *Thinking in systems: A primer*. chelsea green publishing, 2008.

[220] Christopher Meek. "Strong Completeness and Faithfulness in Bayesian Networks". In: *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*. UAI'95. Montréal, Qué, Canada: Morgan Kaufmann Publishers Inc., 1995, pp. 411–418. ISBN: 1558603859.

[221] Jan-Willem van de Meent et al. "An introduction to probabilistic programming". In: *arXiv preprint arXiv:1809.10756* (2018).

[222] Beren Millidge. *Scaffolded LLMs as Natural Language Computers*. LessWrong. Accessed: 2023-04-19. 2023. URL: https://www.lesswrong.com/posts/43C3igfmMrE9Qoyfe/scaffolded-llms-as-natural-language-computers.

[223] Marvin Minsky. *Society of mind*. Simon and Schuster, 1988.

[224] Nikhil Mishra et al. "A simple neural attentive meta-learner". In: (2018).

[225] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. URL: http://dx.doi.org/10.1038/nature14236.

[226] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[227] Ofir Nachum et al. "Data-Efficient Hierarchical Reinforcement Learning". In: *arXiv preprint* (2018). eprint: arXiv:1805.08296.

[228] Ashvin V Nair et al. "Visual reinforcement learning with imagined goals". In: *Advances in Neural Information Processing Systems*. 2018, pp. 9209–9220.

[229] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[230] Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. "Grounding language for transfer in deep reinforcement learning". In: *Journal of Artificial Intelligence Research* 63 (2018), pp. 849–874.

[231] Radford M Neal and Geoffrey E Hinton. "A view of the EM algorithm that justifies incremental, sparse, and other variants". In: *Learning in graphical models*. Springer, 1998, pp. 355–368.

[232] Alex Nichol, Joshua Achiam, and John Schulman. "On first-order meta-learning algorithms". In: *CoRR, abs/1803.02999* (2018).

[233] Helena E Nusse, James A Yorke, and Eric J Kostelich. "Basins of attraction". In: *Dynamics: Numerical Explorations*. Springer, 1994, pp. 269–314.

[234] Junhyuk Oh et al. "Action-conditional video prediction using deep networks in atari games". In: *Advances in neural information processing systems*. 2015, pp. 2863–2871.

[235] Junhyuk Oh et al. "Control of memory, active perception, and action in minecraft". In: *arXiv:1605.09128* (2016).

[236] Junhyuk Oh et al. "Zero-shot task generalization with multi-task deep reinforcement learning". In: *arXiv:1706.05064* (2017).

[237] Shohei Ohsawa et al. *Neuron as an Agent*. 2018. URL: https://openreview.net/forum?id=BkfEzz-0-.

[238] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[239] OpenAI. *Robogym*. https://github.com/openai/robogym. 2020.

[240] Giambattista Parascandolo et al. "Learning Independent Causal Mechanisms". In: *arXiv:1712.00961* (2017).

[241] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "Understanding the exploding gradient problem". In: *ArXiv* abs/1211.5063 (2012).

[242] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017.

[243] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019), pp. 8026–8037.

[244] Deepak Pathak et al. "Learning to control self-assembling morphologies: a study of generalization via modularity". In: *Advances in Neural Information Processing Systems*. 2019, pp. 2295–2305.

[245] Judea Pearl. "Causal diagrams for empirical research". In: *Biometrika* 82.4 (1995), pp. 669–688.

[246] Judea Pearl. *Causality*. Cambridge university press, 2009.

[247] Xue Bin Peng et al. "MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies". In: *arXiv preprint arXiv:1905.09808* (2019).

[248] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

[249] Fernando Pineda. "Generalization of back propagation to recurrent and higher order neural networks". In: *Neural information processing systems*. 1987, pp. 602–611.

[250] Luis Pineda et al. "MBRL-Lib: A Modular Library for Model-based Reinforcement Learning". In: *Arxiv* (2021). URL: `https://arxiv.org/abs/2104.10159`.

[251] S Pinker. "Baby born talking—describes heaven". In: *The language instinct: How the mind creates language* (1994), pp. 265–301.

[252] Plato. *Republic*.

[253] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[254] Aravind Rajeswaran et al. "Meta-learning with implicit gradients". In: (2019).

[255] Aditya Ramesh et al. "Zero-shot text-to-image generation". In: *arXiv preprint* (2021). eprint: `arXiv:2102.12092`.

[256] Sachin Ravi and Hugo Larochelle. "Optimization as a model for few-shot learning". In: (2016).

[257] Scott Reed and Nando De Freitas. "Neural programmer-interpreters". In: *arXiv preprint* (2015). eprint: `arXiv:1511.06279`.

[258] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic backpropagation and approximate inference in deep generative models". In: *International conference on machine learning*. PMLR. 2014, pp. 1278–1286.

[259] Sebastian Riedel, Matko Bosnjak, and Tim Rocktäschel. "Programming with a differentiable forth interpreter". In: *CoRR, abs/1605.06640* (2016).

[260] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. "Routing Networks: Adaptive Selection of Non-Linear Functions for Multi-Task Learning". In: *International Conference on Learning Representations* (2018).

[261] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. "Routing networks: Adaptive selection of non-linear functions for multi-task learning". In: *arXiv preprint arXiv:1711.01239* (2017).

[262] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. 1961.

[263] Tim Roughgarden. *Twenty lectures on algorithmic game theory*. Cambridge University Press, 2016.

[264] Reuven Y. Rubinstein and Dirk P. Kroese. "The Cross-Entropy Method". In: *Information Science and Statistics*. 2004.

[265] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[266] Stuart Russell and Eric Wefald. "Principles of metareasoning". In: *Artificial intelligence* 49.1-3 (1991), pp. 361–395.

[267] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[268] Kazuyuki Samejima, Kenji Doya, and Mitsuo Kawato. "Inter-module credit assignment in modular reinforcement learning". In: *Neural Networks* 16.7 (2003), pp. 985–994.

[269] S Sanborn et al. "Representational efficiency outweighs action efficiency in human program induction." In: *Proceedings of the 40th annual Cognitive Science Society* (Madison, WI). July 2018.

[270] Alvaro Sanchez-Gonzalez et al. "Graph networks as learnable physics engines for inference and control". In: *arXiv:1806.01242* (2018).

[271] Adam Santoro et al. "A simple neural network module for relational reasoning". In: *arXiv:1706.01427* (2017).

[272] Nikolay Savinov et al. "Step-unrolled denoising autoencoders for text generation". In: *arXiv preprint arXiv:2112.06749* (2021).

[273] Imanol Schlag and Jürgen Schmidhuber. "Gated Fast Weights for On-The-Fly Neural Program Generation". In: *NIPS Metalearning Workshop*. 2017.

[274] Imanol Schlag and Jürgen Schmidhuber. "Learning to Reason with Third Order Tensor Products". In: *Advances in Neural Information Processing Systems*. 2018, pp. 10003–10014.

[275] Jürgen Schmidhuber. "Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook". PhD thesis. Technische Universität München, 1987.

[276] Jürgen Schmidhuber. "Learning to control fast-weight memories: An alternative to dynamic recurrent networks". In: *Neural Computation* 4.1 (1992), pp. 131–139.

[277] Jürgen Schmidhuber. "Self-delimiting neural networks". In: *arXiv:1210.0118* (2012).

[278] Jürgen Schmidhuber. *Towards compositional learning with dynamic neural networks*. Inst. für Informatik, 1990.

[279] Jürgen Schmidhuber. "Ultimate cognition à la Gödel". In: *Cognitive Computation* 1.2 (2009), pp. 177–193.

[280] Jurgen Schmidhuber. "A local learning algorithm for dynamic feedforward and recurrent networks". In: *Connection Science* 1.4 (1989), pp. 403–412.

[281] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[282] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv:1707.06347* (2017).

[283] O. G. Selfridge. "Pandemonium: A Paradigm for Learning". In: *Neurocomputing: Foundations of Research*. Cambridge, MA, USA: MIT Press, 1988, pp. 115–122. ISBN: 0262010976.

[284] Amirreza Shaban et al. "Truncated back-propagation for bilevel optimization". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 1723–1732.

[285] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[286] Herbert A Simon. "The science of design: Creating the artificial". In: *Design Issues* (1988), pp. 67–82.

[287] Herbert A Simon. "The sciences of the artificial". In: *Cambridge, MA* (1969), p. 132.

[288] Christopher Simpkins and Charles Isbell. "Composable modular reinforcement learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4975–4982.

[289] Gautam Singh, Fei Deng, and Sungjin Ahn. "Illiterate DALL-E Learns to Compose". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=h0OYV0We3oh.

[290] Gautam Singh, Fei Deng, and Sungjin Ahn. "Illiterate DALL·E Learns to Compose". In: *arXiv preprint arXiv:2110.11405* (2021).

[291] Gautam Singh, Yi-Fu Wu, and Sungjin Ahn. "Simple unsupervised object-centric learning for complex and naturalistic videos". In: *arXiv preprint arXiv:2205.14065* (2022).

[292] Paul Smolensky. "Tensor product variable binding and the representation of symbolic structures in connectionist systems". In: *Artificial intelligence* 46.1-2 (1990), pp. 159–216.

[293] Jascha Sohl-Dickstein et al. "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 2256–2265.

[294] Ray J Solomonoff. "A formal theory of inductive inference. Part I". In: *Information and control* 7.1 (1964), pp. 1–22.

[295] Ray J Solomonoff. "A preliminary report on a general theory of inductive inference". In: United States Air Force, Office of Scientific Research. 1960.

[296] Alec Solway et al. "Optimal behavioral hierarchy". In: *PLoS computational biology* 10.8 (2014), e1003779.

[297] Yang Song and Stefano Ermon. "Generative modeling by estimating gradients of the data distribution". In: *Advances in Neural Information Processing Systems* 32 (2019).

[298] Elizabeth S Spelke. "Principles of object perception". In: *Cognitive science* 14.1 (1990), pp. 29–56.

[299] Peter Spirtes et al. *Causation, prediction, and search*. MIT press, 2000.

[300] Nathan Sprague and Dana Ballard. "Multiple-goal reinforcement learning with modular sarsa (0)". In: (2003).

[301] Jost Tobias Springenberg et al. "Striving for simplicity: The all convolutional net". In: *arXiv preprint arXiv:1412.6806* (2014).

[302] Aravind Srinivas et al. "Universal Planning Networks". In: *arXiv:1804.00645* (2018).

[303] Rupesh K Srivastava et al. "Compete to compute". In: *Advances in neural information processing systems*. 2013, pp. 2310–2318.

[304] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. "End-to-end memory networks". In: *Advances in neural information processing systems*. 2015, pp. 2440–2448.

[305] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[306] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44.

[307] Richard S Sutton. "Temporal Credit Assignment in Reinforcement Learning." In: (1985).

[308] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2020.

[309] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[310] Jacques Thibodeau. *But is it Really in Rome? An Investigation of the Rome Model*. Accessed: 2023-04-18. 2023. URL: https://www.lesswrong.com/posts/QL7J9wmS6W2fWpofd/but-is-it-really-in-rome-an-investigation-of-the-rome-model.

[311] Valentin Thomas et al. "Independently Controllable Features". In: *arXiv:1708.01289* (2017).

[312] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[313] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[314] Lazar Valkov et al. "HOUDINI: Lifelong Learning as Program Synthesis". In: *Advances in Neural Information Processing Systems*. 2018, pp. 8701–8712.

[315] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[316] Hado Van Hasselt et al. "Deep reinforcement learning and the deadly triad". In: *arXiv preprint arXiv:1812.02648* (2018).

[317] Sjoerd Van Steenkiste et al. "Relational neural expectation maximization: Unsupervised discovery of objects and their interactions". In: *arXiv preprint arXiv:1802.10353* (2018).

[318] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. Vol. 30. 2017, pp. 5998–6008.

[319] Rishi Veerapaneni et al. "Entity abstraction in visual model-based reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2020, pp. 1439–1456.

[320] Alexander Sasha Vezhnevets et al. "Feudal networks for hierarchical reinforcement learning". In: *arXiv:1703.01161* (2017), pp. 3540–3549.

[321] William Vickrey. "Counterspeculation, auctions, and competitive sealed tenders". In: *The Journal of finance* 16.1 (1961), pp. 8–37.

[322] C. Wang, S. Joshi, and R. Khardon. "First Order Decision Diagrams for Relational MDPs". In: *Journal of Artificial Intelligence Research* 31 (Mar. 2008), pp. 431–472. ISSN: 1076-9757. DOI: 10.1613/jair.2489. URL: http://dx.doi.org/10.1613/jair.2489.

[323] Dequan Wang et al. "Deep object centric policies for autonomous driving". In: *arXiv:1811.05432* (2018).

[324] Chihiro Watanabe. "Interpreting layered neural networks via hierarchical modular representation". In: *International Conference on Neural Information Processing*. Springer. 2019, pp. 376–388.

[325] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[326] Nicholas Watters et al. "COBRA: Data-Efficient Model-Based RL through Unsupervised Object Discovery and Curiosity-Driven Exploration". In: *arXiv:1905.09275* (2019).

[327] Nicholas Watters et al. "Spatial broadcast decoder: A simple architecture for learning disentangled representations in VAEs". In: *arXiv preprint arXiv:1901.07017* (2019).

[328] Daniel S Weld. "Solving Relational MDPs with First-Order Machine Learning". en. In: (), p. 8.

[329] William F Whitney et al. "Understanding visual concepts with continuation learning". In: *arXiv:1602.06822* (2016).

[330] Nevan Wichers et al. "Hierarchical long-term video prediction without supervision". In: *arXiv:1806.04768* (2018).

[331] Ronald J Williams and David Zipser. "A learning algorithm for continually running fully recurrent neural networks". In: *Neural computation* 1.2 (1989), pp. 270–280.

[332] David J Willshaw, O Peter Buneman, and Hugh Christopher Longuet-Higgins. "Non-holographic associative memory." In: *Nature* (1969).

[333] C. F. Jeff Wu. "On the Convergence Properties of the EM Algorithm". In: *The Annals of Statistics* 11.1 (1983), pp. 95–103. DOI: 10.1214/aos/1176346060. URL: https://doi.org/10.1214/aos/1176346060.

[334] Jiajun Wu et al. "Learning to see physics via visual de-animation". In: *Advances in Neural Information Processing Systems*. 2017, pp. 153–164.

[335] Jianwen Xie, Zilong Zheng, and Ping Li. "Learning energy-based model with variational auto-encoder as amortized sampler". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 10441–10451.

[336] Jianwen Xie et al. "A Tale of Two Flows: Cooperative Learning of Langevin Flow and Normalizing Flow Toward Energy-Based Model". In: *arXiv preprint arXiv:2205.06924* (2022).

[337] Jianwen Xie et al. "A theory of generative convnet". In: *International Conference on Machine Learning*. PMLR. 2016, pp. 2635–2644.

[338] Jianwen Xie et al. "Cooperative training of descriptor and generator networks". In: *IEEE transactions on pattern analysis and machine intelligence* 42.1 (2018), pp. 27–45.

[339] Danfei Xu et al. "Neural task programming: Learning to generalize across hierarchical tasks". In: *arXiv:1710.01813* (2017).

[340] Zhenjia Xu et al. "Modeling Parts, Structure, and System Dynamics via Predictive Learning". In: (2018).

[341] Zhenjia Xu et al. "Unsupervised Discovery of Parts, Structure, and Dynamics". In: *arXiv:1903.05136* (2019).

[342] Ge Yang et al. "Plan2vec: Unsupervised Representation Learning by Latent Plans". In: *Proceedings of The 2nd Annual Conference on Learning for Dynamics and Control*. Vol. 120. Proceedings of Machine Learning Research. 2020, pp. 1–12.

[343] Shuo Yang et al. "Learning multi-object dense descriptor for autonomous goal-conditioned grasping". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 4109–4116.

[344] Wei Yang et al. "Visual semantic navigation using scene priors". In: *arXiv:1810.06543* (2018).

[345] Tianhe Yu et al. "Gradient surgery for multi-task learning". In: *arXiv preprint arXiv:2001.06782* (2020).

[346] Andrii Zadaianchuk, Georg Martius, and Fanny Yang. "Self-supervised Reinforcement Learning with Independently Controllable Subgoals". In: *Conference on Robot Learning*. PMLR. 2022, pp. 384–394.

[347] Vinicius Zambaldi et al. "Deep reinforcement learning with relational inductive biases". In: (2018).

[348] Wojciech Zaremba and Ilya Sutskever. "Learning to execute". In: *arXiv:1410.4615* (2014).

[349] Wojciech Zaremba et al. "Learning simple algorithms from examples". In: *International Conference on Machine Learning*. 2016, pp. 421–429.

[350] Amy Zhang et al. "Composable Planning with Attributes". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. JMLR.org, 2018, pp. 5837–5846.

[351] Lunjun Zhang, Ge Yang, and Bradly C Stadie. "World Model as a Graph: Learning Latent Landmarks for Planning". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 12611–12620. URL: `https://proceedings.mlr.press/v139/zhang21x.html`.

[352] Marvin Zhang et al. "Solar: Deep structured latent representations for model-based reinforcement learning". In: *arXiv:1808.09105* (2018).

[353] Yan Zhang, Jonathon Hare, and Prügel-Bennett Adam. "Deep Set Prediction Networks". In: *arXiv:1906.06565* (2019).

[354] Yan Zhang et al. "Multiset-Equivariant Set Prediction with Approximate Implicit Differentiation". In: *arXiv preprint arXiv:2111.12193* (2021).

[355] Guangxiang Zhu et al. "Object-Oriented Dynamics Learning through Multi-Level Abstraction". In: *arXiv:1904.07482* (2019).

[356] Daniel Zoran et al. "PARTS: Unsupervised Segmentation With Slots, Attention and Independence Maximization". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10439–10447.

Appendix A

Representing Static Entities

A.1 Future Work

Fig. A.1 and Fig. A.2 both highlight intriguing qualitative differences between vanilla and implicit SLATE and understanding what causes these differences would be valuable for future work. What these figures highlight is that multiple decompositions of a scene into components are possible, which may differ in how closely they reflect human intuition on what constitutes a visual entity. This suggests that the optimization objectives for current object-centric models still underspecify the kinds of decompositions we seek to achieve in our models. The paradigm of decomposing static scenes, as opposed to interactive videos (e.g. [319, Fig. 7]), also contributes to this underspecification.

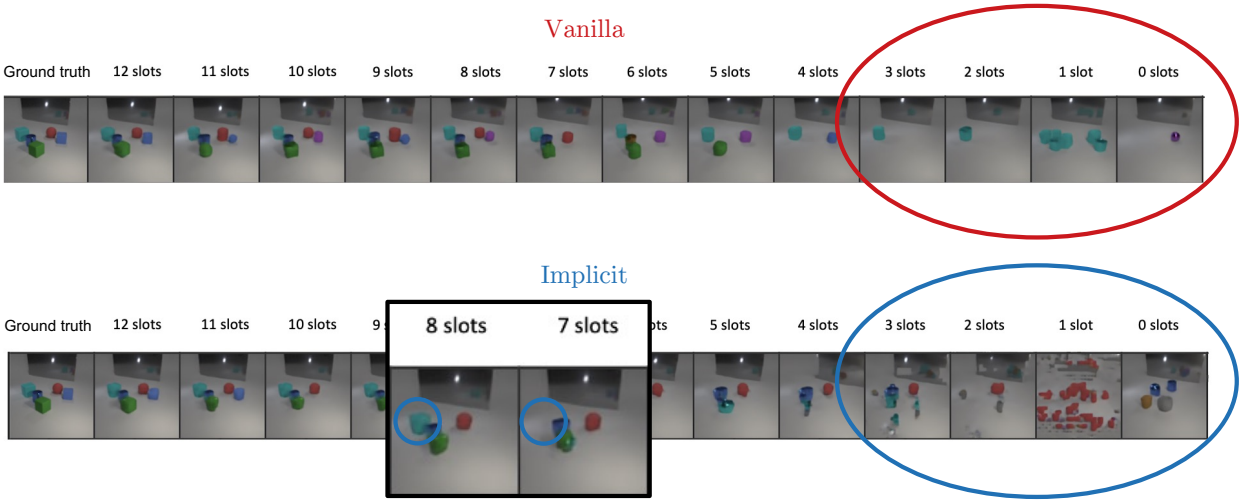


Figure A.1: Implicit differentiation appears to create a stronger dependence among the slots. This figure shows what reconstruction looks if we train and evaluate with 12 slots, then re-render the reconstruction by deleting slots one at a time. When there are still many other slots as context, for both vanilla and implicit SLATE, deleting a slot corresponds to a clean deletion of the corresponding object in the reconstruction, as shown in the inset that highlights what the rendering looks like if we render with eight slots and seven slots. However, as we remove more slots, implicit SLATE generates less coherent compositions than vanilla SLATE, as shown when we render with only one to three slots. What causes this discrepancy is also an open question for future work.

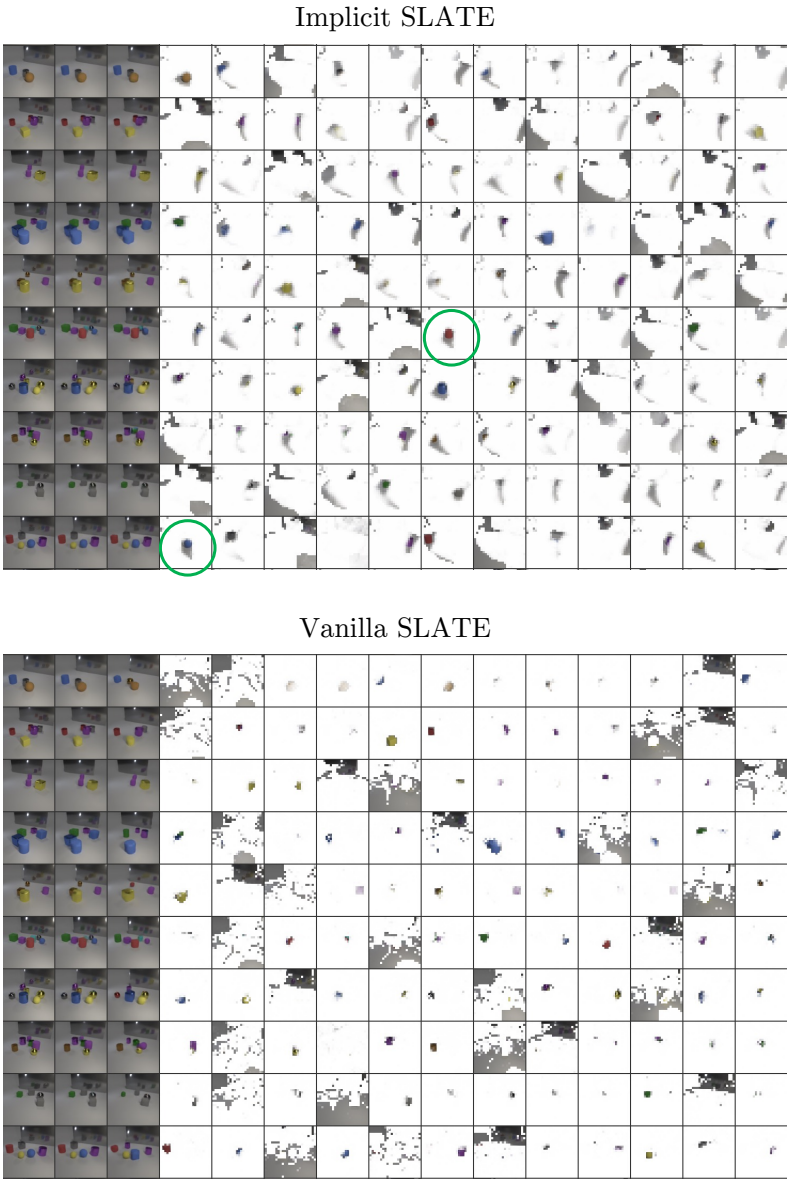


Figure A.2: Despite our work pushing the optimization performance for a state-of-the-art model in object-centric learning (Tab. 2.3), and despite implicit slot attention producing similarly intuitive *predicted* segmentation masks as vanilla slot attention (Fig. 2.8), there appears to be a qualitative difference between the *attention* maps of implicit slate and those of vanilla slate. As this figure shows, the attention masks for vanilla SLATE appear to be more localized to each object, the attention masks for implicit SLATE appear to be more smeared out. One observation is that in some cases implicit SLATE appears to attend not only to the object but also its shadow, as circled in green. However, in other cases the attention maps appear to be smeared in other ways that may attend to a shadow that could possibly happen, but not necessarily a shadow in the given scene. What causes this discrepancy is open question for future work.

A.2 Further Experiments

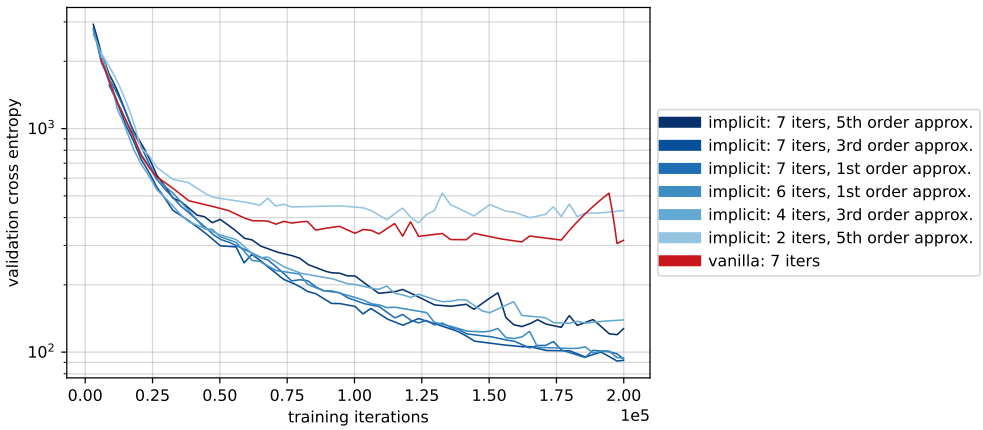


Figure A.3: **Comparing different orders of Neumann approximation.** We sought to understand how the different orders of Neumann approximation affected performance. We observe that the 1st order approximation still largely performs the best, likely because adding more terms to the series expansion requires backpropagating through more iterations of slot attention, which was the problem we had sought to avoid in the first place. However, most approximations still perform better than the vanilla model with the same number of forward iterations.

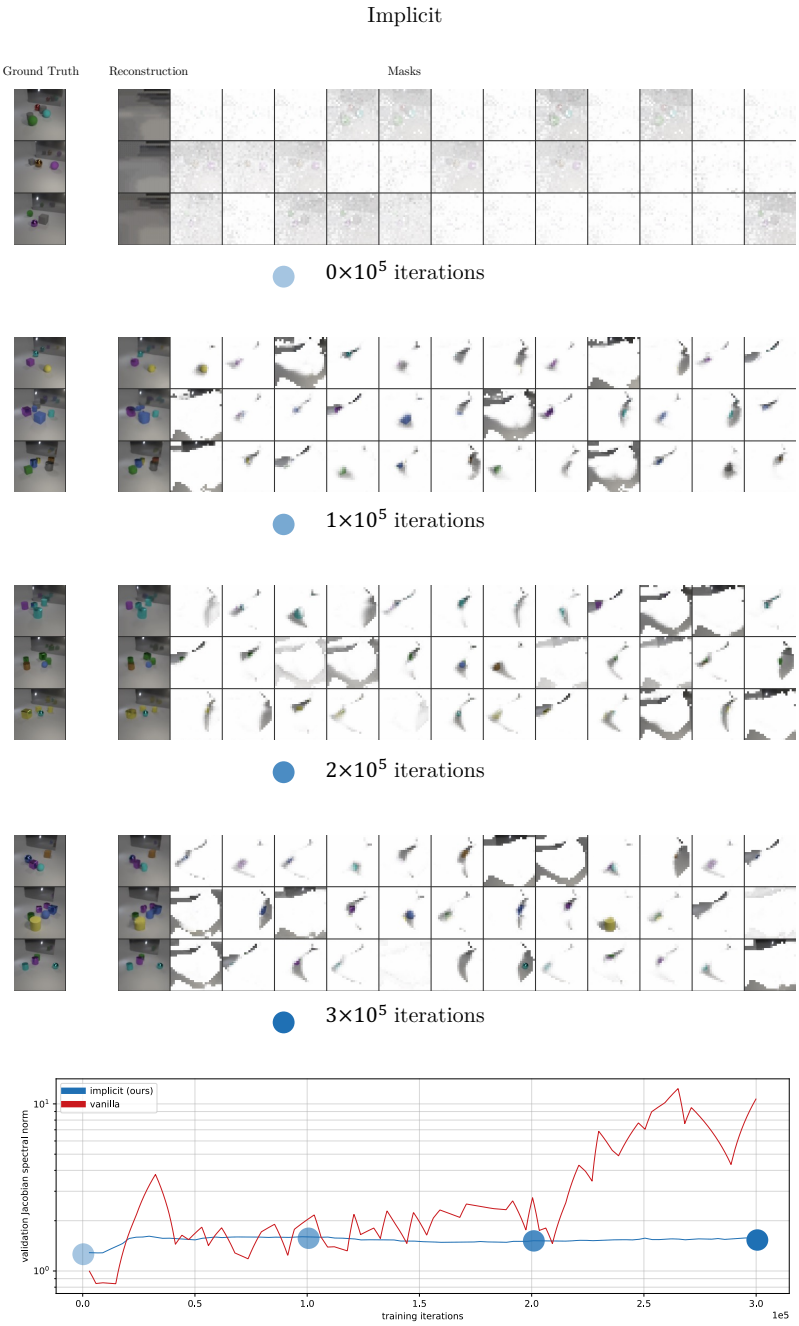


Figure A.4: **Qualitative visualizations without gradient clipping: implicit.** This figure shows qualitative visualizations of implicit SLATE's reconstructions and attention masks when trained without gradient clipping. Compared to Fig. A.5, implicit SLATE's reconstructions matches the ground truth much more closely, and its masks are more coherent. Vanilla SLATE's masks are much noisier, and become degenerate in later stages of training as its Jacobian norm explodes.

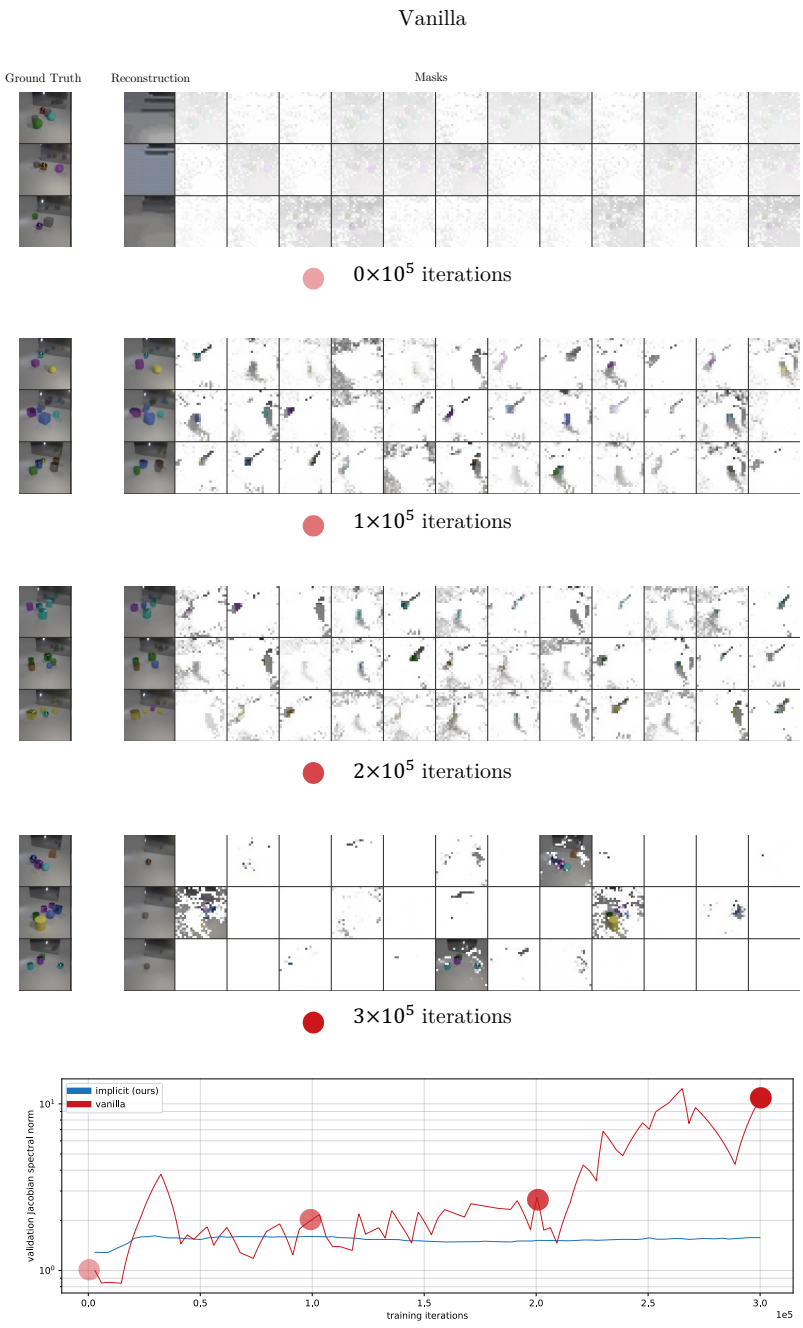


Figure A.5: **Qualitative visualizations without gradient clipping: vanilla.** Compared to Fig. A.4, vanilla SLATE's masks are much noisier, and become degenerate in the later stages of training as its Jacobian norm explodes, whereas implicit SLATE's reconstructions matches the ground truth much more closely, and its masks are more coherent.

Appendix B

Representing Dynamic Entities

B.1 Observation Model

The observation model $\mathcal{G}$ models how the objects $H_{1:K}$ cause the image observation $X \in \mathbb{R}^{N \times M}$. Here we provide a mechanistic justification for our choice of observation model by formulating the observation model as a probabilistic approximation to a deterministic rendering engine.

Deterministic rendering engine: Each object H_k is rendered independently as the sub-image I_k and the resulting K sub-images are combined to form the final image observation X. To combine the sub-images, each pixel $I_{k(ij)}$ in each sub-image is assigned a depth $\delta_{k(ij)}$ that specifies the distance of object k from the camera at coordinate (ij). of the image plane. Thus the pixel $X_{(ij)}$ takes on the value of its corresponding pixel $I_{k(ij)}$ in the sub-image I_k if object k is closest to the camera than the other objects, such that

$$X_{(ij)} = \sum_{k=1}^{K} Z_{k(ij)} \cdot I_{k(ij)}, \tag{B.1}$$

where $Z_{k(ij)}$ is the indicator random variable $\mathbb{1}[k = \text{argmin}_{k \in K}\, \delta_{k(ij)}]$, allowing us to intuitively interpret Z_k as segmentation masks and I_k as color maps.

Modeling uncertainty with the observation model: In reality we do not directly observe the depth values, so we must construct a probabilistic model to model our uncertainty:

$$\mathcal{G}\left(X | H_{1:K}\right) = \prod_{i,j=1}^{N,M} \sum_{k=1}^{K} m_{(ij)}(H_k) \cdot g\left(X_{(ij)} \,|\, H_k\right), \tag{B.2}$$

where every pixel (ij) is modeled through a set of mixture components $g\left(X_{(ij)} \,|\, H_k\right)$ that model how pixels of the individual sub-images I_k are generated, as well as through the mixture weights $m_{ij}(H_k)$ that model which point of each object is closest to the camera. The mixture components are defined as $g\left(X_{(ij)} \,|\, H_k\right) := p\left(X_{ij} | Z_{k(ij)} = 1, H_k\right)$ and the mixture weights are defined as $m_{ij}(H_k) := p\left(Z_{k(ij)} = 1 | H_k\right)$.

B.2 Evidence Lower Bound

Here we provide a derivation of the evidence lower bound. We begin with the log probability of the observations $X^{(1:T)}$ conditioned on a sequence of actions $a^{(0:T-1)}$:

$$\begin{aligned}
\log p\left(X^{(0:T)} \;\middle|\; a^{(0:T-1)}\right) &= \log \int_{h_{1:K}^{(0:T)}} p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right)\, dh_{1:K}^{(0:T)}. \\
&= \log \int_{h_{1:K}^{(0:T)}} p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right) \frac{q\left(h_{1:K}^{(0:T)} \mid \cdot\right)}{q\left(h_{1:K}^{(0:T)} \mid \cdot\right)}\, dh_{1:K}^{(0:T)}. \\
&= \log \mathbb{E}_{h_{1:K}^{(0:T)} \sim q\left(H_{1:K}^{(0:T)} \mid \cdot\right)} \left[\frac{p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right)}{q\left(h_{1:K}^{(0:T)} \mid \cdot\right)}\right] \\
&\geq \mathbb{E}_{h_{1:K}^{(0:T)} \sim q\left(H_{1:K}^{(0:T)} \mid \cdot\right)} \log \left[\frac{p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right)}{q\left(h_{1:K}^{(0:T)} \mid \cdot\right)}\right]. \qquad \text{(B.3)}
\end{aligned}$$

We have freedom to choose the approximating distribution $q\left(H_{1:K}^{(0:T)} \mid \cdot\right)$ so we choose it to be conditioned on the past states and actions, factorized across time:

$$q\left(H_{1:K}^{(0:T)} \mid x^{(0:T)}, a^{(0:T)}\right) = q\left(H_{1:K}^{(0)} \mid x^{(0)}\right) \prod_{t=1}^{T} q\left(H_{1:K}^{(t)} \mid H_{1:K}^{(t-1)}, x^{(t)}, a^{(t-1)}\right)$$

With this factorization, we can use linearity of expectation to decouple Equation B.3 across timesteps:

$$\mathbb{E}_{h_{1:K}^{(0:T)} \sim q\left(H_{1:K}^{(0:T)} \mid x^{(0:T)}, a^{(0:T)}\right)} \log \left[\frac{p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right)}{q\left(h_{1:K}^{(0:T)} \mid x^{(0:T)}, a^{(0:T)}\right)}\right] = \sum_{t=0}^{(t)} \mathcal{L}_{\mathrm{r}}^{(t)} - \mathcal{L}_{\mathrm{c}}^{(t)},$$

where at the first timestep

$$\begin{aligned}
\mathcal{L}_{\mathrm{r}}^{(0)} &= \mathbb{E}_{h_{1:K}^{(0)} \sim q\left(H_{1:K}^{(0)} \mid X^{(0)}\right)} \left[\log p\left(X^{(0)} \mid h_{1:K}^{(0)}\right)\right] \\
\mathcal{L}_{\mathrm{c}}^{(0)} &= D_{KL}\left(q\left(H_{1:K}^{(0)} \mid X^{(0)}\right) || \, p\left(H_{1:K}^{(0)}\right)\right)
\end{aligned}$$

and at subsequent timesteps

$$\begin{aligned}
\mathcal{L}_{\mathrm{r}}^{(t)} &= \mathbb{E}_{h_{1:K}^{(t)} \sim q\left(H_{1:K}^{(t)} \mid h_{1:K}^{(0:t-1)}, X^{(0:t)}, a^{(0:t-1)}\right)} \left[\log p\left(X^{(t)} \mid h_{1:K}^{(t)}\right)\right] \\
\mathcal{L}_{\mathrm{c}}^{(t)} &= \mathbb{E}_{h_{1:K}^{(t-1)} \sim q\left(H_{1:K}^{(t-1)} \mid h_{1:K}^{(0:t-2)}, X^{(1:t-1)}, a^{(0:t-2)}\right)} \left[D_{KL}\left(q\left(H_{1:K}^{(t)} \mid h_{1:K}^{(t-1)}, X^{(t)}, a^{(t-1)}\right) || \, p\left(H_{1:K}^{(t)} \mid h_{1:K}^{(t-1)}, a^{(t-1)}\right)\right)\right].
\end{aligned}$$

By the Markov property, the marginal $q(H_{1:K}^{(t)} \mid h_{1:K}^{(0:t-1)}, X^{(0:t)}, a^{(0:t-1)})$ is computed recursively as

$$\mathbb{E}_{h^{(t-1)} \sim q\left(H_{1:K}^{(t-1)} \mid h_{1:K}^{(0:t-2)}, X^{(0:t-1)}, a^{(0:t-2)}\right)} \left[q\left(H_{1:K}^{(t)} \mid h_{1:K}^{(t-1)}, X^{(t)}, a^{(t-1)}\right)\right]$$

whose base case is $q\left(H^{(0)} \mid X^{(0)}\right)$ when $t = 0$.

We approximate observation distribution $p(X \mid H_{1:K})$ and the dynamics distribution $p(H'_{1:K} \mid H_{1:K}, a)$ by learning the parameters of the observation model $\mathcal{G}$ and dynamics model $\mathcal{D}$ respectively as outputs of neural networks. We approximate the recognition distribution $q(H^{(t)}_{1:K} \mid h^{(t-1)}_{1:K}, X^{(t)}, a^{(t-1)})$ via an inference procedure that refines better estimates of the posterior parameters, computed as an output of a neural network. To compute the expectation in the marginal $q(H^{(t)}_{1:K} \mid h^{(0:t-1)}_{1:K}, X^{(0:t)}, a^{(0:t-1)})$, we follow standard practice in amortized variational inference by approximating the expectation with a single sample of the sequence $h^{(0:t-1)}_{1:K}$ by sequentially sampling the latents for one timestep given latents from the previous timestep, and optimizing the ELBO via stochastic gradient ascent [77, 175, 258].

B.3 Posterior Predictive Distribution

Here we provide a derivation of the posterior predictive distribution for the dynamic latent variable model with multiple latent states. Section B.2 described how we compute the distributions $p(X \mid H_{1:K})$, $p(H'_{1:K} \mid H_{1:K}, a)$, $q(H^{(t)}_{1:K} \mid h^{(t-1)}_{1:K}, X^{(t)}, a^{(t-1)})$, and $q(H^{(0:T)}_{1:K} \mid x^{(1:T)}, a^{(1:T)})$. Here we show that these distributions can be used to approximate the predictive posterior distribution $p(X^{(T+1:T+d)} \mid x^{(0:T)}, a^{(0:T+d)})$ by maximizing the following lower bound:

$$\begin{aligned}
\log p\left(X^{(T+1:T+d)} \mid x^{(0:T)}, a^{(0:T+d)}\right) &= \int_{h^{(0:T+d)}_{1:K}} p\left(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)}\right) \, dh^{(0:T+d)}_{1:K} \\
&= \int_{h^{(0:T+d)}_{1:K}} p\left(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)}\right) \frac{q\left(h^{(0:T+d)}_{1:K} \mid \cdot\right)}{q\left(h^{(0:T+d)}_{1:K} \mid \cdot\right)} \, dh^{(0:T+d)}_{1:K} \\
&= \log \mathbb{E}_{h^{(0:T+d)}_{1:K} \sim q\left(H^{(0:T+d)}_{1:K} \mid \cdot\right)} \left[\frac{p\left(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)}\right)}{q\left(h^{(0:T+d)}_{1:K} \mid \cdot\right)}\right] \\
&\geq \mathbb{E}_{h^{(0:T+d)}_{1:K} \sim q\left(H^{(0:T+d)}_{1:K} \mid \cdot\right)} \log \left[\frac{p\left(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)}\right)}{q\left(h^{(0:T+d)}_{1:K} \mid \cdot\right)}\right].
\end{aligned} \tag{B.4}$$

The numerator $p(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)})$ can be decomposed into two terms, one of which involving the posterior $p(h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)})$:

$$p\left(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)}\right) = p\left(X^{(T+1:T+d)} \mid h^{(0:T+d)}_{1:K}\right) p\left(h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)}\right),$$

This allows Equation B.4 to be broken up into two terms:

$$\mathbb{E}_{h^{(0:T+d)}_{1:K} \sim q\left(H^{(0:T+d)}_{1:K} \mid \cdot\right)} \log p\left(X^{(T+1:T+d)} \mid h^{(0:T+d)}_{1:K}\right) - D_{KL}\left(q\left(H^{(0:T+d)}_{1:K} \mid \cdot\right) || \, p\left(H^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)}\right)\right)$$

Maximizing the second term, the negative KL-divergence between the variational distribution $q(H^{(0:T+d)}_{1:K} \mid \cdot)$ and the posterior $p(H^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)})$ is the same as maximizing the following lower bound:

$$\mathbb{E}_{h^{(0:T)}_{1:K} \sim q\left(h^{(0:T)}_{1:K} \mid \cdot\right)} \log p\left(x^{(0:T)} \mid h^{(0:T)}_{1:K}, a^{(0:T-1)}\right) - D_{KL}\left(q\left(H^{(0:T+d)}_{1:K} \mid \cdot\right) || \, p\left(H^{(0:T+d)}_{1:K} \mid a^{(0:T+d)}\right)\right) \tag{B.5}$$

where the first term is due to the conditional independence between $X^{(0:T)}$ and the future states $H_{1:K}^{(T+1:T+d)}$ and actions $A^{(T+1:T+d)}$. We choose to express $q\left(H_{1:K}^{(0:T+d)} \mid \cdot\right)$ as conditioned on past states and actions, factorized across time:

$$q\left(H^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d-1)}\right) = q\left(H_{1:K}^{(0)} \mid x^{(0)}\right) \prod_{t=1}^{T+d} q\left(H_{1:K}^{(t)} \mid H_{1:K}^{(t-1)}, x^{(t)}, a^{(t-1)}\right).$$

In summary, Equation B.4 can be expressed as

$$\begin{aligned}
&\mathbb{E}_{h_{1:K}^{(0:T+d)} \sim q\left(H^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d-1)}\right)} \log p\left(X^{(T+1:T+d)} \mid h_{1:K}^{(0:T+d)}\right) \\
+&\mathbb{E}_{h_{1:K}^{(0:T)} \sim q\left(H^{(0:T)} \mid x^{(0:T)}, a^{(0:T-1)}\right)} \log p\left(x^{(0:T)} \mid h_{1:K}^{(0:T)}, a^{(0:T-1)}\right) \\
-&D_{KL}\left(q\left(H^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d-1)}\right) || p\left(H_{1:K}^{(0:T+d)} \mid a^{(0:T+d)}\right)\right)
\end{aligned}$$

which can be interpreted as the standard ELBO objective for timesteps $0:T$, plus an addition reconstruction term for timesteps $T+1:T+d$, a reconstruction term for timesteps $0:T$. We can maximize this using the same techniques as maximizing Equation B.3.

Whereas approximating the ELBO in Equation B.4 can be implemented by rolling out OP3 to predict the next observation via teacher forcing [331], approximating the posterior predictive distribution in Equation B.4 can be implemented by rolling out the dynamics model d steps beyond the last observation and using the observation model to predict the future observations.

B.4 Interactive Inference

Algorithms 3 and 4 detail M steps of the interactive inference algorithm at timestep 0 and $t \in [1, T]$ respectively. Algorithm 3 is equivalent to the IODINE algorithm described in [131]. Recalling that $\lambda_{1:K}$ are the parameters for the distribution of the random variables $H_{1:K}$, we consider in this chapter the case where this distribution is an isotropic Gaussian (e.g. $\mathcal{N}(\lambda_k)$ where $\lambda_k = (\mu_k, \sigma_k)$), although OP3 need not be restricted to the Gaussian distribution. The *refinement network* f_q produces the parameters for the distribution $q(H_k^{(t)} \mid h_k^{(t-1)}, x^{(t)}, a^{(t)})$. The *dynamics network* f_ℓ produces the parameters for the distribution $\ell(H_k^{(t)} \mid h_k^{(t-1)}, h_{[\neq k]}^{(t-1)}, a^{(t)})$. To implement q, we repurpose the dynamics model to transform $h_k^{(t-1)}$ into the initial posterior estimate $\lambda_k^{(0)}$ and then use f_q to iteratively update this parameter estimate. β_k indicates the auxiliary inputs into the refinement network used in [131]. We mark the major areas where the algorithm at timestep t differs from the algorithm at timestep 0 in blue.

Algorithm 3 Interactive Inference: Timestep 0

Input: observation $x^{(0)}$
Initialize: parameters $\lambda^{(0,0)}$
for $i = 0$ **to** $M - 1$ **do**
  Sample $h_k^{(0,i)} \sim \mathcal{N}\left(\lambda_k^{(0,i)}\right)$ for each entity k
  Evaluate $\mathcal{L}^{(0,i)} \approx \log \mathcal{G}\left(x^{(0)} \mid h_{1:K}^{(0,i)}\right) - D_{KL}\left(\mathcal{N}\left(\lambda_{1:K}^{(0,i)}\right) || \mathcal{N}(0, I)\right)$
  Calculate $\nabla_{\lambda_k} \mathcal{L}^{(0,i)}$ for each entity k
  Assemble auxiliary inputs β_k for each entity k
  Update $\lambda_k^{(0,i+1)} \leftarrow f_{\text{refine}}\left(x^{(0)}, \nabla_\lambda \mathcal{L}^{(0,i)}, \lambda^{(0,i)}, \beta_k^{(0,i)}\right)$ for each entity k
end for
return $\lambda^{(0,M)}$

Algorithm 4 Interactive Inference: Timestep t

Input: observation $x^{(t)}$, previous action $a^{(t-1)}$, previous entity states $h_{1:K}^{(t-1)}$
Predict $\lambda_k^{(t,0)} \leftarrow f_\partial\left(h_k^{(t-1)}, h_{[\neq k]}^{(t-1)}, a^{(t-1)}\right)$ for each entity k
for $i = 0$ **to** $M - 1$ **do**
  Sample $h_k^{(t,i)} \sim \mathcal{N}\left(\lambda^{(t,i)}\right)$ for each entity k
  Evaluate $\mathcal{L}^{(t,i)} \approx \log \mathcal{G}\left(x^{(t)} \mid h_{1:K}^{(t)}\right) - D_{KL}\left(\mathcal{N}\left(\lambda_{1:K}^{(t,i)}\right) || \mathcal{N}\left(\lambda_{1:K}^{(t,0)}\right)\right)$
  Calculate $\nabla_{\lambda_k} \mathcal{L}^{(t,i)}$ for each entity k
  Assemble auxiliary inputs β_k for each entity k
  Update $\lambda_k^{(t,i+1)} \leftarrow f_q\left(x^{(t)}, \nabla_{\lambda_k} \mathcal{L}^{(t,i)}, \lambda_k^{(t,i)}, \beta_k^{(t,i)}\right)$ for each entity k
end for
return $\lambda^{(t,M)}$

Training: We can train the entire OP3 system end-to-end by backpropagating through the entire inference procedure, using the ELBO at every timestep as a training signal for the parameters of $\mathcal{G}$, $\mathcal{D}$, $\mathcal{Q}$ in a similar manner as [317]. However, the interactive inference algorithm can also be naturally be adapted to predict rollouts by using the dynamics model to propagate the $\lambda_{1:K}$ for multiple steps, rather than just the one step for predicting $\lambda_{1:K}^{(t,0)}$ in line 2 of Algorithm 4. To train OP3 to rollout the dynamics model for longer timescales, we use a curriculum that increases the prediction horizon throughout training.

B.5 Cost Function

Let $\hat{I}(H_k) := m(H_k) \cdot g\left(X \mid H_k\right)$ be a *masked* sub-image (see Appdx: B.1). We decompose the cost of a particular configuration of objects into a distance function between entity states, $c(H_a, H_b)$. For the first environment with single-step planning we use L_2 distance of the corresponding masked subimages: $c(H_a, H_b) = L_2(\hat{I}(H_a), \hat{I}(H_b))$. For the second environment with multi-step planning we a different distance function since the previous one may care

more about if a shape matches than if the color matches. We instead use a form of intersection over union but that counts intersection if the mask aligns and pixel color values are close $c(H_a, H_b) = 1 - \frac{\sum_{i,j} m_{ij}(H_a)>0.01 \text{ and } m_{ij}(H_b)>0.01 \text{ and } L_2(g(H_a)_{(ij)}, g(H_b)_{(ij)})<0.1}{\sum_{i,j} m_{ij}(H_a)>0.01 \text{ or } m_{ij}(H_b)>0.01}$. We found this version to work better since it will not give low cost to moving a wrong color block to the position of a different color goal block.

B.6 Architecture and Hyperparameter Details

We use similar model architectures as in [131] and so have rewritten some details from their appendix here. Differences include the dynamics model, inclusion of actions, and training procedure over sequences of data. Like [145], we define our latent distribution of size R to be divided into a deterministic component of size R_d and stochastic component of size R_s. We found that splitting the latent state into a deterministic and stochastic component (as opposed to having a fully stocahstic representation) was helpful for convergence. We parameterize the distribution of each H_k as a diagonal Gaussian, so the output of the refinement and dynamics networks are the parameteres of a diagonal Gaussian. We parameterize the output of the observation model also as a diagonal Gaussian with means μ and global scale $\sigma = 0.1$. The observation network outputs the μ and mask m_k.

Training: All models are trained with the ADAM optimizer [174] with default parameters and a learning rate of 0.0003. We use gradient clipping as in [241] where if the norm of global gradient exceeds 5.0 then the gradient is scaled down to that norm.

Inputs: For all models, we use the following inputs to the refinement network, where LN means Layernorm and SG means stop gradients. The following image-sized inputs are concatenated and fed to the corresponding convolutional network:

Description	Formula	LN	SG	Ch.
image	X			3
means	μ			3
mask	m_k			1
mask-logits	$\hat{m}_k$			1
mask posterior	$p(m_k \mid X, \mu)$			1
gradient of means	$\nabla_{\mu_k} \mathcal{L}$	✓	✓	3
gradient of mask	$\nabla_{m_k} \mathcal{L}$	✓	✓	1
pixelwise likelihood	$p(X \mid H)$	✓	✓	1
leave-one-out likelih.	$p(X \mid H_{i \neq k})$	✓	✓	1
coordinate channels				2
			total:	17

Observation and Refinement Networks

The posterior parameters $\lambda_{1:K}$ and their gradients are flat vectors, and we concatenate them with the output of the convolutional part of the refinement network and use the result as input to the refinement LSTM:

Description	Formula	LN	SG
gradient of posterior	$\nabla_{\lambda_k}\mathcal{L}$	✓	✓
posterior	λ_k		

All models use the ELU activation function and the convolutional layers use a stride equal to 1 and padding equal to 2 unless otherwise noted. For the table below $R_s = 64$ and $R = 128$.

Observation Model Decoder

Type	Size/*Ch.*	Act. Func.	Comment
Input: H_i	R		
Broadcast	$R+2$		+ coordinates
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*4*	Linear	RGB + Mask

Refinement Network

Type	Size/*Ch.*	Act. Func.	Comment
MLP	128	Linear	
LSTM	128	Tanh	
Concat $[\lambda_i, \nabla_{\lambda_i}]$	$2R_s$		
MLP	128	ELU	
Avg. Pool	R_s		
Conv 3×3	R_s	ELU	
Conv 3×3	*32*	ELU	
Conv 3×3	*32*	ELU	
Inputs	*17*		

Dynamics Model

The dynamics model $\mathcal{D}$ models how each entity H_k is affected by action A and the other entity $H_{[\neq k]}$. It applies the same function $\mathcal{d}(H'_k \mid H_k, H_{[\neq k]}, A)$ to each state, composed of

several functions illustrated and described in Fig. 3.4:

$$\tilde{H}_k = d_o(H_k) \qquad \tilde{A} = d_a(A^t) \qquad \tilde{H}_k^{\text{act}} = d_{ao}(\tilde{H}_k \tilde{A})$$
$$H_k^{\text{interact}} = \sum_{i \neq k}^{K} d_{oo}(\tilde{H_i}^{\text{act}}, \tilde{H_k}^{\text{act}}) \qquad H_k^{'} = d_{\text{comb}}(\tilde{H}_k^{\text{act}}, H_k^{\text{interact}}),$$

where for a given entity k, $d_{ao}(\tilde{H}_k \tilde{A}) := d_{\text{act-eff}}(\tilde{H}_k, \tilde{A}) \cdot d_{\text{act-att}}(\tilde{H}_k, \tilde{A})$ computes how ($d_{\text{act-eff}}$) and to what degree ($d_{\text{act-att}}$) an action affects the entity and $d_{oo}(\tilde{H_i}^{\text{act}}, \tilde{H_k}^{\text{act}}) := d_{\text{obj-eff}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{act}}) \cdot d_{\text{obj-att}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{act}})$ computes how ($f_{\text{obj-eff}}$) and to what degree ($d_{\text{obj-att}}$) other entities affect that entity. $d_{\text{obj-eff}}$ and $d_{\text{obj-att}}$ are shared across all entity pairs. The other functions are shared across all entities. The dynamics network takes in a sampled state and outputs the parameters of the posterior distribution. Similar to [145] the output H'_k is then split into deterministic and stochastic components each of size 64 with separate networks f_{det} and f_{sto}. All functions are parametrized by single layer MLPs.

Dynamics Network

Function	Output	Act. Func.	MLP Size
$d_o(H_k)$	$\tilde{H}_k$	ELU	128
$d_a(A)$	$\tilde{A}$	ELU	32
$d_{\text{act-eff}}(\tilde{H}_k, \tilde{A})$		ELU	128
$d_{\text{act-att}}(\tilde{H}_k, \tilde{A})$		Sigmoid	128
$d_{\text{obj-eff}}(\tilde{H}_i^{\text{act}}, \tilde{H}_j^{\text{act}})$		ELU	256
$d_{\text{obj-att}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{act}})$		Sigmoid	256
$d_{\text{comb}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{interact}})$	H'_k	ELU	256
$f_{\text{det}}(H'_k)$	$H'_{\text{k,det}}$		128
$f_{\text{sto}}(H'_k)$	$H'_{\text{k,sto}}$		128

This architectural choice for the dynamics model is an action-conditioned modification of the interaction function used in Relational Neural Expectation Maximization (RNEM) [317], which is a latent-space attention-based modification of the Neural Physics Engine (NPE) [57], which is one of a broader class of architectures known as graph networks [28].

B.7 Experiment Details

Single-Step Block-Stacking

The training dataset has 60,000 trajectories each containing before and after images of size 64x64 from [164]. Before images are constructed with actions which consist of choosing a shape (cube, rectangle, pyramid), color, and an (x, y, z) position and orientation for the block to be dropped. At each time step, a block is dropped and the simulation runs until the

block settles into a stable position. The model takes in an image containing the block to be dropped and must predict the steady-state effect. Models were trained on scenes with 1 to 5 blocks with $K = 7$ entity variables. The cross entorpy method (CEM) begins from a uniform distribution on the first iteration, uses a population size of 1000 samples per iteration, and uses 10% of the best samples to fit a Gaussian distribution for each successive iteration.

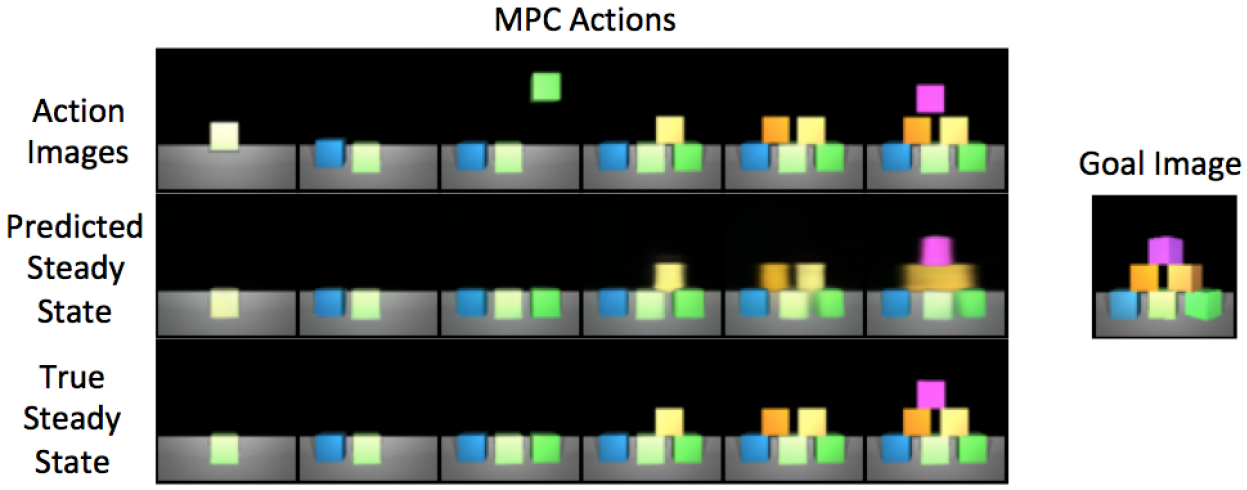


Figure B.1: Qualitative results on building a structure from the dataset in [164]. The input is an "action image," which depicts how an action intervenes on the state by raising a block in the air. OP3 is trained to predict the steady-state outcome of dropping the block. We see how OP3 is able to accurately and consistently predict the steady state effect, successively capturing the effect of inertial dynamics (gravity) and interactions with other objects.

Multi-Step Block-Stacking

The training dataset has 10,000 trajectories each from a separate environment with two different colored blocks. Each trajectory contains five frames (64x64) of randomly picking and placing blocks. We bias the dataset such that 30% of actions will pick up a block and place it somewhere randomly, 40% of actions will pick up a block and place it on top of a another random block, and 30% of actions contain random pick and place locations. Models were trained with K = 4 slots. We optimize actions using CEM but we optimize over multiple consecutive actions into the future executing the sequence with lowest cost. For a goal with n blocks we plan n steps into the future, executing n actions. We repeat this procedure $2n$ times or until the structure is complete. Accuracy is computed as $\frac{\text{\# blocks in correct position}}{\text{\# goal blocks}}$, where a correct position is based on a threshold of the distance error.

For MPC we use two difference action spaces:

Coordinate Pick Place: The normal action space involves choosing a pick (x,y) and place (x,y) location.

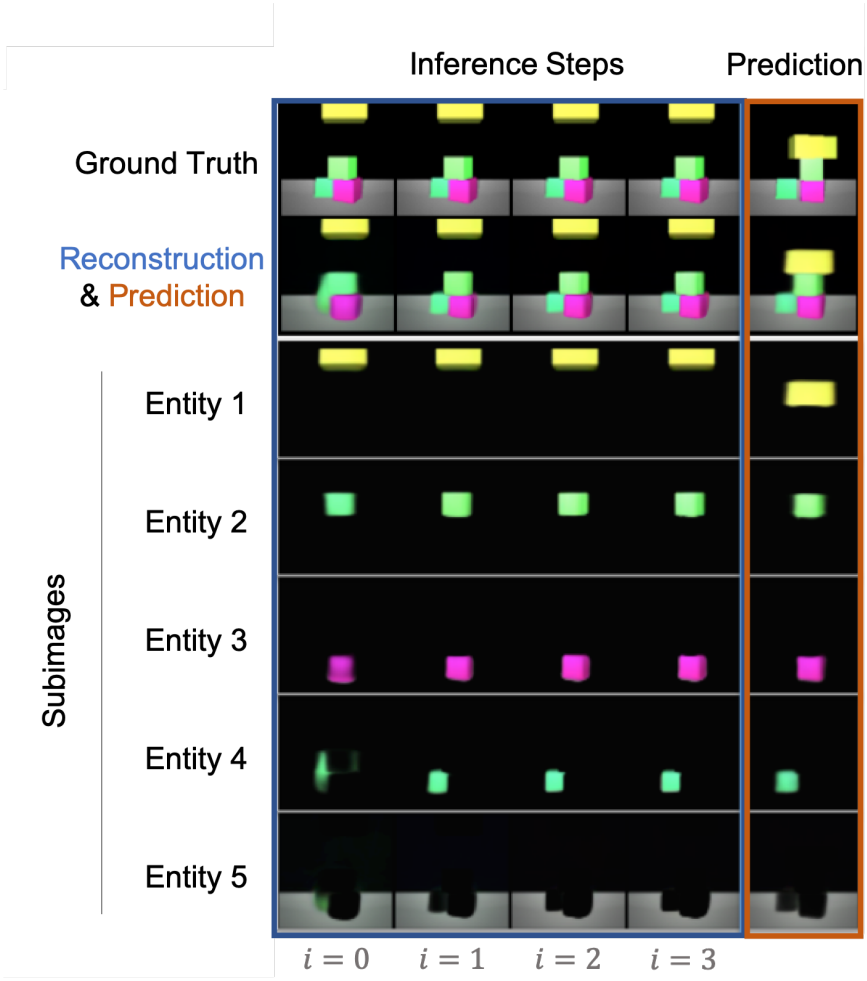


Figure B.2: We show a demonstration of a rollout for the dataset from [164]. The first four columns show inference iterations (refinement steps) on the single input image, while the last column shows the predicted results using the dynamics module on the learnt hidden states. The bottom 5 rows show the subimages of each entity at each iteration, demonstrating how the model is able to capture individual objects, and the dynamics afterwards. Notice that OP3 only predicts a change in the yellow block while leaving the other latents unaffected. This is a desriable property for dynamics models that operate on scenes with multiple objects.

Entity Pick Place: A concern with the normal action space is that successful pick locations are sparse (2%) given the current block size. Therefore, the probability of picking n blocks consecutively becomes 0.02^n which becomes improbable very fast if we just sample pick locations uniformly. We address this by using the pointers to the entity variables to create an action space that involves directly choosing one of the latent entities to move and then a place (x, y) location. This allows us to easily pick blocks consecutively if we can successfully map a latent `entity_id` of a block to a corresponding successful pick location.

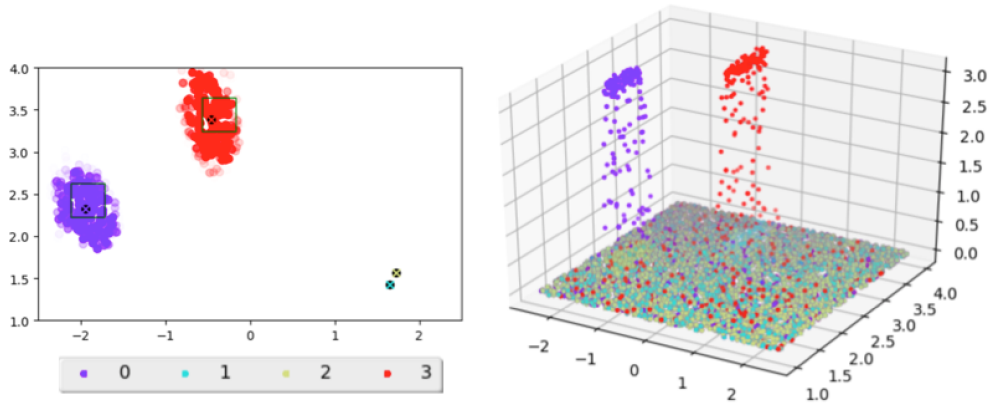


Figure B.3: Two-dimensional (left) and three-dimensional (right) visualization of attention values where colors correspond to different latents. The blocks are shown as the green squares in the 2D visualizatio; picking anywhere within the square automatically picks the block up. The black dots with color crosses denote the computed `pick_xy` for a given h_k. We see that although the individual values are noisy, the means provide good estimates of valid pick locations. In the right plot we see that attention values for all objects are mostly 0, except in the locations corresponding to the objects (purple and red).

In order to determine the pick (x, y) from an `entity_id` k, we sample coordinates uniformly over the pick (x, y) space and then average these coordinates weighted by their attention coefficient on that latent:

$$\texttt{pick\_xy}|h_k = \frac{\sum_{x',y'} p(h_k|x,y) * \texttt{pick\_x'y'}}{\sum_{x',y'} p(h_k|x',y')}$$

where $p(h_k|x, y)$ are given by the attention coefficients produced by the dynamics model given h_k and the pick location (x, y) and x', y' are sampled from a uniform distribution. The attention coefficient of H_k is computed as $\sum_{i \neq k}^{K} d_{\text{obj-att}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{act}})$ (see Appdx. B.6)

B.8 Ablations

We perform ablations on the block stacking task from [164] examining components of our model. Table B.1 shows the effect of non-symmetrical models or cost functions. The "Unfactorized Model" and "No Weight Sharing" follow (c) and (d) from Figure 3.1 and are unable to sufficiently generalize. "Unfactorized Cost" refers to simply taking the mean-squared error of the compositie prediction image and the goal image, rather than decomposing the cost per entity masked subimage. We see that with the same OP3 model trained on the same data, not using an entity-centric factorization of the cost significantly underperforms a cost function that does decompose the cost per entity (c.f. Table 3.1).

No Weight Sharing	Unfactorized Model	Unfactorized Cost
0 %	0 %	5%

Table B.1: Accuracy of ablations. The no weight sharing model did not converge during training.

B.9 Interpretability

We do not explicitly explore interpretability in this work, but we see that an entity-factorized model readily lends itself to be interpretable by construction. The ability to decompose a scene into specific latents, view latents invididually, and explicitly see how these latents interact with each other could lead to significantly more interpretable models than current unfactorized models. Our use of attention values to determine the pick locations of blocks scratches the surface of this potential. Additionally, the ability to construct cost functions based off individual latents allows for more interpretable and customizable cost functions.

Appendix C

Representing Physical Transformations

C.1 Implementation Details

This section details the implementation design decisions for each component of NCS. The hyperparameters of dSLATE are given in Tab. C.1.

Background: SLATE backbone

SLATE [289] is an autoencoder architecture that uses slot attention (SA) [207] as a bottleneck. It preprocesses the image with a discrete variational autoencoder [255] into a grid of image features, encodes these features into a grid of tokens, infers slots from this token grid with SA, which also produces an attention mask over the features each slot attends to. These slots are trained using a transformer decoder [318, 253] to autoregressively reconstruct the tokens using the slots as keys/values.

Constructing nodes by clustering states

We found that we obtained better clusterings when we used the SA attention mask α as the state s for *block-rearrange* and when we used the action-dependent part of the SA slot $\boldsymbol{\lambda}^s$ as the state s for *robogym-rearrange*. We also empirically found that certain choices of distance metric used for K-means `clustering` and `binding` (implemented as nearest-neighbors) depended on which choice of state representation we used, and this is summarized in Table C.2. The K-means implmentation is adapted from `https://github.com/overshiki/kmeans_pytorch`.

When applying the trained dSLATE to the experience buffer to construct the graph we found that increasing the number of SA iterations improved the entity representations, so even though we trained dSLATE with slot attention three iterations, for constructing the graph we used seven iterations. Lastly, we found that the number of clusters used to for K-Means is the most important hyperparameter for creating a graph that reflected the state transitions. We swept over 16 to 50 clusters and report the optimal number of clusters we found in Table C.3.

Number of epochs		200
Episodes per epoch		5K
Episode length		5
Batch size		32
Peak LR		0.0002
LR warmup steps		30000
Dropout		0.1
Discrete VAE	Vocabulary Size	4096
	Temp. Cooldown	1.0 to 0.1
	Temp. Cooldown Steps	30000
	LR (no warmup)	0.0003
	Image Size	64
	Image Tokens	Image Size / 4
transformer decoder	Layers	4
	Heads	4
	Hidden Dim.	192
Slot attention	Slots	5
	Iterations	3
	Slot Heads	1
	Slot Dim. (h)	192
	Type Dim. (λ^z)	96
	State Dim. (λ^s)	96
transformer dynamics	Layers	4
	Heads	4
	Hidden Dim.	96

Table C.1: **Hyperparameters for training dSLATE** These hyperparameters are almost identical to those found in Singh, Deng, and Ahn [289, Fig. 7], but because dSLATE operates on video demonstrations rather than static images, we changed some hyperparameters to save memory cost. We changed the batch size from 50 to 32, the number of transformer layers and heads from 8 to 4, the number of slot attention iterations from 7 to 3 without observing a significant change in performance. Because each video in the experience buffer contains four objects, we used five slots, one more than the number of objects, following the convention used in Van Steenkiste et al. [317] and Veerapaneni et al. [319].

Action selection

To implement `align` we use the `scipy.optimize.linear_sum_assignment` implementation of the Hungarian algorithm, with Euclidean distances between the z^k's as the matching cost.

Given the set of current entities $\mathbf{h}_t$ and goal constraints $\mathbf{h}_g$, `select-constraint` returns the index k of the goal constraint to satisfy next. By NCS' construction, the edge between the

State representation	α	λ^s
`isolate` distance metric	cosine	cosine
`cluster` distance metric	IoU	squared Euclidean
`bind` distance metric	cosine	squared Euclidean

Table C.2: **Hyperparameters for constructing the transition graph with NCS**. This table shows the distance metrics we use for the `isolate`, `cluster`, and `bind` functions described in 4.4. For *block-rearrange* we use the SA attention mask α as the state s, and for *robogym-rearrange* we use the action-dependent part of the SA slot $\boldsymbol{\lambda}^s$ as the state s.

	block-rearrange	*robogym-rearrange*	*block-stacking*
number of clusters	30	45	47

Table C.3: **Number of clusters used for constructing the nodes of the transition graph.**

nodes that h_t^k and h_g^k are bound to is the state transition that would be executed if the action associated to the edge were taken in the environment. If NCS does not find an edge between the two nodes, such as if h_t^k and h_g^k were incorrectly bound to the graph, then NCS simply takes a random action. textttselect-constraint consists of two steps: (1) ranking transitions (2) sampling a transition.

Ranking The goal of the ranking step is to compute a ranking among the indices of $(h_g^1, ..., h_g^K)$ to choose which index k to actually select to affect with an action. Intuitively, we should rank indices k according to how different s_t^k and s_g^k are because a large difference would indicate that the constraint h_g^k is not satisfied, which means we would need to take an action to move the corresponding object represented by h_t^k. We reuse the distance metric $d(\cdot, \cdot)$ used for `isolate` to implement this ranking.

Sampling Given our ranking, the goal of the sampling step is to select a $k \in \{1, ..., K\}$ whose associated entity we will affect with an action. One way to do this is to simply choose k as $k = \text{argmax}_{k' \in \{1,...,\tilde{K}\}} d(s_t^{k'}, s_{t+1}^{k'})$ as in `isolate`, but we empirically found that sampling k from a categorical distribution whose pre-normalized probabilities are given by $d(s_t^{k'}, s_{t+1}^{k'})$ resulted in better task performance so we used this stochastic sampling approach. One explanation for why using the argmax may be worse is that it relies on the distance metric $d(\cdot, \cdot)$, and the state representation s, to be such that the distance metric flawlessly assigns high value to entities k that need to be moved and low value to entities k that do not need to be moved. But because the state space $\mathcal{S}$ is learned through the dSLATE training process without explicit supervision on the geometry of the space, a pair of points that should be

farther apart than another set of points may not be accurately reflected by using a fixed distance metric $d(\cdot, \cdot)$. Future work will investigate imposing explicit supervision on the geometry of $\mathcal{S}$.

C.2 Baseline Implementation Details

Random (Rand) The random policy takes actions using `env.action_space.sample()`.

Behavior cloning (BC) This approach trains a policy to output the actions directly taken in the provided dataset. We use an MSE loss to train the policy to imitate the actions.

Implicit Q-learning (IQL) IQL is a simple, offline RL approach that uses temporal difference (TD) learning with the dataset actions and trains a behavior policy value function. To produce an optimal value function, IQL estimates the maximum of the Q-function using expectile regression with an asymmetric MSE using the following objectives:

$$L_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[L_2^\tau(Q_{\hat{\theta}}(s,a) - V_\psi(s))] \text{ where } L_2^\tau(u) = |\tau - \mathbb{1}(u<0)|u^2 \tag{C.1}$$

$$L_Q(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}[(r(s,a) + \gamma V_\psi(s') - Q_\theta(s,a))^2] \tag{C.2}$$

$$L_\pi(\phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[\exp\left(\beta(Q_{\hat{\theta}}(s,a) - V_\psi(s))\right) \log \pi_\phi(a|s)]. \tag{C.3}$$

The $V(s)$ estimates are used for TD-backups and the optimal policy is extracted with advantage-weighted behavioral cloning.

Model predictive control (MPC) This approach uses model predictive control with the cross entropy method (CEM) to select actions, using the transformer dynamics model of dSLATE to perform rollouts in latent space. This is similar to the approached used in OP3 [319], except that we use more recently proposed architectural components (slot attention [207] instead of IODINE [132], a transformer instead of a graph network [28, 317, 57]) so our MPC results are not directly comparable to that of OP3. We use the same dSLATE checkpoint that was used for NCS.

We implement this MPC baseline using the `mbrl-lib` library [250] with 10 CEM iterations, an elite ratio of 0.05, and a population size of 250 which was the best configuration we found that fit within a wall clock budget of two days for 8 objects and 100 test episodes. We swept over CEM iterations of $[5, 10, 20]$, elite ratio of $[0.05, 0.1, 0.2]$, and population sizes of $[250, 500, 1000]$, and found that the elite ratio was the most important hyperparameter.

The cost function is computed by first aligning the predicted slots $\mathbf{h}_T$ and goal constraints $\mathbf{h}_g$ using the same `align` procedure in Appendx. C.1, and then adding up the squared Euclidean distance between slots as $cost = \sum_k (h_T^k - h_g^k)^2$.

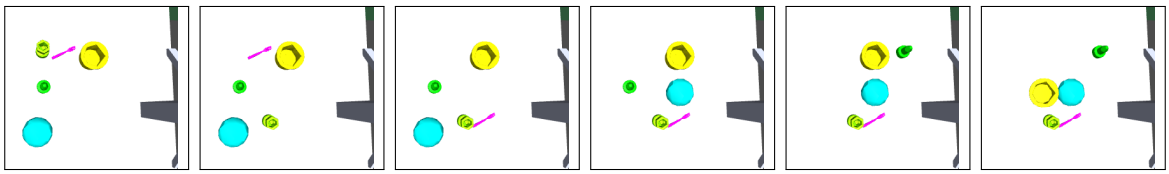


Figure C.1: An example of solving a task in the robogym rearrange environment used in this chapter.

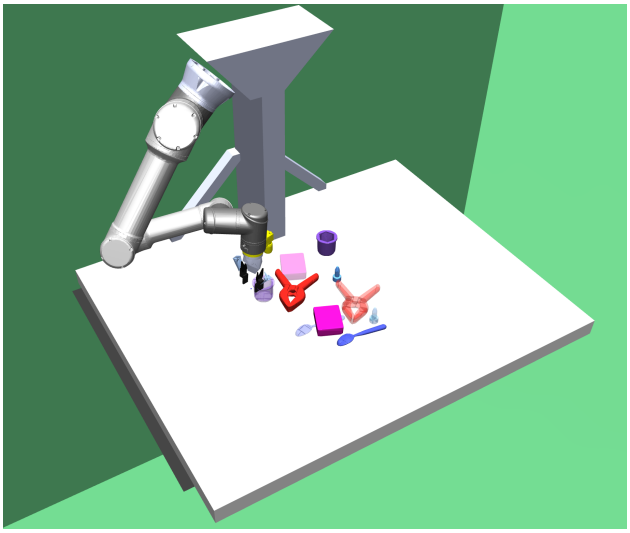


Figure C.2: The original Robogym rearrange setup

Non-factorized graph search (NF) This approach is an ablation to NCS that does not construct a graph over state transitions of individual entities but instead constructs a graph over state transition over entity sets, i.e. each transition is $(\mathbf{s}, a, \mathbf{s}')$ rather than $(s^k, a, s^{k\prime})$. As with MPC, we use the same dSLATE checkpoint that was used for NCS.

The purpose of this ablation is to elucidate the benefit of factorizing the transition graph over *individual entities* rather than *entity sets*. Because nodes in the transition graph for NF represent a set of entity states rather than individual entity states, we use Dijkstra's algorithm, as in [89, 342, 350] to plan a unbroken path from the node the initial observation is bound to to the node a goal observation is bound to. For each time-step, we plan a path along the nodes using Dijkstra's algorithm, then return the action associated with the first edge along that path. Like NCS, NF is a non-parametric model, which means that for a set of entities to be bound to a node in the graph, that node must contain the exact set of entity states corresponding to the states of the entities. If we do not successfully bind to the graph, or if we do not find a path between the current node and the goal node, we sample a random action as NCS does.

C.3 Environment Details

Environments *Block-rearrange* is implemented in PyBullet [67] while *robogym-rearrange* is implemented in Mujoco [313].

Robogym-rearrange (see figures C.1 and C.2) is adapted from the rearrange environment in OpenAI's Robogym simulation framework [239] and removes the assumption from *block-rearrange* that all objects are the same size, shape, and orientation and the assumption of predefined locations. Furthermore, due to 3D perspective, the objects can look slightly different in different locations. Objects are uniformly sampled from a set of 94 meshes consisting of the YCB object set [44] and a set of basic geometric shapes, with colors sampled from a set of 13. The camera angle is a bird's eye view over the table, and the size of each object is normalized by its longest dimension, so tall thin objects appear smaller. The objects' target positions are randomly sampled such that they don't overlap with each other or any of the initial positions, and the target orientation is set to be unchanged. Because locations take continuous values, we define a match threshold of at most 0.05 for both the initial pick position and the goal placement (the table dimensions are 0.6 by 0.8).

Sensorimotor interface Each observation is a tuple of an initial image displaying the current observation and a goal image displaying constraints to be satisfied – the goal locations of the objects. Each action is a tuple $(w, \Delta w)$, where w is a three-dimensional Cartesian coordinate (x, y, z) in the environment arena. Objects are initialized at random non-overlapping locations that also do not overlap with their goal locations. For these tasks the z (height) coordinate is always fixed. An object is picked if w is within a certain threshold of its location. For *block-rearrange* where object locations are fixed points in a grid, the object is snapped to the nearest grid location to $w + \Delta w$. Constraints are considered satisfied if objects are placed within a certain threshold of their target location.

C.4 Additional Results

This section presents additional results and analyses of NCS.

Analysis of key hyperparameters

In this section, we analyze the sensitivity of task performance to several hyperparameters used in NCS when creating the graph: the number of clusters, the number of examples from the experience buffer to use, and the number of slots used in slot attention. We perform this evaluation in the robogym environment with four objects in the complete goal specification. As Fig. C.3 shows, performance depends on the number of initialized clusters and the number of batches from the training set used to construct the graph. With too few clusters, the clusters are too coarse-grained to differentiate objects in significantly different positions. With too many, the performance deteriorates as the data is needlessly split into

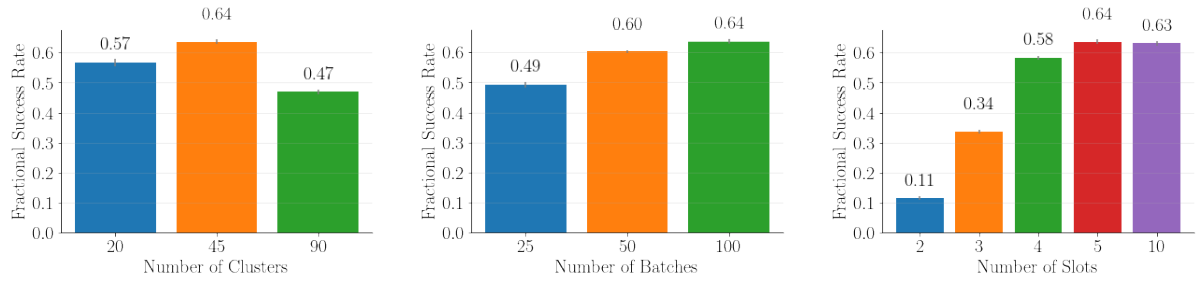


Figure C.3: The performance of our method as the number of initialized clusters and batches from the training set used to construct the graph, and the number of slots are varied.

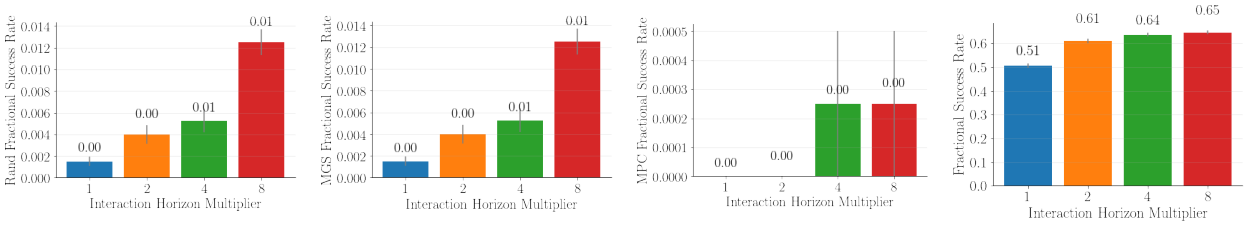
duplicate clusters. Performance improves with more data, as the graph has better coverage. Although NCS performs worse when there are insufficient slots to represent all objects present in the environment, performance is barely impacted by having double the number of necessary slots. Our method can thus still work in environments with an unknown but upper-bounded number of objects.

More computation time for model-based baselines

We tested whether doubling the computation time for the model-based baselines would improve their performance to be comparable to NCS's. For the results Chapter 4, we capped the length of the episode as 4x the minimum number of actions required to solve the task. In Fig. C.4, we vary this interaction horizon multiplier from 1x to 8x. NCS degrades less with shorter interaction horizons compared to the baselines. We find that NF performs similar to the random baseline. Since NF takes a random action if it cannot bind the given entity set to its graph, this result suggests that the space of subsets of entities is so combinatorially large that NF does not successfully bind to the graph most of the time. We verified that this is the case by inspecting when NF takes random actions. MPC performs the worst out of all the methods, performing worse than random. We tested that the cost function described in Appdx. C.2 ranks latents that match the goal constraint with a lower cost than randomly sampled latents, which suggests that the main source of error is due to the inaccuracy in the prediction rollouts. This can be expected, as learned models suffer from compounding errors when rolled out [165] and prior methods that use MPC for object-centric methods only roll out for very short horizons [319].

More challenging settings

Finally, we analyzed NCS in more challenging settings that crudely emulate the noisy nature of real-world robotics. As Fig. C.5 (left) shows, NCS is more robust than the baselines to the addition of Gaussian noise to the action at every time step, up until the noise variance is comparable to the maximum distance for successful picking and goal placements. The performance remains high given significantly fewer interaction steps (Fig. C.5, right).



(a) Rand (b) NF (c) MPC (d) NCS

Figure C.4: **Varying interaction horizon.** The performance of the NF (b) and MPC (c) baselines compared to NCS (d, reproduced from Fig. C.5) and the random baseline (a) on *robogym-rearrange* as we vary the interaction horizon (as a multiple of the minimum steps needed to complete the task). *Note that the scale of the y-axis is not the same.* While a longer horizon improves performance, NCS still achieves at least 50x better accuracy with an interaction horizon multiplier of 1 than the performance obtained by increasing the interaction horizon multiplier for the model-based baselines to 8.

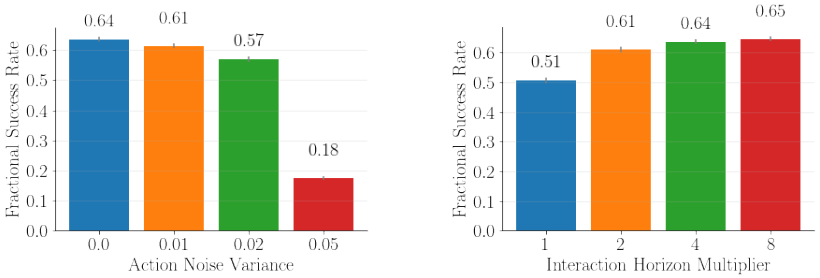


Figure C.5: **Stress testing NCS** This figure shows the performance of NCS on *robogym-rearrange* as we vary the amount of noise added to the actions (left) and vary the interaction horizon, defined as a multiple of the minimum steps needed to complete the task (right).

Nevertheless, our success rate is still nowhere perfect, signifying much more work to do in scaling NCS to the real world.

C.5 Combinatorial Space

This section details the calculation of the combinatorial size of the task space described in § 4.5. The number of object configurations in the initial state is $\binom{|S|}{k}$. In the complete specification setting, all objects must be moved, so $t \geq k$. At each step, any of the k occupied grid cells can be moved to any of the $\binom{|S|}{k}$ unoccupied grid cells, so the number of successor states is $k \times (|S| - k)$. For *block-rearrange* $|S| = 16$ so with $k = 7$ the number of possible trajectories is $\geq 4.5 \times 10^{16}$.

C.6 Limitations and future work.

NCS relies on a nonparametric, non-learning-based approach for control to highlight the generalization capability of our representation of the combinatorial task space, but this limits NCS to only composing previously seen transitions for previously seen entities. Collapsing the combinatorial space along state transitions already provides significant gains but does not adapt to the introduction of novel objects at test time. NCS is currently implemented with tools such as SLATE and K-means that have much potential for improvement. We expect future variations of NCS will improve upon our results by replacing SLATE and K-means with their future successors.

Beyond the challenge of improving object-centric models to robustly model real pixels, extending our method to real world environments, such as those studied in Gokhale et al. [115] and Chang et al. [51] would require overcoming the additional challenge of translating our high-level pick-and-move action primitives into motor torques for a real robot in a way that handles different object geometries, masses, and properties. Given that many works in learning robotics (e.g. Devin et al. [75] and Yang et al. [343]) tackle this exact problem of goal-conditioned object grasping and manipulation, one potential approach to scale our method to real world environments is to train such goal-conditioned policies as the pick-and-move primitives for NCS to compose.

In this chapter, we have assumed objects can be moved independently. Preliminary experiments suggest that NCS can be augmented to support tasks like block-stacking that involve dependencies among objects, but how to handle these dependencies would warrant a standalone treatment in future work.

C.7 Why the name "Neural Constraint Satisfaction?"

NCS can be seen as physically solving a embodied constraint satisfaction probem, where states are variables, identities are variable values, and actions carry out variable assignments. Crucially the variables, their domains, the assignment operator, and the constraints are all learned from the sensorimotor interface, hence the name Neural Constraint Satisfaction.

Appendix D

Representing Virtual Transformations

D.1 Data

Numerical arithmetic (Sec. D.4): The dataset contains arithmetic expressions of k terms where the terms are integers $\in [0, 9]$ and the operators are $\in \{+, \times, -\}$. The number of possible problems is $(10^k)(3^{k-1})$. The learner sees $5810/(2.04 \cdot 10^{14}) = 2.85 \cdot 10^{-11}$ of the training distribution. The number of possible problems in the extrapolation set is $(10^{20})(3^{19}) = 1.16 \cdot 10^{29}$. An input expression is a sequence of one-hot vectors of size 13.

# Terms	Prob. Space	# Train Samples	Frac. of Prob. Space
2	$(10^2)(3^1) = 3 \cdot 10^2$	210	$7 \cdot 10^{-1}$
3	$(10^3)(3^2) = 9 \cdot 10^3$	700	$7.78 \cdot 10^{-2}$
4	$(10^4)(3^3) = 2.7 \cdot 10^5$	700	$2.6 \cdot 10^{-3}$
5	$(10^5)(3^4) = 8.1 \cdot 10^6$	700	$8.64 \cdot 10^{-5}$
6	$(10^6)(3^5) = 2.43 \cdot 10^8$	700	$2.88 \cdot 10^{-6}$
7	$(10^7)(3^6) = 7.29 \cdot 10^9$	700	$9.60 \cdot 10^{-8}$
8	$(10^8)(3^7) = 2.19 \cdot 10^{11}$	700	$3.20 \cdot 10^{-9}$
9	$(10^9)(3^8) = 6.56 \cdot 10^{12}$	700	$1.07 \cdot 10^{-10}$
10	$(10^{10})(3^9) = 1.97 \cdot 10^{14}$	700	$3.56 \cdot 10^{-12}$
Total	$2.04 \cdot 10^{14}$	5810	$2.85 \cdot 10^{-11}$

Table D.1: Numerical Arithmetic Dataset

Multilingual arithmetic (Sec. 5.4): The dataset contains arithmetic expressions of k terms where the terms are integers $\in [0, 9]$ and the operators are $\in \{+, \cdot, -\}$, expressed in five different languages. With 5 choices for the source language and target language, the number of possible problems is $(10^k)(3^{k-1})(5^2)$. In training, each source language is seen with 4 target languages and each target language is seen with 4 source languages: 20 pairs are seen in training and 5 pairs are held out for testing. The learner sees $46200/(1.68 \cdot 10^8) = 2.76 \cdot 10^{-4}$ of the training distribution. The entire space of possible problems in the extrapolation set is $(10^{10})(3^9)(5^2) = 4.92 \cdot 10^{15}$ out of which we draw samples from the 5 held-out language pairs

$((10^{10})(3^9)(5) = 9.84 \cdot 10^{14}$ possible). An input expression is a sequence of one-hot vectors of size $13 \times 5 + 1 = 66$ where the single additional element is a `STOP` token (for training the RNN).

# Terms	Prob. Space	Train Prob. Space	# Train Samples	% Train Dist.	% Prob. Space
2	$10^2 \cdot 3^1 \cdot 25 = 7.5 \cdot 10^3$	$10^2 \cdot 3^1 \cdot 20 = 6 \cdot 10^3$	$210 \cdot 20 = 4.2 \cdot 10^3$	70%	56%
3	$10^3 \cdot 3^2 \cdot 25 = 2.25 \cdot 10^5$	$10^3 \cdot 3^2 \cdot 20 = 1.8 \cdot 10^5$	$700 \cdot 20 = 1.4 \cdot 10^4$	7.78%	6.22%
4	$10^4 \cdot 3^3 \cdot 25 = 6.75 \cdot 10^6$	$10^4 \cdot 3^3 \cdot 20 = 5.4 \cdot 10^6$	$700 \cdot 20 = 1.4 \cdot 10^4$	0.26%	0.207%
5	$10^5 \cdot 3^4 \cdot 25 = 2.02 \cdot 10^8$	$10^5 \cdot 3^4 \cdot 20 = 1.62 \cdot 10^8$	$700 \cdot 20 = 1.4 \cdot 10^4$	0.00864%	0.00691%
Total	$2.09 \cdot 10^8$	$1.68 \cdot 10^8$	46200	0.0276%	0.0221%

Table D.2: Multilingual Arithmetic Dataset

Spatially transformed MNIST (Sec. 5.4): The generative process for transforming the standard MNIST dataset to the input the learner observes is described as follows. We first center the 28x28 MNIST image in a 42x42 black background. We have three types of transformations to apply to the image: scale, rotate, and translate. We can scale big or small (by a factor of 0.6 each way). We can rotate left or right (by 45 degrees each direction). We can translate left, right, up, and down, but the degree to which we translate depends on the size of the object: we translate the digit to the edge of the image, so smaller digits get translated more than large digits. Large digits are translated by 20% of the image width, unscaled digits are translated by 29% of the image width, and small digits are translated by 38% of the image width. In total there are $2 + 2 + 4 \times 3 = 16$ individual transformation operations used in the generative process. Because some transformation combinations are commutative, we defined an ordering with which we will apply the generative transformations: scale then rotate then translate. For length-2 compositions of generative transformations, there are scale-small-then-translate (1×4), scale-big-then-translate (1×4), rotate-then-translate (2×4), and scale-then-rotate (2×2). We randomly choose 16 of these 20 for training, 2 for validation, 2 for test, as shown in Figure 5.4 (center). For length-3 compositions of generative transformations, there are scale-small-then-rotate-then-translate ($1 \times 2 \times 4$) and scale-big-then-rotate-then-translate ($1 \times 2 \times 4$). All 16 were held out for evaluation.

D.2 Learner Details

All learners are implemented in PyTorch [242] and the code is available at `https://github.com/mbchang/crl`.

Arithmetic

Baseline: The RNN is implemented as a sequence-to-sequence [305] gated recurrent unit (GRU) [65].

CRL Controller: The controller consists of a policy network and a value function, each implemented as GRUs that read in the input expression. The value function outputs a value estimate for the current expression. For the numerical arithmetic task, the policy network first selects a reducer and then conditioned on that choice selects the location in the input expression to apply the reducer. For the multilingual arithmetic task, the policy first samples whether to halt, reduce, or translate, and then conditioned on that choice (if it doesn't halt) it samples the reducer (along with an index to apply it) or the translator.

CRL Modules: The reducers are initialized as a two-layer feedforward network with ReLU non-linearities [229]. The translators are a linear weight matrices.

Image Transformations

Baselines: The CNN is a variant of an all-convolutional network [301]. This was also used as the pre-trained image classifier. The affine-STN predicts all 6 learnable affine parameters as in Jaderberg, Simonyan, Zisserman, et al. [161].

CRL Controller: The controller consists of a policy network and a value function, each implemented with the same architecture as the CNN baseline.

CRL Modules: The rotate-STN's localization network is constrained to output the sine and cosine of a rotation angle, the scale-STN's localization network is constrained to output the scaling factor, and the translate-STN's localization network is constrained to output spatial translations

D.3 Experiment Details

Multilingual Arithmetic

Training procedure: The training procedure for the controller follows the standard Proximal Policy Optimization training procedure, where the learner samples a set of episodes, pushes them to a replay buffer, and every k episodes updates the controller based on the episodes collected. Independently, every k' episodes we consolidate those k' episodes into a batch and use it to train the modules. We found via a grid search $k = 1024$ and $k' = 256$. Through an informal search whose heuristic was performance on the training set, we settled on updating the curriculum of CRL every 10^5 episodes and updating the curriculum of the RNN every $5 \cdot 10^4$ episodes.

Domain-specific details: In the case that `HALT` is called to early, CRL treats it as a no-op. Similarly, if a reduction operator is called when there is only one token in the expression, the learner also treats it as a no-op. There are other ways around this domain-specific nuance, such as to always halt whenever `HALT` is called but only do backpropagation from the loss if the expression has been fully reduced (otherwise it wouldn't make sense to compute a loss on an expression that has not been fully reduced). The way we interpret these "invalid actions"

is analogous to a standard practice in reinforcement learning of keeping an agent in the same state if it walks into a wall of a maze.

Symmetry breaking: We believe that the random initialization of the modules and the controller breaks the symmetry between the modules. For episodes 0 through k the controller still has the same random initial weights, and for episodes 0 through k' the modules still have the same random initial weights. Because of the initial randomness, the initial controller will select certain modules more than others for certain inputs; similarly initially certain modules will perform better than others for certain inputs. Therefore, after k episodes, the controller's parameters will update in a direction that will make choosing the modules that luckily performed better for certain inputs more likely; similarly, after k' episodes, the modules' parameters will update in a direction that will make them better for the inputs they have been given. So gradually, modules that initially were slightly better at certain inputs will become more specialized towards those inputs and they will also get selected more for those inputs.

Training objective: The objective of the composition of modules is to minimize the negative log likelihood of the correct answer to the arithmetic problem. The objective of the controller is to maximize reward. It receives a reward of 1 if the token with maximum log likelihood is that of the correct answer, 0 if not, and -0.01 for every computation step it takes. The step penalty was found by a scale search over $\{-1, -0.1, -0.01, -0.001\}$ and -0.01 was a penalty that we found balanced accuracy and computation time to a reasonable degree during training. There is no explicit feedback on what the transformations should be and on how they are composed.

Image Transformations

Training procedure: The training procedure is similar to the mulitlingual arithmetic case. We update the policy every 256 episodes and the modules everye 64 episodes. We observed that directly training for large translations was unstable, so to overcome this we used a curriculum. The curriculum began without any translation, then increased the direction of translation by 1% of the image width every $3 \cdot 10^4$ episodes until the amount of translation matched 20% of the image width for large digits, 29% of the image width for unscaled digits, and 38% of the image width for small digits. Unlike in the multilingual arithmetic case, during later stages of the curriculum we do not continue training on earlier stages of the curriculum.

Domain-specific details: In the bounded-horizon setup, we manually halt CRL according to the length of the generative transformation combinations of the task: if the digit was generated by applying two transformations, then we halt CRL's controller after it selects two modules. Therefore, we did not use a step-penalty in this experiment.

Symmetry breaking: The transformation parameters were initialized to output an identity transformation, although the the localization network were randomly initialized across modules, which breaks the symmetry among the modules.

Training objective: The objective is to classify a transformed MNIST digit correctly based on the negative log likelihood of the correct classification from a pre-trained classifier. The objective of the controller is to maximize reward. It receives a reward of 1 for a correct classification and 0 if not. There is no explicit feedback on what the transformations should be and on how they are composed.

D.4 Additional Experiments

Numerical Math

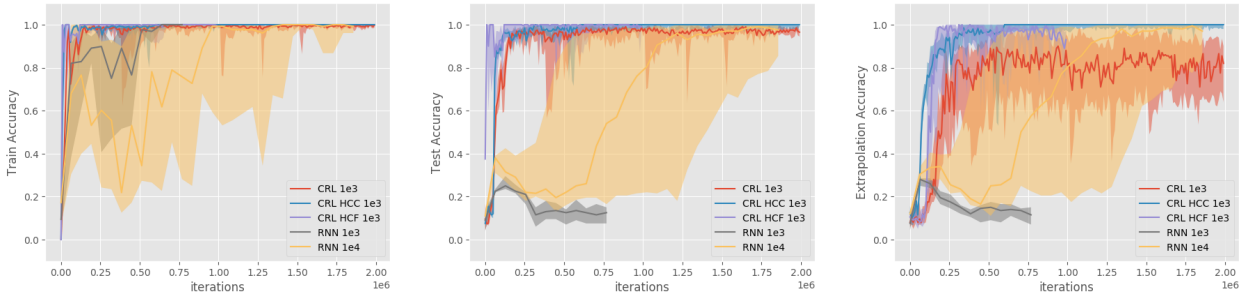
The input is a numerical arithmetic expression (e.g. $3 + 4 \times 7$) and the desired output (e.g. 1) is the evaluation of the expression modulo 10. In our experiments we train on a curriculum of length-2 expressions to length-10 expressions, adding new expressions to an expanding dataset over the course of training. The first challenge is to learn from this limited data (only 6510 training expressions) to generalize well to unseen length-10 expressions in the test set ($\approx 2^{14}$ possible). The second challenge is to extrapolate from this limited data to length-20 expressions ($\approx 10^{29}$ possible). We compare with an RNN architecture [66] directly trained to map input to output.

Though the RNN eventually generalizes to different 10-length expressions and extrapolates to 20-length expressions (yellow in Fig. D.1) with 10 times more data as CRL, it completely overfits when given the same amount of data (gray). In contrast, CRL (red) does not overfit, generalizing significantly better to both the 10-length and 20-length test sets. We believe that the modular disentangled structure in CRL biases it to cleave the problem distribution at its joints, yielding this 10-fold reduction in sample complexity relative to the RNN.

We found that the controller naturally learned windows centered around operators (e.g. $2 + 3$ rather than $\times 4-$), suggesting that it has discovered semantic role of these primitive two-term expressions by pattern-matching common structure across arithmetic expressions of different lengths. Note that CRL's extrapolation accuracy here is not perfect compared to [42]; however CRL achieves such high extrapolation accuracy with only sparse supervision, *without* the step-by-step supervision on execution traces, the stack-based model of execution, and hardcoded transformations.

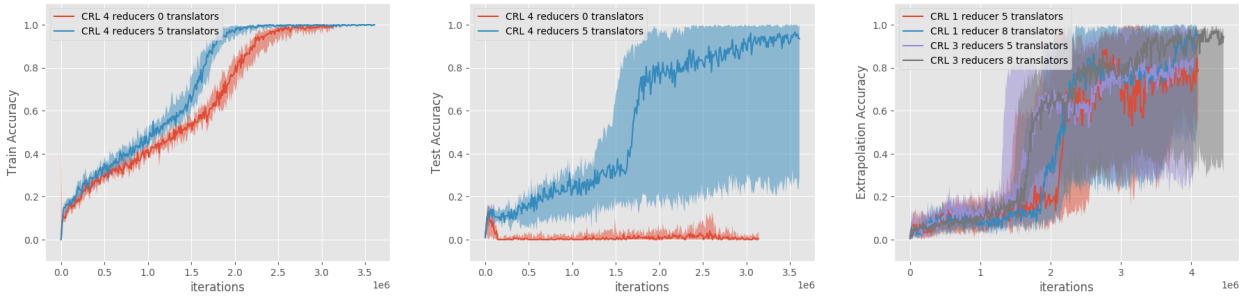
Variations

Here we study the effect of varying the number of modules available to our learner. Figs. D.2a and D.2b highlight a particular pathological choice of modules that causes CRL to overfit. If CRL uses four reducers and zero translators (red), it is not surprising that it fails to generalize to the test set: recall that each source language is only seen with four target languages during training with one held out; each reducer can just learn to reduce to one of the four target languages. What is interesting though is that when we add five translators to the four reducers (blue), we see certain runs achieve 100% generalization, even though CRL need



(a) Training (length 10) (b) Testing (length 10) (c) Testing (length 20)

Figure D.1: **Numerical math task.** We compare our learner with the RNN baseline. As a sanity check, we also compare with a version of our learner which has a hardcoded controller (HCC) and a learner which has hardcoded modules (HCF) (in which case the controller is restricted to select windows of 3 with an operator in the middle). All models perform well on the training set. Only our method and its HCC, HCF modifications generalize to the testing and extrapolation set. The RNN requires 10 times more data to generalize to the testing and extrapolation set. For **(b, c)** we only show accuracy on the expressions with the maximum length of those added so far to the curriculum. "1e3" and "1e4" correspond to the order of magnitude of the number of samples in the dataset, of which 70% are used for training. 10, 50, and 90 percentiles are shown over 6 runs.



(a) Pathological Train (b) Pathological Test (c) Different numbers of modules

Figure D.2: **Variations:** The minimum number of reducers and translators that can solve the multilingual math problems is 1 and m respectively, where m is the number of languages. This is on an extrapolation task, which has more terms *and* different language pairs. (a, b): Four reducers and zero translators (red) is a pathological choice of modules that causes CRL to overfit, but it does not when translators are provided. (c) In the non-pathological cases, regardless of the number of modules, the learner metareasons about the resources it has to customize its computation to the problem. 10, 50, and 90 percentiles are shown over 6 runs.

not use the translators at all in order to fit the training set. That the blue training curve is slightly faster than the red offers a possible explanation: it may be harder to find a program where each reducer can reduce any source language to their specialized target language, and

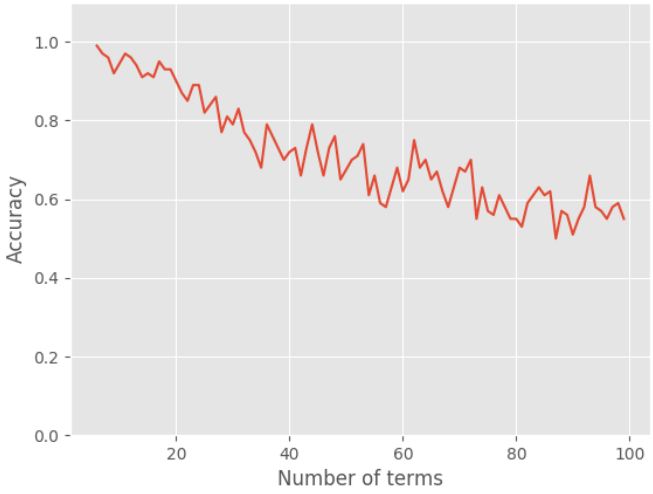


Figure D.3: Extrapolation

easier to find programs that involve steps of re-representation (through these translators), where the solution to a new problem is found merely by re-representing that problem into a problem that learner is more familiar with. The four-reducers-five-translators could have overfitted completely like the four-reducers-zero-translators case, but it consistently does not.

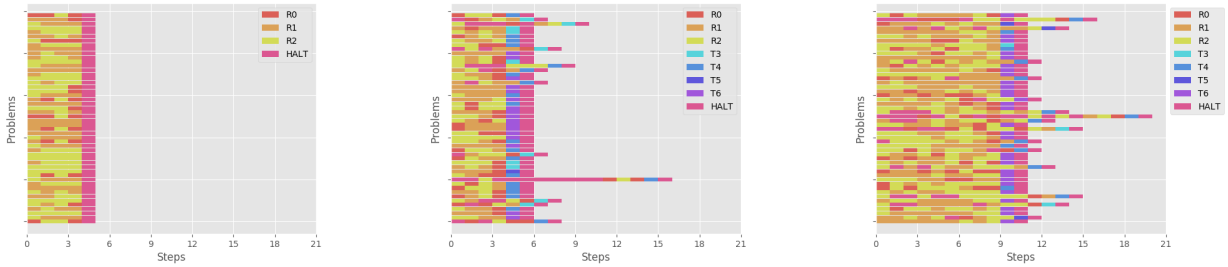
We find that when we vary the number of reducers (1 or 3) and the number of translators in (5 or 8) in Fig. D.2c, the extrapolation performance is consistent across the choices of different numbers of modules, suggesting that CRL is quite robust to the number of modules in non-pathological cases.

How far can we push extrapolation?

Figure D.3 shows the extrapolation accuracy from 6 to 100 terms after training on a curriculum from 2 to 5 terms (46200 examples) on the multilingual arithmetic task (Sec. 5.4). The number of possible 100-term problems is $(10^{100})(3^{99})(5^2) = 4.29 \cdot 10^{148}$ and CRL achieves about 60% accuracy on these problems; a random guess would be 10%.

Execution Traces: Function Selection

Fig. D.4 compares the execution traces of CRL on different language pairs from training of **(a,b)** length 5 and of **(c)** length 10. We observe that in many cases the controller chooses to take an additional step to translate the fully reduced answer into an answer in the target language, which shows that it composes together in a novel way knowledge of how to solve a arithmetic problem with knowledge of how to translate between languages.



(a) Validation (length 5) (b) Test (length 5) (c) Test (length 10)

Figure D.4: **Multilingual Arithmetic Execution Traces**

Execution Traces: Examples

Here are two randomly selected execution traces from the numerical arithmetic extrapolation task (train on 10 terms, extrapolate to 20 terms), where CRL's accuracy hovers around 80%. These expressions are derived from the internal representations of CRL, which are softmax distributions over the vocabulary (except for the first expression, which is one-hot because it is the input). The expressions here show the maximum value for each internal representation.

This is a successful execution, with input `6*1*3-4+6*0*0+1-7-3+3+3*4+1+1+3+3+6+2+7`. The correct answer is `3`. Notice that the order in which controller applies its modules does not strictly follow the order of operations but respects the rules of order of operations: for example, it may decide to perform addition (A) before multiplication (B) if it doesn't affect the final answer.

```
6*1*3-4+6*0*0+1-7-3+3+3*4+1+1+3+3+6+2+7      # 3 * 4 = 2
6*1*3-4+6*0*0+1-7-3+3+2+1+1+3+3+6+2+7        # 3 + 2 = 5
6*1*3-4+6*0*0+1-7-3+5+1+1+3+3+6+2+7          # 1 - 7 = 4
6*1*3-4+6*0*0+4-3+5+1+1+3+3+6+2+7            # 0 * 0 = 0
6*1*3-4+6*0+4-3+5+1+1+3+3+6+2+7              # 4 - 3 = 1
6*1*3-4+6*0+1+5+1+1+3+3+6+2+7                # 1 + 3 = 4
6*1*3-4+6*0+1+5+1+4+3+6+2+7                  # 5 + 1 = 6
6*1*3-4+6*0+1+6+4+3+6+2+7                    # 1 + 6 = 7
6*1*3-4+6*0+7+4+3+6+2+7                      # 2 + 7 = 9
6*1*3-4+6*0+7+4+3+6+9                        # 3 + 6 = 9
6*1*3-4+6*0+7+4+9+9                          # 6 * 0 = 0
6*1*3-4+0+7+4+9+9                            # tried to HALT
6*1*3-4+0+7+4+9+9                            # 9 + 9 = 8
6*1*3-4+0+7+4+8                              # 1 * 3 = 3
6*3-4+0+7+4+8                                # 0 + 7 = 7
6*3-4+7+4+8                                  # 6 * 3 = 8
8-4+7+4+8                                    # 8 - 4 = 4
4+7+4+8                                      # 4 + 7 = 1
1+4+8                                        # 1 + 4 = 5
5+8                                          # 5 + 8 = 3
3                                            # HALT
END
```

```
---------------------------------
above this line is extrapolation
(A)
(B)
```

The following trace with input `5+6-4+5*7*3*3*8*0*1-4+6-3*5*3+6-0+0-4-6` has a correct answer of `0` but was unsuccessful. Notice that it tends to follow of order of operations by doing multiplication first, although it does make mistakes (D), which in this case was the reason for its incorrect answer. Note that CRL never receives explicit feedback about its mistakes on what its modules learn to do or the order in which it applies them; it only receives a sparse reward signal at the very end. Although (C) was a calculation mistake, it turns out that it does not matter because the subexpression would be multiplied by 0 anyways.

```
5+6-4+5*7*3*3*8*0*1-4+6-3*5*3+6-0+0-4-6      # 3 * 8 = 4
5+6-4+5*7*3*4*0*1-4+6-3*5*3+6-0+0-4-6        # 0 - 4 = 6
5+6-4+5*7*3*4*0*1-4+6-3*5*3+6-0+6-6          # 5 * 7 = 5
5+6-4+5*3*4*0*1-4+6-3*5*3+6-0+6-6            # 3 * 4 = 4 (mistake) (C)
5+6-4+5*4*0*1-4+6-3*5*3+6-0+6-6              # tried to HALT
5+6-4+5*4*0*1-4+6-3*5*3+6-0+6-6              # 5 * 4 = 0
5+6-4+0*0*1-4+6-3*5*3+6-0+6-6                # 6 - 6 = 0
5+6-4+0*0*1-4+6-3*5*3+6-0+0                  # 6 - 3 = 3          (D: order of operations mistake)
5+6-4+0*0*1-4+3*5*3+6-0+0                    # tried to HALT
5+6-4+0*0*1-4+3*5*3+6-0+0                    # tried to HALT
5+6-4+0*0*1-4+3*5*3+6-0+0                    # tried to HALT
5+6-4+0*0*1-4+3*5*3+6+0                      # 3 * 5 = 5
5+6-4+0*0*1-4+5*3+6+0                        # 0 * 1 = 0
5+6-4+0*1-4+5*3+6+0                          # 5 * 3 = 5          --------------------------------
5+6-4+0*1-4+5+6+0                            # 0 * 1 = 0          above this line is extrapolation
5+6-4+0-4+5+6+0                              # tried to HALT
5+6-4+0-4+5+6+0                              # tried to HALT
5+6-4+0-4+5+6+0                              # tried to HALT
5+6-4+0-4+5+6+0                              # tried to HALT
5+6-4+0-4+5+6+0                              # tried to HALT
5+6-4+0-4+5+6+0                              # tried to HALT
5+6-4+0-4+5+6+0                              # tried to HALT
5+6-4+0-4+5+6+0                              # 6 + 0 = 0
5+6-4+0-4+5+6                                # 5 + 6 = 1
5+6-4+0-4+1                                  # 0 - 4 = 6
5+6-4+6+1                                    # 5 + 6 = 1
1-4+6+1                                      # 1 - 4 = 7
7+6+1                                        # 7 + 6 = 3
3+1                                          # 3 + 1 = 4
4                                            # HALT
END
```

Appendix E

Representing Policies as Games

E.1 Game Theory

In the context of Chapter 6, a **strategy** of a primitive ω is equivalent to its bidding policy ψ. A **strategy profile** is the set of strategies $\psi^{1:N}$ for all primitives $\omega^{1:N}$. For emphasis, we equivalently write the utility $U^i\left(\psi^{1:N}\right)$ for player i as $U^i\left(\psi^i;\psi^{-i}\right)$, where ψ^i is the strategy for player i and ψ^{-i} is the strategy for all other players.

Definition E.1.1. ***Best Response****: A strategy ψ^i is the best response for player i if given the strategies of all other players ψ^{-i}, ψ^i maximizes player i's utility $U^i\left(\psi^i;\psi^{-i}\right)$.*

Definition E.1.2. ***Nash Equilibrium****: A strategy profile is in a Nash equilibrium if given the strategies of other players, each player's strategy is a best response.*

Definition E.1.3. ***Dominant Strategy Equilibrium****: A strategy ψ is a dominant strategy if it is the best response for a player no matter what strategies the other players play. A dominant strategy equilibrium is the unique Nash equilibrium where every player plays their dominant strategy.*

Definition E.1.4. ***Weakly Dominated Strategies****: Consider player i playing strategy ψ^i. Let the strategies for all other players be ψ^{-i}. A strategy ψ^i* ***weakly dominates*** *$\tilde{\psi}^i$ if for all strategies of other players ψ^{-i}, $U^i(\psi;\psi^{-i}) \geq U^i(\tilde{\psi};\psi^{-i})$, and there exists a $\tilde{\psi}^{-i}$ such that $U^i(\psi^i;\tilde{\psi}^{-i}) > U^i(\tilde{\psi}^i;\tilde{\psi}^{-i})$. ψ is* ***dominated*** *if there exists a strategy which dominates it.*

Definition E.1.5. ***Iterated Deletion of Dominated Strategies****: Iterated deletion of dominated strategies repeatedly (a) removes all dominated strategies for each player, then (b) updates the dominance relation (as now there are fewer strategies of other players to consider). It terminates when all remaining strategies are undominated.*

Vickrey Auction The Vickrey Auction is a type of single-item sealed bid auction. It is single-item, which means there is a single auction item up for sale. It is sealed-bid, which means that the players have no knowledge of each others' bids.

E.2 Societal Decision-Making Framework

Abstraction Level	Global	Local
Agent	Society $\Omega = \{\omega^i\}_{i=1}^N$	Primitive $\omega^i = \left(\psi^i, \phi^i\right)$
Environment	global MDP	local auction at state s
State Space	$\mathcal{S}$	the single state $\{s\}$
Action Space	the indices of the primitives $\mathcal{A} = \{1, 2, ..., N\}$	the space of non-negative bids $\mathcal{B} = \mathbb{R}_{\geq 0}$
Objective	$J(\pi_\Omega) = \mathbb{E}_{\tau \sim p^\pi(\tau)} \left[\sum_{t=0}^T \gamma^t r\left(s_t, \omega_t\right)\right]$	$U_s^i(\psi^{1:N}) = \mathbf{v}_s^i \cdot X^i(\mathbf{b}) - P^i(\mathbf{b})$
Problem	$\max_{\pi_\Omega} J(\pi_\Omega)$	$\max_{\psi^i} U_s^i(\psi^{1:N})$

Table E.1: **Societal Decision-Making.** This table specifies the **agent**, **environment**, **objective**, and **problem** at both the **global** and **local** levels of abstraction in the societal-decision-making framework.

Though both the monolithic decision-making framework as well as the societal decision-making (in which the transformations ϕ correspond to literal actions) can both be used for reinforcement learning, the key difference between the two is that the learnable parameters are trained to optimize the objective of the MDP in the monolithic framework, whereas the learnable parameters are trained to optimize the objective of the auction at each state of the MDP in the societal framework.

E.3 The Cloned Vickrey Society as a Solution to Societal Decision-Making

The MDP specifies the environment. The society specifies the abstract agent that interacts in the environment. The **Market MDP** is a global MDP governed by a specific type of auction mechanism (the Vickrey mechanism) that is agnostic to the architecture of the society that interacts with it. A **cloned society** is a specific architecture of society (one with redundant primitives) that is agnostic to the auction mechanism governing the global MDP. The **cloned Vickrey society** specifies a specific architecture of a society (a cloned society) that interacts with a specific type of global MDP (a Market MDP).

Proofs

Proposition 6.5.1. *Assume at each state s the local auction allocates $X^i(\mathbf{b}) = 1$ if i wins and $X^i(\mathbf{b}) = 0$ if i loses. Then all primitives ω^i bidding their optimal societal Q-values $Q_\Omega^*(s, \omega^i)$ collectively induce an optimal global policy.*

Proof. For state s, the index of the primitive with the highest bid is equal to the primitive with the highest optimal societal Q-value for that state. That is, $\text{argmax}_i \, \mathbf{b}_t^i = \text{argmax}_i \, Q_\Omega^*(s_t, \omega_t^i)$. Thus, selecting the highest-bidding primitive to win by definition follows the optimal policy for the Market MDP. □

Theorem 6.5.2 *If the valuations $\mathbf{v}_s^i$ for each state s are the optimal societal Q-values $Q_\Omega^*(s, \omega^i)$, then the society's optimal global policy coincides with the primitives' unique dominant strategy equilibrium under the Vickrey mechanism.*

Proof. By defining the valuation of each primitive ω^i to be its optimal societal Q-value $Q_\Omega^*(s_t, \omega_t^i)$, the DSIC property of the Vickrey auction guarantees the dominant strategy equilibrium is to bid exactly $Q_\Omega^*(s_t, \omega_t^i)$. The welfare-maximization property of the Vickrey auction guarantees if all primitives played their dominant strategies, then the primitive with the highest valuation wins, so specifying the valuations such that activating the primitive with the highest valuation at time t follows the optimal global policy at that timestep. □

Proposition 6.5.3. *In a Market MDP, it is a Nash equilibrium for every primitive to bid $Q_\Omega^*(s, \omega^i)$. Moreover, if the Market MDP is finite horizon, then bidding $Q_\Omega^*(s, \omega^i)$ is the unique Nash equilibrium that survives iterated deletion of weakly dominated strategies.*

Proof. Consider time-step t. Assuming that every other primitive at every other time-step bids Q_Ω^*, the best response under the Vickrey mechanism for primitive ω^i at timestep t would be to truthfully bid $r(s_t, \omega_t^i) + \gamma \cdot \max_k \, Q_\Omega^*(s_{t+1}, \omega^k)$, which by definition of the Bellman optimality equations [30] is $Q_\Omega^*(s_t, \omega_t^i)$.

For the finite horizon case, we proceed by backwards induction.

Base Case: The last time-step T is a bandit problem where $Q_\Omega^*(s_T, \omega_T^i) = r(s_T, \omega_T^i)$, so by Theorem 6.5.2 bidding $Q_\Omega^*(s_t, \omega_t^i)$ is the unique dominant strategy.

Inductive Hypothesis: In time-step $t+1$, the strategy which survives iterated deletion of dominated strategies is to bid the optimal societal Q-value.

Inductive Step: By the Inductive Hypothesis, in time-step $t+1$, all primitives bid their optimal societal Q-values if they use any strategy which survives iterated deletion of dominated strategies. This means that in time-step t, each primitive's valuation for winning is given exactly by their optimal societal Q-value: $r(s_t, \omega_t^i) + \gamma \cdot \max_k \, Q_\Omega^*(s_{t+1}, \omega^k)$. Therefore, it is a dominant strategy to bid Q_Ω^*, and all other bids are dominated (and therefore removed). □

Lemma 6.5.4 *For a cloned society, at the Nash equilibrium specified in Proposition 6.5.3, what the winning primitive $\hat{\omega}^i$ at time t receives from the winning primitive $\hat{\omega}^k$ at $t+1$ is exactly what $\hat{\omega}^k$ pays: $\mathbf{b}_{s_{t+1}}^k$.*

Proof. If two primitives have the same ϕ, then their societal Q-values are identical, so their optimal strategies ψ are identical. Then at the Nash equilibrium specified in Proposition 6.5.3, their bids are also identical. □

Theorem 6.5.5. *Define a **cloned Vickrey society** as a cloned society that solves a Market MDP. Then it is a Nash equilibrium for every primitive in the cloned Vickrey society to bid $Q^*_\Omega(s, \omega^i)$. In addition, the price that the winning primitive pays for winning is equivalent to what it bid.*

Proof. (1) is by Proposition 6.5.3 and (2) is by Lemma 6.5.4. □

E.4 Decentralized RL Algorithms for the Cloned Vickrey Society

Section 6.5 presented the cloned Vickrey society as a concrete instantiation of the societal decision-making framework whose optimal global policy emerges as a Nash equilibrium of self-interested primitives engaging in local economic transactions. Section 6.6 removed the assumption that primitives know their valuations and presented decentralized RL algorithms for learning these valuations through interaction.

In Chapter 6, we specified the class of decentralized RL algorithms by the learning objective – the set of local auction utilities at every state. We present in Algorithm 5 the algorithm pseudocode for an on-policy variant of such a decentralized RL algorithm, but the learning objective is in principle agnostic to the RL algorithm use to train each primitive. However, simply specifying only one variant within this class of algorithm in Chapter 6 leaves much to still be explored. Adapting methods developed in the monolithic framework, including bandit algorithms, off-policy algorithms, and on-policy algorithms, for optimizing the local auction utilities the would be an interesting direction for future work.

E.5 Implementations of an On-Policy Decentralized RL Algorithm

Stochastic policy gradient In Chapter 6, we considered optimizing the bidding policies with a stochastic policy gradient algorithm, which means that the bidding policies parameterize a distribution of bids. The stochasticity of the bidding policy means that there need not be an explicit exploration strategy such as ϵ-greedy, which simplifies the analysis in Chapter 6. In future work it would be interesting to explore deterministic bidding policies as well.

Bidding Policies We implement all bidding policies as neural networks that output the parameters of a Beta-distribution. Because the Beta-distribution has support between 0 and 1, we normalized environment rewards so returns fit in this range. In theory, the range of possible bids could be $[0, \infty)$ and need not be restricted to $[0, 1]$. The society decision-making framework prescribes a different unique local auction, with different primitives, at each state s. However, because the state space could be extremely large or even continuous, in practice

Algorithm 5 On-Policy Decentralized RL

Initialize: Primitives $\omega^{1:N}$, Memory $m^{1:N}$, RL update rule f
while True **do**
  ▷ *Sample Episode*
  while episode has not terminated **do**
    $\omega^{1:N}$ observe state s
    $\omega^{1:N}$ produce bids $\mathbf{b}_s^1, ..., \mathbf{b}_s^N$
    Auction selects winner $\hat{\omega}$ with transformation $\hat{\phi}_{\mathcal{T}}$
    $\hat{\omega}$ produces $s' = \hat{\phi}_{\mathcal{T}}(s)$
    Record environment reward $r(s, \hat{\omega})$
    $s \leftarrow s'$
  end while
  ▷ *Compute Utilities*
  for each time-step t until the end of sampled episode **do**
    $\hat{\omega}_t$ gets $\hat{U}^i_{s_t}(\omega^{1:N}) = r(s_t, \hat{\omega}_t) + \gamma \cdot \max_k \mathbf{b}^k_{s_{t+1}} - \max_{j \neq i} \mathbf{b}^j_{s_t}$
    Losers ω^j_t get $U^j_{s_t}(\omega^{1:N}) = 0$
    for all primitives ω^i **do**
      Primitive ω^i_t stores $(s_t, \mathbf{b}^i_{s_t}, s_{t+1}, U^i_{s_t}(\omega^i_t))$ into m^i
    end for
  end for
  ▷ *Update*
  if time to update **then**
    for all primitives ω^i **do**
      Update primitive ω^i with update rule f with memory m^i
    end for
  end if
end while

we share the same set of primitives $\omega^{1:N}$ across all states, express their bidding policies as functions of the state, and rely on the function approximation capabilities of neural networks to learn different state-conditioned bidding strategies.

Redundancy Implementing two clones of each primitive means that each clone should share the same transformation ϕ: if ϕ were the literal action of `go-left`, then there would be two primitives that bid in the local auction to execute the `go-left` action. We implemented redundant clones by sharing the weights of their bidding policies ψ. Therefore, cloned primitives have identical bidding distributions, from which different bids are sampled. Alternatively, we also explored giving clone primitives the same transformation ϕ but independently parameterized bidding policies ψ. While we did not find much difference in global performance with independently parameterized bidding policies, such a scheme could be useful for multi-task learning where the same transformation could have different optimal societal Q-values depending on the task.

Learning Objectives The utility of the cloned Vickrey society is the learning objective. We compared three possible implementations of this learning objective: bucket-brigade (BB), Vickrey (V), and credit-conserving Vickrey (CCV) as described in Section 6.7 and Figure 6.4. We also compared with a baseline that sets the learning objective to be the environment reward. For all implementations and the baseline, the learning objective that a loser ω that the auction receives is 0 because its allocation $X^i(\mathbf{b})$ and payment $P^i(\mathbf{b})$ are both 0. Letting $\hat{\mathbf{b}}_{s_t}$ and $\mathbf{b}'_{s_t}$ denote the highest and second highest bid at time t respectively, the learning objective $\hat{U}^i_{s_t}(\omega^{1:N})$ of the winner $\hat{\omega}$ for state s_t is given below.

	Learning Objective
Bucket-Brigade	$\hat{U}^i_{s_t}(\omega^{1:N}) = \left[r(s_t, \hat{\omega}_t) + \gamma \cdot \hat{\mathbf{b}}_{s_{t+1}}\right] - \hat{\mathbf{b}}_{s_t}$
Vickrey	$\hat{U}^i_{s_t}(\omega^{1:N}) = \left[r(s_t, \hat{\omega}_t) + \gamma \cdot \hat{\mathbf{b}}_{s_{t+1}}\right] - \mathbf{b}'_{s_t}$
Credit-Conserving Vickrey	$\hat{U}^i_{s_t}(\omega^{1:N}) = \left[r(s_t, \hat{\omega}_t) + \gamma \cdot \mathbf{b}'_{s_{t+1}}\right] - \mathbf{b}'_{s_t}$
Environment Reward	$\hat{U}^i_{s_t}(\omega^{1:N}) = [r(s_t, \hat{\omega}_t)]$

E.6 Experimental Details

We implemented our experiments using the PyTorch library [243]. For all experiments, for proximal policy optimization we used a policy learning rate of $4 \cdot 10^{-5}$, a value function learning rate of $5 \cdot 10^{-3}$, a clipping ratio of 0.2, a GAE [281] smoothing parameter of 0.95, a discount factor of 0.99, and the Adam [174] optimizer. Each bidding policy has its own replay buffer. Each bidding policy is updated every 4096 transitions with a minibatch size of 256. For *Two Rooms* and *Mental Rotation* we added an entropy bonus, with a weighting coefficient of 0.1, to the PPO loss.

The plots for *Market Bandit*, *Two Rooms*, and *Mental Rotation* were averaged over 3 seeds and the other plots were averaged over 5 seeds. The metric for the learning curves is the mean return over 4096 environment steps. The error bars represent the 10th, 50th, 90th quantile. All experiments were run on CPU, except for the *TwoRooms* environment, which was run on GPU.

Numerical Simulations

All transformations ϕ correspond to literal actions. The bidding policies ψ are implemented as linear neural networks with a single hidden layer of 16 units that output the α and β parameters of the Beta distribution. There is no activation function for the hidden layer. We

used the softplus activation function to output α and β. The valuation functions for the bidding policies are also implemented as linear neural networks with a single hidden layer of 16 units.

Market Bandit

At every round, we stochastically drop out a subset of primitives. For cloned societies, one clone could be dropped out while the other clone stays in the auction. The drop-out sampling procedure is as follows. Letting N be the number of total primitives (which means there are $N/2$ unique primitives in cloned societies), we first sample a random integer m in $\{2, 3, ..., N\}$. Then we sample a subset of m primitives from the N total primitives without replacement. For a given round, only the primitives that participated in that round are updated.

Duality

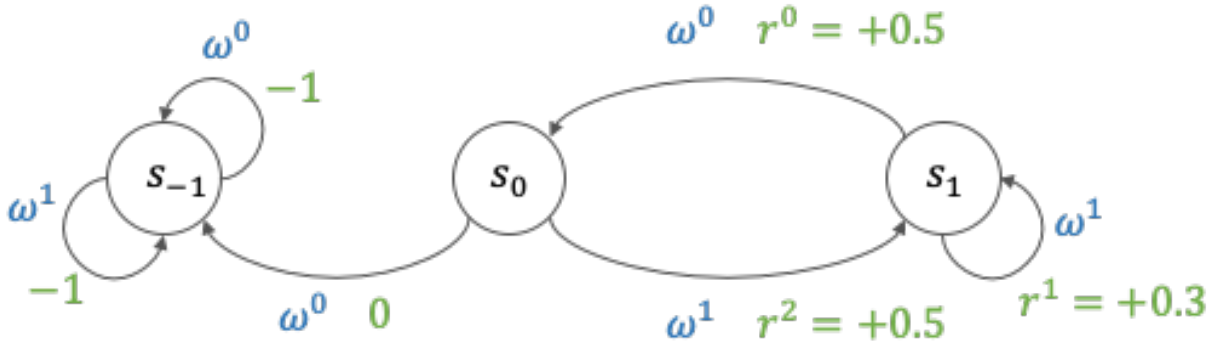


Figure E.1: **The *Duality* environment.**

As stated in Figure 6.4, without redundant primitives the solitary *CCV* implementation sacrifices Bellman optimality in general. We now use the *Duality* environment as an example to illustrate such an instance where the dominant strategy of the primitives does not coincide with the global optimal policy of the society, even if the primitives have full knowledge of their own valuations. The DSIC property of the Vickrey auction makes it straightforward to analyze the dominant strategies of the primitives, which the following paragraphs illustrate. Recall that based on the *CCV* learning objective, the valuation of the winner $\hat{\omega}$ at time t is the immediate reward plus the discounted *second*-highest bid (not the highest bid) at the next time-step: $r(s_t, \hat{\omega}_t) + \gamma \cdot \mathbf{b}'_{s_{t+1}}$. Let r^0 be equal to the environment reward $r(s_1, \omega^0)$, r^1 be equal to the environment reward $r(s_1, \omega^1)$, and r^2 be equal to the environment reward $r(s_0, \omega^1)$, where in Figure E.1 we see that $r^0 = 0.5$, $r^1 = 0.3$, and $r^2 = 0.5$

At state s_0, primitive ω^0 will bid 0, the lowest possible bid, because the local auction at state s_0 would lead to unbounded negative reward at s_{-1}. This means that the valuation that ω^0 has for winning at state s_1 is $r^0 + \gamma \cdot 0 = r(s_1, \omega^0)$, since 0 must be the second highest bid

at state s_0. Thus the dominant strategy for ω^0 is to truthfully bid r^0. At s_1, primitive ω^1 will bid some number c. Since activating ω^1 is a self-loop, ω^1 can sell s_1 back to itself in the next time-step. Thus the valuation that ω^1 has for winning at state s_1 is $r^1 + \gamma \cdot \min(r^0, c)$. Thus the dominant strategy for ω^0 is to truthfully bid $c = r^1 + \gamma \cdot \min(r^0, c)$.

In the undiscounted case, where $\gamma = 1$, if we solve for c, we have that if $r^1 > 0$, then $c = r^1 + r^0$, which is greater than r^0, which is what ω^0 would bid as its dominant strategy. Therefore, in the case that $r^1 > 0$, ω^1 will continue to sell s_1 to itself. As long as $r^1 < r^0 + r^2$, the self-loop at s_1 will be less optimal than cycling back and forth between s_0 and s_1.

In the discounted case, where $0 < \gamma < 1$ and we again assume that $r^1 > 0$. Solving c again, we see that if in the case that $r^0 < \frac{r^1}{1-\gamma}$, then $c = r^1 + \gamma r^0$ and $c > r^0$. In this case ω^1 will bid higher than ω^0 at state s_1, creating a perpetual self-loop. If instead $r^0 > \frac{r^1}{1-\gamma}$, then $c = \frac{r^1}{1-\gamma}$ and $c < r^0$. In this case ω^0 will bid higher than ω^1 at state s_1, which is the optimal global policy.

Therefore the *Duality* environment illustrates that without redundancy, for fortuitous settings of r^0 and r^1, the dominant strategy equilibrium of the society may coincide with the optimal global policy, but if the for other choices of r^0 and r^1, the dominant strategy equilibrium of the society may be globally suboptimal. Adding redundant primitives makes the auction utilities consistent with the Bellman optimality equations and therefore does not suffer from such suboptimal equilibria.

Two Rooms

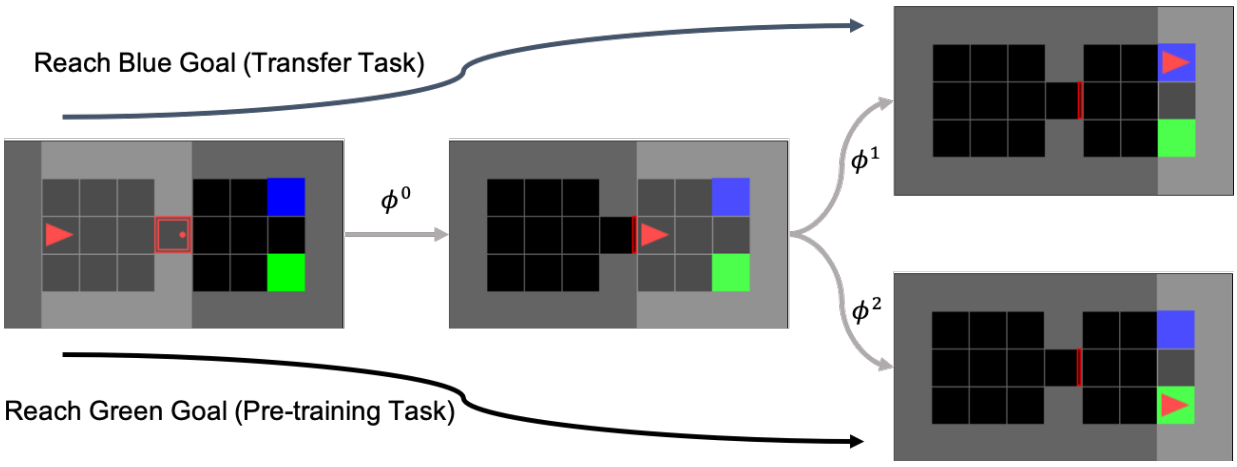


Figure E.2: **The *Two Rooms* environment.**

The transformations $\phi^{0:2}$ are subpolicies pre-trained with PPO and have an action space equivalent to the action space of the Minigrid environment. For these transformation subpolicies, the bidding policies, the value functions for the bidding policies, the non-hierarchical

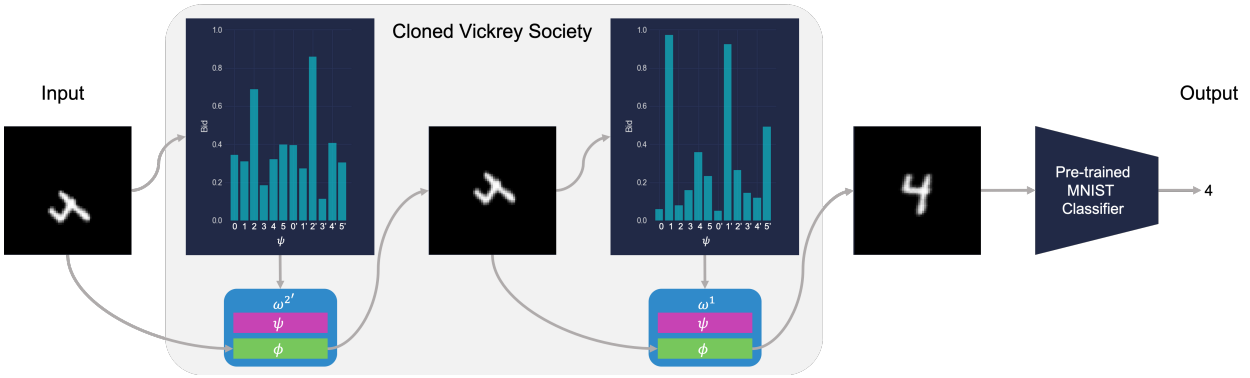


Figure E.3: **The *Mental Rotation* environment.**

monolithic baseline, and the hierarchical monolithic baseline, we adapted the convolutional neural network architecture from `https://github.com/lcswillems/rl-starter-files`.

ϕ^0 is initialized randomly in the room on the left and is pre-trained to take the done action once it opens the red door and enters the room on the right, upon which it receives a terminal reward. ϕ^1 is initialized randomly in the room on the right and is pre-trained to take the done action upon reaching the green square, upon which it receives a terminal reward. ϕ^2 is initialized randomly in the room on the right and is pre-trained to take the done action upon reaching the blue square, upon which it receives a terminal reward.

In the pre-training task, the society receives a terminal reward upon reaching the green square and no intermediate rewards. In the transfer task, the society receives a terminal reward upon reaching the blue square and no intermediate rewards. These subpolicies $\phi^{0:2}$ are frozen for these tasks. For all learners, we trained to convergence on the pre-training task, then we initialized training for the transfer task from the best saved checkpoint on the pre-training task. The non-hierarchical monolithic baseline could not solve the pre-training task so we did not consider it for the transfer task.

Mental Rotation

Environment Each 28×28 image in the MNIST training set was first inset into a black background of 64×64. Then the image first rotated either clockwise or counterclockwise by 60 degrees, then translated left, up, down, right by 29% of the image width, for a total of eight possible transformation combinations given these six affine transformations. These transformation combinations were taken from Chang et al. [58].

Primitives The transformations ϕ correspond to the same six affine transformations, summarized in the following table. The primitive ω^i, and its bidding policy ψ^i, correspond to the transformation ϕ^i. The clones are indicated by i and i'.

ϕ^0, $\phi^{0'}$	`rotate-counterclockwise`
ϕ^1, $\phi^{1'}$	`rotate-clockwise`
ϕ^2, $\phi^{2'}$	`translate-up`
ϕ^3, $\phi^{3'}$	`translate-down`
ϕ^4, $\phi^{4'}$	`translate-left`
ϕ^5, $\phi^{5'}$	`translate-right`

Neural network architecture The pre-trained MNIST classifier, the bidding policies, and the value functions for the bidding policies follow the same architecture as the convolutional neural network used in Chang et al. [58], with different output dimensions (10 as the output dimension of the MNIST classifier, 1 for the other networks). The PyTorch architecture is given below:

```
network = nn.Sequential(
            nn.Conv2d(1, 8, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(8, 8 * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(8 * 2),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(8 * 2, 8 * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(8 * 4),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(8 * 4, 8 * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(8 * 8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(8 * 8, outdim, 4, 1, 0, bias=False))
```

Appendix F

Local Credit Assignment

F.1 Background

This section supplements §7.3 with more background on algorithmic information theory and standard causality. For a more thorough treatment on the foundational mathematics and formalism, please refer to Li, Vitányi, et al. [201] for algorithmic information theory, to Pearl [246] for standard causality, and to Janzing and Schölkopf [166] and Peters, Janzing, and Schölkopf [248] for algorithmic causality.

Notation

We denote with bolded uppercase monospace a computation graph at a single level of abstraction (e.g. the model of execution $\mathtt{G}$, the model of credit assignment $\mathtt{C}$). We denote with blackboard bold (e.g. the algorithmic causal model of learning $\mathbb{L}$) a computation graph that represents multiple levels of abstraction.

We denote binary strings that represent the data nodes in $\mathbb{L}$ with lower case (e.g. x or $\mathtt{f}$), where script (x) is used to emphasize that the string typically represents a variable and monospace ($\mathtt{f}$) is used to emphasize that the string typically represents a function. We use bolded lower case (e.g. $\mathbf{x}$, $\boldsymbol{\delta}$, $\mathtt{f}$) to indicate a group of binary strings. We denote the function nodes in $\mathbb{L}$ (e.g., $\texttt{APPLY}$, $\texttt{UPDATE}$) with uppercase.

We write $\mathtt{f}(x) \to y$ to mean "a program $\mathtt{f}$ that takes a string x as input and produces a string y as output."

Background on algorithmic causality

The formalism of algorithmic causality derives from Janzing and Schölkopf [166] and Peters, Janzing, and Schölkopf [248], which builds upon algorithmic statistics [106]. Here we directly restate or paraphrase additional relevant definitions, postulates, and theorems from Janzing and Schölkopf [166] and Gács, Tromp, and Vitányi [106].

Algorithmic information theory

Kolmogorov complexity Kolmogorov complexity [295, 294, 181, 50, 49, 201] is a function $K : \{0,1\}^* \to \mathbb{N}$ from the binary strings $\{0,1\}^*$ to the natural numbers $\mathbb{N}$ that represents the amount of information contained in an object (represented by a binary string).

Definition F.1.1 (**Kolmogorov-complexity**). *Given a universal Turing machine and universal programming language as reference, the* ***Kolmogorov complexity*** $K(s)$ *is the length of the shortest program that generates* s*. The* ***conditional Kolmogorov complexity*** $K(y \mid x)$ *of a string* y *given another string* x *is the length of the shortest program that generates* y *given* x *as input. Let the shortest program for string* x *be denoted as* x^**. The* ***joint Kolomogorov complexity*** $K(x, y)$ *is defined as:*

$$K(x,y) \stackrel{+}{=} K(x) + K(x \mid y^*) \stackrel{+}{=} K(y) + K(y \mid x^*).$$

The **invariance theorem** [181] states that the Kolmogorov complexities of two strings written in two different universal languages differ only up to an additive constant. Therefore, we can assume any reference universal language for defining K (e.g. Python) and work with equalities ($\stackrel{+}{=}$) and inequalities ($\stackrel{+}{\geq}$, $\stackrel{+}{\leq}$) up to an additive constant.

Algorithmic mutual information Algorithmic mutual information I measures the amount of information two objects have in common:

Definition F.1.2 (**algorithmic mutual information**). *The* ***algorithmic mutual information*** *of two binary strings* x, y *is*

$$I(x:y) \stackrel{+}{=} K(x) + K(y) - K(x,y).$$

The ***conditional algorithmic mutual information*** *of strings* x, y *given string* z *is*

$$I(x:y \mid z) \stackrel{+}{=} K(x \mid z) + K(y \mid z) - K(x,y \mid z)$$

We can intuitively think of $I(x : y \mid z)$ as, given z, the number of bits that can be saved when describing y knowing the shortest program that generated x. We analogously extend this definition to the joint conditional algorithmic mutual information of strings $x_1, ..., x_n$ given strings $y_1, ..., y_m$, using "..." instead of ":" for notational a convenience:

Definition F.1.3 (**joint conditional algorithmic mutual information**). *Given strings* $x_1, ..., x_n$ *and* $y_1, ..., y_m$*, the* ***joint algorithmic mutual information*** *of* $x_1, ..., x_n$ *given* $y_1, ..., y_m$ *is:*

$$I(x_1, ..., x_n \mid y_1, ..., y_m) \stackrel{+}{=} \sum_{i=1}^{n} K(x_i \mid y_1, ..., y_m) - K(x_1, ..., x_n \mid y_1, ..., y_m).$$

Then algorithmic independence is the property of two strings that says that the description of one cannot be further compressed given knowledge of the other.

Definition F.1.4 (**algorithmic conditional independence**). *Given three strings x, y, z, x is **algorithmically conditionally independent** of y given z, denoted by $x \perp\!\!\!\perp y \mid z$, if the additional knowledge of y does not allow for stronger compression of x, given z. That is:*

$$x \perp\!\!\!\perp y \mid z \;\Leftrightarrow\; I(x : y \mid z) \stackrel{+}{=} 0.$$

Joint conditional independence of strings $x_1, ..., x_n$ given $y_1, ..., y_m$ is defined analogously as $I(x_1, ..., x_n \mid y_1, ..., y_m) \stackrel{+}{=} 0$.

Lemma F.1.1 (**algorithmic joint conditional independence**). *If strings $x_1, ..., x_n$ are algorithmically jointly conditionally independent given strings $y_1, ..., y_m$, then*

$$K(x_1, ..., x_n \mid y_1, ..., y_m) \stackrel{+}{=} \sum_{i=1}^{n} K(x_i \mid y_1, ..., y_m), \tag{F.1}$$

meaning that, conditioned on knowing $y_1, ..., y_m$, the length of the joint description of $x_1, ..., x_n$ cannot be further compressed than sum of the lengths of the descriptions of the individual strings x_i. The proof is by starting with $I(x_1, ..., x_n \mid y_1, ..., y_m) \stackrel{+}{=} 0$ and rearranging Def. F.1.3.

We state as a lemma the following result from Gács, Tromp, and Vitányi [106, Corollary II.8] that states the mutual information of strings x and y cannot be increased by separately processing by functions f and g.

Lemma F.1.2 (**information non-increase**). *Let f and g be computable programs. Then*

$$I(\mathtt{f}(x) : \mathtt{g}(y)) \stackrel{+}{\leq} I(x : y) + K(\mathtt{f}) + K(\mathtt{g}). \tag{F.2}$$

This intuitively makes sense: if $K(\mathtt{f})$ is constant with respect to x and $K(\mathtt{g})$ is constant with respect to y (i.e. $K(\mathtt{f}) \stackrel{+}{=} 0$ and $K(\mathtt{g}) \stackrel{+}{=} 0$), then mutual information cannot increase between x and y separately with $\mathtt{f}$ and $\mathtt{g}$. In particular, if x and y were independent to begin with (i.e. $I(x : y) \stackrel{+}{=} 0$), then $I(\mathtt{f}(x) : \mathtt{g}(y)) \stackrel{+}{=} 0$.

Terminology In this chapter, we regard the following statements about a program $\mathtt{f}(x) \to y$ as equivalent:

- "$K(\mathtt{f}) \stackrel{+}{=} 0$."
- "$\mathtt{f}$ is an $O(1)$-length program."
- "$K(\mathtt{f})$ is constant with respect to x."

Note that $K(\mathtt{f}) \stackrel{+}{=} 0$ implies that $\mathtt{f}$ and x are algorithmically independent (i.e. $I(\mathtt{f} : x) \stackrel{+}{=} 0$) because

$$0 \stackrel{+}{\leq} I(x : \mathtt{f}) \stackrel{+}{=} K(\mathtt{f}) - K(\mathtt{f} \mid x^*) \stackrel{+}{\leq} K(\mathtt{f}) \stackrel{+}{=} 0.$$

Causality

Before we review algorithmic causality, we first review some key concepts in standard causality: structural causal models and d-separation.

The following definition defines standard causal models over random variables as Bayesian networks represented as directed acyclic graphs (DAG), analogous to the algorithmic version we presented in Def. 7.3.1.

Definition F.1.5 (**structural-causal-model**). *A **structural causal model** (SCM) [245, 246] represents the assignment of random variable X as the output of a function, denoted by lowercase monospace (e.g. $\mathtt{f}$), that takes as input an independent noise variable N_X and the random variables $\{PA_X\}$ that represent the parents of X in a DAG:*

$$X := \mathtt{f}(\{PA_X\}, N_X). \tag{F.3}$$

Given the noise distributions $\mathbb{P}(N_X)$ for all variables X in the DAG, SCM entails a joint distribution $\mathbb{P}$ over all the variables in the DAG [248].

The graph-theoretic concept of d-separation is used for determining conditional independencies induced by a directed acyclic graph (see point 3 in Thm. F.1.3):

Definition F.1.6 (d-**separation**). *A path p in a DAG is said to be d-separated (or blocked) by a set of nodes Z if and only if*

1. *p contains a chain $i \to m \to j$ or fork $i \leftarrow m \to j$ such that the middle node m is in Z, or*

2. *p contains an inverted fork (or collider) $i \to m \leftarrow j$ such that the middle node m is not in Z and such that no descendant of m is in Z.*

*A set of nodes Z **d-separates** a set of nodes X from a set of nodes Y if and only if Z blocks every (possibly undirected) path from a node in X to a node in Y.*

Algorithmic causality

For convenience, we re-state the technical content from §7.3 here.

Definition 7.3.1 (computational graph). *Define a **computational graph** $\mathtt{G} = (\boldsymbol{x}, \mathtt{f})$ as a directed acyclic factor graph (DAG) of variable nodes $\boldsymbol{x} = x_1, ..., x_N$ and function nodes $\mathtt{f} = \mathtt{f}^1, ..., \mathtt{f}^N$. Let each x_j be computed by a program $\mathtt{f}^j$ with length $O(1)$ from its parents $\{pa_j\}$ and an auxiliary input n_j. Assume the n_j are jointly independent: $n_j \perp\!\!\!\perp \{n_{\neq j}\}$. Formally, $x_j := \mathtt{f}^j(\{pa_j\}, n_j)$, meaning that the Turing machine computes x_j from the input $\{pa_j\}, n_j$ using the additional program $\mathtt{f}^j$ and halts.*

Theorem 7.3.1 (algorithmic causal Markov condition). *Let $\{pa_j\}$ and $\{nd_j\}$ respectively represent concatenation of the parents and non-descendants (except itself) of x_j in a computational graph. Then $\forall x_j$, $x_j \perp\!\!\!\perp \{nd_j\} \mid \{pa_j\}^*$.*

Postulate 7.3.2 (faithfulness). *Given sets S, T, R of nodes in a computational graph, $I(S:T|R^*) \stackrel{+}{=} 0$ implies R d-separates S and T.*

The following theorem Janzing and Schölkopf [166, Thm. 3] establishes the connection between the graph-theoretic concept of d-separation with condition algorithmic independence of the nodes of the graph.

Theorem F.1.3 (**equivalence of algorithmic Markov conditions**). *Given the strings $x_1, ..., x_n$ and a computational graph, the following conditions are equivalent:*

1. ***Recursive form:*** *the joint complexity is given by the sum of complexities of each node x_j, given the optimal compression of its parents $\{pa_j\}$:*

$$K(x_1, ..., x_n) \stackrel{+}{=} \sum_{j=1}^{n} K(x_j|\{pa_j\}^*).$$

2. ***Local Markov Condition:*** *Every node x_j is independent of its non-descendants $\{nd_j\}$, given the optimal compression of its parents $\{pa_j\}$:*

$$I(x_j : nd_j|\{pa_j\}^*) \stackrel{+}{=} 0.$$

3. ***Global Markov Condition:*** *Given three sets S, T, R of nodes*

$$I(S:T|R^*) \stackrel{+}{=} 0$$

if R d-separates S and T.

Together, Thm. 7.3.1, Post. 7.3.2, and Thm. F.1.3 imply that variable nodes in a computational graph are algorithmically independent if and only if they are d-separated in the computational graph.

F.2 Assumptions

This section states our assumptions for the results that we prove in §F.4. This chapter analyzes learning algorithms from the perspective of algorithmic information theory, specifically algorithmic causality. To perform this analysis, we assume the following, and state our justifications for such assumptions:

1. The learning algorithm is implemented in on a universal Turing machine with a universal programming language.

   Justification: *This is a standard assumption in machine learning research that the machine learning algorithm can be implemented on a machine.*

2. Each initial parameter of the learnable functions `f` is jointly algorithmically independent of the other initial parameters.

Justification: *This is a standard assumption in machine learning research that the noise from the random number generator is given background knowledge [166, §2.3], thus allowing us to ignore possible dependencies among the parameters induced by the random number generator in developing our algorithms.*

3. The function nodes of ACML – `APPLY`, `UPDATE`, and the internal function nodes of Π – are $O(1)$-length programs.

Justification: *Assuming* `APPLY` *(which encompasses the transition and reward functions of the MDP, see §7.6) is an $O(1)$-length program is reasonable because it is a standard assumption that they are fixed with respect to the activations and parameters of the learning algorithm. Assuming* `UPDATE` *is an $O(1)$-length program is a standard assumption in machine learning research that the source code that computes the update rule (e.g. a gradient descent step) is agnostic to the feedback signals (e.g. gradients) it takes as input. Assuming the internal function nodes of Π are $O(1)$-length programs is reasonable because these function nodes are operations in the programming language like addition, multiplication, etc that are agnostic to the activations and parameters of the learning algorithm.*

4. $\mathbb{L}$ is faithful. That is, any conditional independence among the data nodes (`f`, x, δ, and the internal variable nodes of Π) in $\mathbb{L}$ is due to the causal structure of $\mathbb{L}$ rather than a non-generic setting of these data nodes.

Justification: *Faithfulness has been justified for standard causal models [220]. Deriving an algorithmic analog has been the subject of ongoing work [196, 195]. For our work, a violation of faithfulness means that two nodes x, y in the computational graph have $I(x : y) \overset{+}{=} 0$ but are not d-separated in the computational graph. This would happen if x and y were tuned in such a way that makes one compressible given the other. Given assumption (2) above, the source of a violation of faithfulness must be the data experienced by the learning algorithm. Indeed, the data could be such that after learning certain parameters within* `f` *may be conditionally independent given the training history, as suggested by Csordás, Steenkiste, and Schmidhuber [68], Filan et al. [92], and Watanabe [324]. However, as our focus is on theoretical results that hold regardless of the data distribution the learning algorithm is trained on, we consider the specific instances where the data does induce such faithfulness violations as "non-generic" and thus out of scope of the chapter.*

F.3 Additional Theoretical Results

All modular credit assignment mechanisms must be factorized in the following way:

Theorem F.3.1 (modular factorization). *The credit assignment mechanism* $\Pi(\boldsymbol{\tau}, \boldsymbol{f}) \to \boldsymbol{\delta}$ *is modular if and only if*

$$K\left(\boldsymbol{\delta} \mid \boldsymbol{x}, \boldsymbol{f}\right) \stackrel{+}{=} \sum_{t=1}^{T} K\left(\delta_t \mid \boldsymbol{x}, \boldsymbol{f}\right). \tag{F.4}$$

For modular credit assignment mechanisms, the complexity of computing feedback for the entire system is minimal because all redundant information among the gradients has been "squeezed out." This connection between simplicity and modularity is another way of understanding why if a credit assignment mechanisms were not modular it would be impossible for Π to modify a function without simultaneously modifying another, other than due to non-generic instances when δ_t has a simple description, i.e. $\delta_t = 0$, which, unless imposed, are not likely to hold over all iterations of learning.

F.4 Proofs

Given the assumptions stated in §F.2, we now provide the proofs for our theoretical results. We will prove Lemma 7.5.1 first. Together with the faithfulness postulate (Post. 7.3.2) and the equivalence of algorithmic Markov conditions (Thm. F.1.3) we can prove algorithmic independence by inspecting the graph of $\mathbb{L}$ for d-separation.

Dynamic Modularity and the Algorithmic Causal Model of Learning

Lemma 7.5.1 (algorithmic causal model of learning). *Given a model of execution* `E` *and of credit assignment* `C`*, define the* ***algorithmic causal model of learning*** *(ACML) as a dynamic computational graph* $\mathbb{L}$ *of the learning process. We assume* Π *has its own internal causal structure with internal variable and function nodes. The function nodes of* $\mathbb{L}$ *are* `APPLY`*,* `UPDATE`*, and internal function nodes of* Π*. The variable nodes of* $\mathbb{L}$ *are* x*,* $\boldsymbol{f}$*,* δ*, and internal variable nodes of* Π*.* `APPLY` *and* `UPDATE` *are assumed to have length* $O(1)$*. The internal function nodes of* Π *jointly independent, and along with the variable nodes of* $\mathbb{L}$*, are assumed to not have length* $O(1)$*. Then these variable nodes satisfy the algorithmic causal Markov condition with respect to* $\mathbb{L}$ *for all steps of credit assignment.*

Proof. If we can express Π in the form of the computational graph defined in Definition 7.3.1, then we will have shown that $\mathbb{L}$ is a well-defined computational graph, and so by Thm. 7.3.1 it satisfies the algorithmic causal Markov condition. To express Π as a computational graph, let us denote the internal function nodes Π as $\mathtt{g}^j$, indexed by j. For a given input u_j into $\mathtt{g}^j$, we can equivalently write $\mathtt{g}^j(u_j)$ as $\texttt{APPLY}(\mathtt{g}^j, u_j)$. Then since we assume `APPLY` is $O(1)$, by treating $\mathtt{g}^j$ as analogous to the auxiliary input n_j in Definition 7.3.1, Π is a well-defined computational graph, which completes the proof.

□

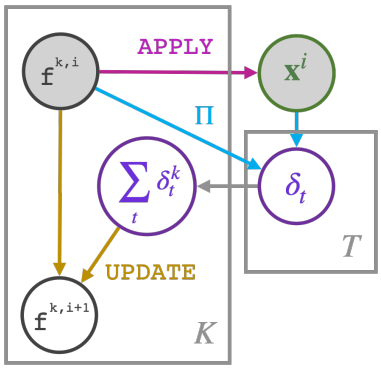


Figure F.1: This figure shows the computation graph of $\mathbb{L}$ across one credit assignment update. Inputs to the credit assignment mechanism are shaded. A modular credit assignment mechanism (shown with blue edges) is equivalent to showing the gradients δ_t as conditionally independent, as shown by the plate notation labeled with T. Dynamic modularity at iteration $i-1$ is equivalent to showing that the functions $\mathtt{f}^{k,i}$ are inside the plate labeled with N. Then because the `UPDATE` operation, shown with yellow edges, operates only within the plate labeled with N, the updated functions $\mathtt{f}^{k,i+1}$ are also conditionally independent given $(\mathbf{x}, \mathtt{f})$.

Remark. By Lemma F.1.2, if a set of data nodes in $\mathbb{L}$ are independent, then processing them separately with factor nodes of $\mathbb{L}$ will maintain this independence. For example, given that the `UPDATE` operation is applied in separately for each pair $(\mathtt{f}^k, \sum_t \delta_t^k)$ to produce a corresponding $\mathtt{f}^{k\prime}$, then if $(\mathtt{f}^k, \sum_t \delta_t^k)$ were independent of $(\mathtt{f}^j, \sum_t \delta_t^j)$ before applying `UPDATE`, then $\mathtt{f}^{k\prime}$ would be independent of $\mathtt{f}^{j\prime}$ after applying `UPDATE`.

Theorem 7.4.1 (modular credit assignment). *Dynamic modularity is enforced at learning iteration i if and only if static modularity holds at iteration $i = 0$ and the credit assignment mechanism satisfies the modularity constraint.*

Proof. We will prove by induction on i. The inductive step will make use of the equivalence between d-separation and conditional independence.

Base case: $i = 1$. There is no training history, so static modularity is equivalent to dynamic modularity.

Inductive hypothesis: Assuming that dynamic modularity holds if and only if static modularity holds at $i = 0$ and modular credit assignment holds for learning iteration $i-1$, dynamic modularity holds if and only if static modularity and modular credit assignment hold for learning iteration i.

Inductive step: The modularity constraint states

$$I\left(\delta_1, ..., \delta_M \mid \mathbf{x}_i, \mathtt{f}_i\right) \stackrel{+}{=} 0.$$

Dynamic modularity at iteration $i-1$ states that

$$\forall k \neq j, \quad I\left(\mathtt{f}^{k,i} : \mathtt{f}^{j,i} \;\middle|\; \mathbf{x}_{i-1}, \mathtt{f}_{i-1}\right) \stackrel{+}{=} 0.$$

These two above statements correspond to the computational graph in Fig. F.1. Note that by Def. F.1.1, disjoint subsets of $\delta_1, ..., \delta_T$ also have have zero mutual information up to an additive constant. Letting these subsets be $\sum_t \delta_t^k$ where k is the index of function $\mathtt{f}^k$ in $\mathtt{f}$, then

$$I\left(\sum_t \delta_t^1, ..., \sum_t \delta_t^N \;\middle|\; \mathbf{x}_i, \mathtt{f}_i\right) \stackrel{+}{=} 0. \tag{F.5}$$

Then, as we can see by direct inspection in Fig. F.1, $\mathtt{f}^{k_i}$ and $\mathtt{f}^{j_i}$ are d-separated by $(\mathbf{x}_i, \mathtt{f}_i)$, which is equivalent to saying that dynamic modularity holds for iteration i. □

Theorem 7.5.2 (modularity criterion). *If $\mathbb{L}$ is faithful, the modularity constraint holds if and only if for all i, outputs δ_t and $\delta_{\neq t}$ of Π are d-separated by its inputs $\boldsymbol{x}$ and $\boldsymbol{f}$.*

Proof. The forward direction holds by the equivalence of algorithmic causal Markov conditions (Thm. F.1.3), and the backward direction holds by the faithfulness assumption. □

Theorem F.3.1 (modular factorization). *The credit assignment mechanism $\Pi(\boldsymbol{\tau}, \boldsymbol{f}) \to \boldsymbol{\delta}$ is modular if and only if*

$$K\left(\boldsymbol{\delta} \mid \boldsymbol{x}, \boldsymbol{f}\right) \stackrel{+}{=} \sum_{t=1}^{T} K\left(\delta_t \mid \boldsymbol{x}, \boldsymbol{f}\right). \tag{F.6}$$

Proof. The proof comes from the definition of algorithmic mutual information (Def. F.1.3).

$$K\left(\boldsymbol{\delta} \mid \mathbf{x}, \mathtt{f}\right) \stackrel{+}{=} \sum_{t=1}^{T} K\left(\delta_t \mid \mathbf{x}, \mathtt{f}\right) \tag{F.7}$$

$$\sum_{t=1}^{T} K\left(\delta_t \mid \mathbf{x}, \mathtt{f}\right) - K\left(\boldsymbol{\delta} \mid \mathbf{x}, \mathtt{f}\right) \stackrel{+}{=} 0 \tag{F.8}$$

$$I\left(\delta_1, ..., \delta_T \mid \mathbf{x}, \mathtt{f}\right) \stackrel{+}{=} 0. \tag{F.9}$$

□

Modularity in Reinforcement Learning

In the following, we sometimes use b_t instead of b_{s_t} to reduce clutter.

Corollary 7.6.0.1 (policy gradient). *All policy gradient methods do not satisfy the modularity criterion.*

Proof. It suffices to identify a single shared hidden variable that renders $\delta_1, ..., \delta_T$ not d-separated. Computing the policy gradient includes the log probability of the policy as one of its terms. Computing this log probability for any action involves the same normalization constant $\sum_k b^k$. This normalization constant is a hidden variable that renders $\delta_1, ..., \delta_T$ not d-separated, as shown in Fig. F.2. □

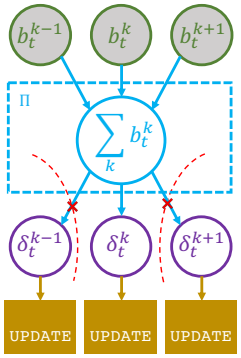


Figure F.2: This figure shows part of the computational graph within Π for policy gradient methods. Conditioning on $\mathbf{x}$ implies we condition on the lightly shaded nodes. $\sum_k b_t^k$ is the shared hidden variable that renders $\delta_1, ..., \delta_T$ not d-separated.

Corollary 7.6.0.2 (n-step TD). *All TD($n > 1$) methods do not satisfy the modularity criterion.*

Proof. It suffices to identify a single shared hidden variable that renders $\delta_1, ..., \delta_T$ not d-separated. TD($n > 1$) methods include a sum of estimated returns or advantages at different steps of the decision sequence that is shared among multiple δ_t's. This sum is the hidden variable that renders $\delta_1, ..., \delta_T$ not d-separated, as shown in Fig. F.3. □

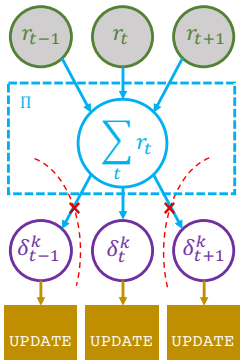


Figure F.3: This figure shows part of the computational graph within Π for TD($n > 1$) methods. Conditioning on $\mathbf{x}$ implies we condition on the lightly shaded nodes. $\sum_t r_t$ is the shared hidden variable that renders $\delta_1, ..., \delta_T$ not d-separated.

Corollary 7.6.0.3 (single-step TD). *TD(0) methods satisfy the modularity criterion for acyclic $\boldsymbol{x}$.*

Proof. If the decision mechanism $\mathtt{f}^k$ were selected (i.e. $w_t^k = 1$) at step i, TD(0) methods produce, for some function g, gradients as $\delta_t^k := g(b_t^k, s_t, s_{t+1}, r_t, \mathtt{f})$. Otherwise, $\delta_t^k := 0$. The

only hidden variable is $[\max_j b^j_{s_{t+1}}]$, and for acyclic $\mathbf{x}$ there is only one state s_t in $\mathbf{x}$ that transitions into s_{t+1}. Therefore the hidden variable is unique to each of $\delta_1, ..., \delta_t$, so $\delta_1, ..., \delta_t$ remain d-separated, as shown in Fig. F.4. □

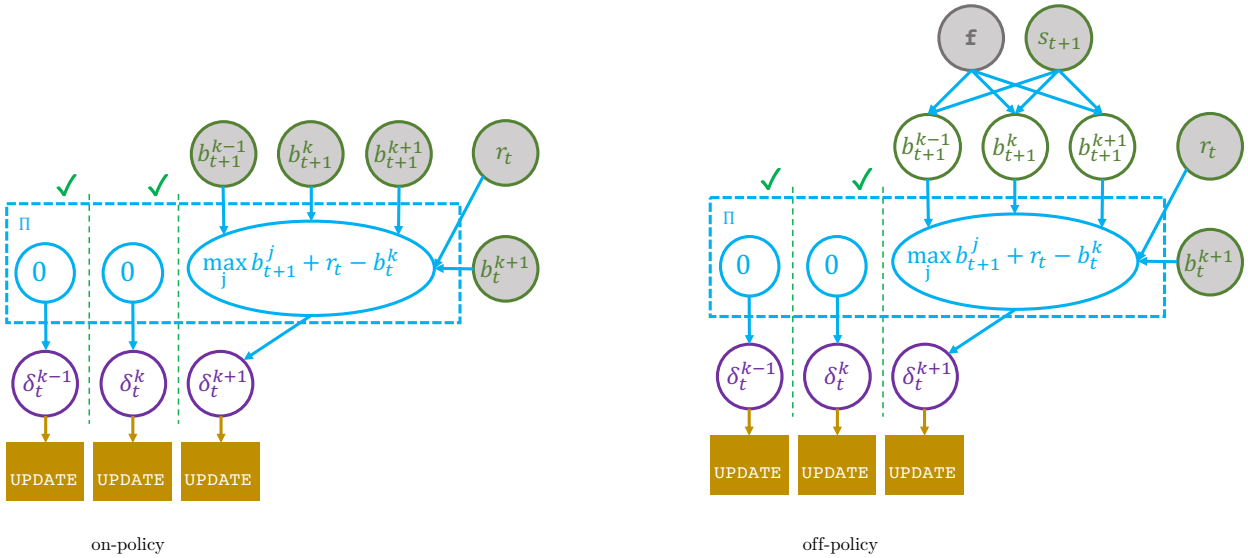


Figure F.4: This figure shows part of the computational graph within Π for on-policy and off-policy TD(0) methods. Conditioning on $(\mathbf{x}, \mathtt{f})$ implies we condition on the lightly shaded nodes. For on-policy methods such as CVS and SARSA, the hidden variable would be $\max_k b^k_{t+1}$ for CVS and the bid corresponding to the decision mechanism that was sampled through ε-greedy for SARSA. The figure shows $\max_k b^k_{t+1}$ for concreteness. For off-policy methods such as Q-learning, the bids b_{t+1} are computed from s_{t+1} and $\mathtt{f}$, both of which we condition on. In both cases, the hidden variable is only parent to one of the δ_t's, and thus the $\delta_1, ..., \delta_T$ remain d-separated.

Corollary 7.6.-3.1 (tabular). *In the tabular setting, Thm. 7.4.1 holds for Q-learning, SARSA, and CVS.*

Proof. In the tabular setting, decision mechanisms are columns of the Q-table corresponding to each action. These columns do not share parameters, so static modularity holds. Then because Q-learning, SARSA, and CVS are TD(0) methods, by Corollary 7.6.0.3, their credit assignment mechanisms are modular. Therefore Thm. 7.4.1 holds. □

Corollary 7.6.-3.2 (function approximation). *In the function approximation setting, Thm. 7.4.1 holds for TD(0) methods whose decision mechanisms do not share parameters.*

Proof. The decision mechanisms of CVS do not share weights, so static modularity holds. By Corollary 7.6.0.3 its credit assignment mechanism is modular. Therefore Thm. 7.4.1 holds. □

F.5 Simulation Details

We implemented our simulations using the PyTorch library [243].

Implementation Details

The underlying PPO [282] implementation used for CVS, PPO, and PPOF used a policy learning rate of 4×10^{-5}, a value function learning rate of 5×10^{-3}, a clipping ratio of 0.2, a GAE [281] parameter of 0.95, a discount factor of 0.99, entropy coefficient of 0.1, and the Adam [174] optimizer. For all algorithms, the policy and value functions for our algorithms were implemented as fully connected neural networks that used two hidden layers of dimension 20, with a ReLU activation. All algorithms performed a PPO update every 4096 samples with a minibatch size of 256.

Training Details

All learning curves are plotted from ten random seeds, with a different learning algorithm represented by a different hue. The dark line represents the mean over the seeds. The error bars represent one standard deviation above and below the mean.

Our protocol for transfer is as follows. A transfer problem is defined by a (training, transfer) task pair, where the initial network parameters for the transfer task are the network parameters learned the training task for H samples. In our simulations, we set H to 10^7 because that was about double the number of samples for all algorithms to visually converge on the training task for all seeds. To calculate the relative sample efficiency of CVS over PPO and PPOF (e.g. 13.9x and 6.1x respectively in the bottom-up right corner of Fig. 7.6), we set the criterion of convergence as the number of samples after which the return deviates by no more than $\varepsilon = 0.01$ from the optimal return for 30 epochs of training, where each epoch of training trains on 4096 samples.

Environment Details

The environment for our experiments shown in Figs. 7.4 and 7.6 are represented as discrete-state, discrete-action MDPs. Each state is represented by a binary-valued vector.

The structure of the MDP can best be explained via an analogy to a room navigation task, which we will explain in the context of the $A \to B \to C$ task in the *linear chain* topology. In this task, there are four rooms, room 0, room 1, room 2, and room 3. Room 0 has two doors, labeled A and F, that lead to room 1. Room 1 has two doors, labeled B and E, that lead to room 2. Room 2 has two doors, labeled C and D. Doors are unlocked by keys. The state representation is a concatenation of two one-hot vectors. The first one-hot vector is of length four; the "1" indicates the room id. The second one-hot vector is of length six; the "1" indicates the presence of a key for door A, B, C, D, E, or F. Only one key is present in a room at any given time. If the agent goes through the door corresponding to the key present

in the room, then the agent transitions into the next room; otherwise the agent stays in the same room. In the last room, if the agent opens the door corresponding to the key that is present in the room, then the agent receives a reward of 1. All other actions in every other state receive a reward of 0. Therefore the agent only gets a positive reward if it opens the correct sequence of doors. For all of our experiments, the optimal policy is acyclic, but a suboptimal decision sequence could contain cycles.

Therefore, for the training task in the *linear chain* topology where the optimal solution is $A \rightarrow B \rightarrow C$, the optimal sequence of states are

```
[1, 0, 0, 0 ; 1, 0, 0, 0, 0, 0]  # room 0 with key for A
[0, 1, 0, 0 ; 0, 1, 0, 0, 0, 0]  # room 1 with key for B
[0, 0, 1, 0 ; 0, 0, 1, 0, 0, 0]  # room 2 with key for C.
```

For the transfer task whose optimal solution is $A \rightarrow B \rightarrow D$, the optimal sequence of states are

```
[1, 0, 0, 0 ; 1, 0, 0, 0, 0, 0]  # room 0 with key for A
[0, 1, 0, 0 ; 0, 1, 0, 0, 0, 0]  # room 1 with key for B
[0, 0, 1, 0 ; 0, 0, 0, 1, 0, 0]  # room 2 with key for D.
```

Whereas for the *linear chain* topology the length of the optimal solution is three actions, for the *common ancestor* and *common descendant* topologies this length is two actions. *Common ancestor* and *common descendant* are multi-task problems. As a concrete example, the training task for *common ancestor* is a mixture of two tasks, one whose optimal solution is $A \rightarrow B$ and one whose optimal solution is $A \rightarrow C$. Following the analogy to room navigation, this task is set up such that after having gone through door A, half the time there is a key for door B and half there is a key for door C.

Computing Details

For our experiments, we used the following machines:

- AWS: c5d.18xlarge instance
- Azure: Standard D64as_v4 (64 vcpus, 256 GiB memory), 50GiB Standard SD attached.

The average runtime training on 10^7 samples was three hours for PPO and PPOF and 6 hours for CVS.