

Characterizing Circuits with Deep Embeddings

Arjun Mishra



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-190

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-190.html>

August 13, 2021

Copyright © 2021, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to thank Professor Vladimir Stojanovic for his guiding hand and encouragement throughout my research at Berkeley. I must also thank my teammates, Kourosh Hakashmenshi and Rohan Lagewag, for being such incredible collaborators. There were many ups and downs but I learned a lot. Finally, I would like to thank the Regents of the UC and particularly the administration at Berkeley for taking a chance on me as an unproven but passionate teenager.

Finally, I must thank my family for their support, love, and time through the many years it took to get here.

Characterizing Circuits with Deep Embeddings

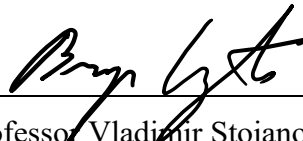
by Arjun Mishra

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor Vladimir Stojanovic
Research Advisor

08/11/2021

(Date)



Professor Borivoje Nikolic
Second Reader

8/12/2021

07/31/2021

Characterizing Circuits with Deep Embeddings

By Arjun Mishra

A thesis submitted in partial satisfaction of the requirements for the degree
of Master of Science in Electrical Engineering and Computer Sciences in the
graduate division of University of California, Berkeley

*To my family, who has always pushed me to be the best version of myself
and instilled a love of learning in me from a young age.*

Contents

CONTENTS	II
LIST OF FIGURES	III
ACKNOWLEDGEMENTS	IV
ABSTRACT	1
INTRODUCTION	2
BACKGROUND	4
2.1 NAÏVE AUTOENCODERS	4
2.2 VARIATIONAL AUTOENCODERS	5
RELATED WORK	6
3.1 GLOVE EMBEDDINGS.....	6
3.2 CHEMICAL EMBEDDINGS	6
3.3 BROADER REPRESENTATION LEARNING	6
SYSTEM OVERVIEW	7
4.1 DATASET.....	7
4.1 ENCODING FLOW	9
4.2 TRAINING PROCESS.....	10
USE CASES	11
5.1 CIRCUIT TRANSFER.....	11
5.2 CIRCUIT CHARACTERIZATION.....	11
5.3 DATA AND COMPUTATION LIMITATIONS	11
IMPLEMENTATION	12
6.1 MODEL IMPLEMENTATION	12
EVALUATION	14
7.1 MODEL EVALUATION.....	14
FUTURE WORK	15
8.1 MODEL EVALUATION.....	15
8.2 MODEL TRAINING	15
CONCLUSION	16
BIBLIOGRAPHY	17

List of Figures

FIGURE 1: EMBEDDINGS CORRESPOND TO INTUITIVE FEATURES [4]	2
FIGURE 2: AUTOENCODER ARCHITECTURE	4
FIGURE 3: CLUSTERING BOMB DATASET WITH PCA ON DEVICE TYPE. DEV 1 AND DEV 4 ARE LOW-VOLTAGE, DEVICES 2 AND 3 ARE HIGH-VOLTAGE.	7
FIGURE 5: HVT REFERS TO HIGH VOLTAGE TRANSISTOR, LVT TO LOW VOLTAGE TRANSISTOR, AND NOM TO NORMAL VOLTAGE TRANSISTOR	9
FIGURE 6: LOADING DATA FROM THE BOMB OPEN DATASET.....	12
FIGURE 7: COMPUTING THE LOSS FOR A GIVEN LOOP ITERATION. NOTE THAT THIS IS THE CODE IMPLEMENTATION OF THE FORMULA DESCRIBED EARLIER.	13
FIGURE 8: COMPARING TEST PERFORMANCE OF EMBEDDING MODELS VS THE BASELINE.....	14

Acknowledgements

I would like to thank Professor Vladimir Stojanovic for his guiding hand and encouragement throughout my research at Berkeley. I must also thank my teammates, Kouros Hakhshemshahi and Rohan Lagewag, for being such incredible collaborators. There were many ups and downs but I learned a lot. Finally, I would like to thank the Regents of the UC and particularly the administration at Berkeley for taking a chance on me as an unproven but passionate teenager.

Finally, I must thank my family for their support, love, and time through the many years it took to get here.

Abstract

Characterizing Circuits with Deep Embeddings

By

Arjun Mishra

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

In recent years, the use of machine learning for solving complex problems has spread like wildfire. Specifically, machine learning has proved to be very effective in generating embeddings, both for tasks related to simple words/images and for those involving complex data arising in the domains of biology and chemistry. Inspired by these breakthroughs, we look at the problem of generating embeddings from an underlying dataset of circuits and prove their utility on several posterior tasks.

Chapter 1

Introduction

There has been growing interest in the application of machine learning to less traditional datasets. This is both due to the proliferation of such datasets, but also a result of improvements in the ability to encode information. Encoding information into embeddings, lower dimensional representations of high dimensional data, is useful in allowing raw data to be used for computational tasks. These encodings have been shown to be useful in several domains. The traditional techniques for encoding were PCA and T-SNE; these have some key limitations [1, 2]. In general, linear dimensionality reduction techniques, such as PCA, tend to be extremely fast to run and deterministic. However, by virtue of the linear assumption — they struggle to encode information from difficult datasets. These limitations and the inability to generate good representations have led researchers to investigate generating embeddings using online machine learning techniques. Deep Learning has proven to be a very effective one of these techniques.

Deep learning took a major step in 2012 with the development of AlexNet on the ImageNet benchmark [3]. One of the key advantages of deep learning is that the feature detection is not only automatic but also extremely effective. In a deep learning model, similar inputs to the network, such as say different images of a cat, activate very similar feature maps within the neural network. This then immediately suggests that the networks used for classification and regression could also be used for generating representations.

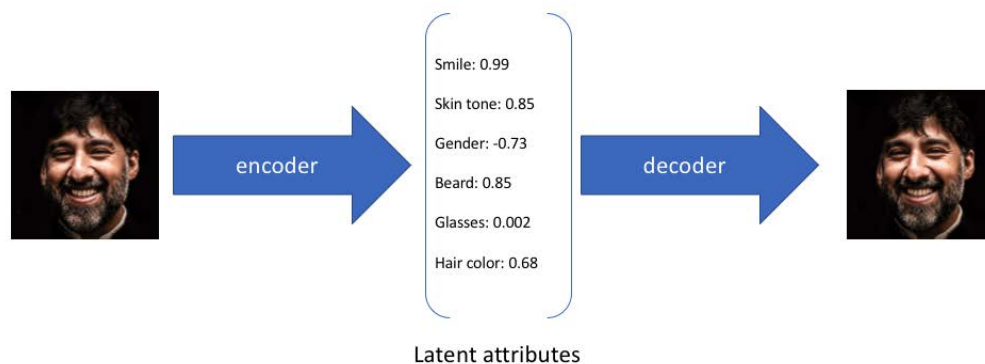


Figure 1: Embeddings correspond to intuitive features [4]

The technique of encoding inputs and outputting lower-dimensional vectors is referred to as Representation Learning. Representation Learning is an example of unsupervised learning as explicit labels are not needed to generate representations.

In this paper, we explore the use of Representation Learning to characterizing “circuits”, using the open BOMB dataset. Specifically, we aim to show that training embeddings on the BOMB dataset is both

feasible and *useful* [5]. We achieve this goal by training many flavors of Variational Auto Encoders (VAEs) on top of the BOMB dataset and then comparing these VAEs both quantitatively and qualitatively:

- visually inspecting the clustering of high-level information within the latent space
- measuring outright loss
- using the encodings to feed a downstream Square Law prediction task

The major contributions of this paper are:

- We demonstrate that a VAE algorithm can converge on the BOMB dataset
- We show that that representation learning leads to better performance for low-data downstream tasks
- We find that using the embeddings can lead to a major improvement in computing latency on downstream tasks given a reduction in dimensionality by over **2500x**

Chapter 2

Background

2.1 Naïve Autoencoders

Autoencoders are computational models used to encode and decode information. Figure 2 illustrates a model architecture for an autoencoder. Note that both the encoder and decoder can be any function which maps from an input dimension to the latent dimension for the encoder and from the latent dimension to the input dimension for the decoder. The reason that the input dimension of the encoder must match the output dimension of the decoder is because in the most naïve example of an autoencoder the loss function would optimize the reconstruction loss. [5, 6]

1. To formalize this let us define variable input $x \in \mathbb{R}^d = X$, which is mapped to $h \in \mathbb{R}^p = F$
2. The transition of the encoder as $\phi: X \rightarrow F$
3. The transition of the decoder as $\varphi: F \rightarrow X$
4. $\phi, \varphi = \underset{\phi, \varphi}{\operatorname{argmin}} ||X - (\varphi \circ \phi)X||^2$

Purely minimizing reconstruction loss is flawed due to a tendency to overfit and therefore instead in this paper we chose to primarily focus on using Variational Auto Encoders (VAEs).

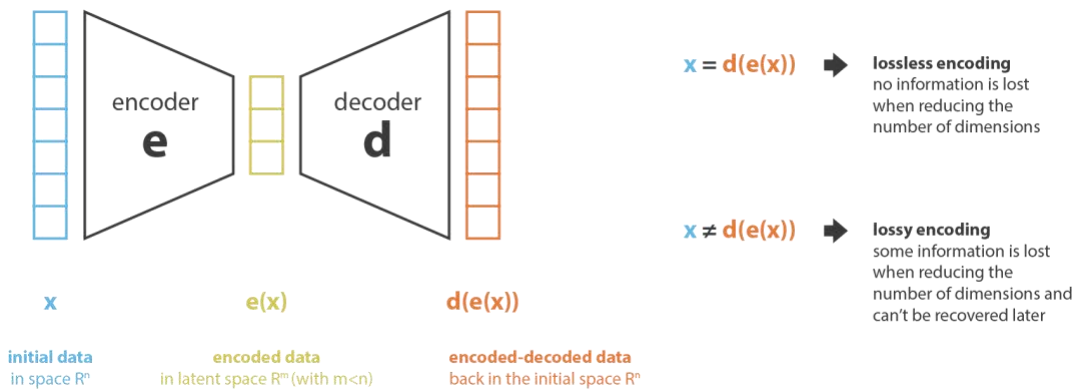


Figure 2: Autoencoder Architecture

There is great variability in the encoding or decoding architectures that autoencoders use. In a simple design, one could use a fully connected network as both the encoder and decoder, but it is possible to mix and match, for example to have the encoder be a Convolutional Neural Network (CNN) and the decoder be a Fully Connected Neural Network. In this paper for all possible designs, we chose to use fully connected layers as both the encoder and decoder and mirror the number of layers on either side of the information bottleneck.

2.2 Variational Autoencoders

VAEs are slightly different than traditional autoencoders. The key difference between a VAE and a classic autoencoder is in the idea of “variational inference.” To develop some intuition, consider that there is an observable variable x and a latent variable z .

Ideally, we want $p(z|x)$ which is equal to $\frac{p(x|z)p(z)}{p(x)}$ but computing the $p(x)$ here is since computing $\int p(x|z)p(z)$ is intractable. We approximate the intractable distribution by defining a distribution $q(z|x)$. To find a good approximation, we want to ensure that the parameters of distribution q are similar to those of the intractable distribution.

Mathematically we have the following optimization requirement $\min (KL(q(z|x)||p(z|x)))$. Through further derivation, it turns out that we can minimize the earlier expression by maximizing $\mathbb{E}_{q(z|x)} \log p(x|z) - KL(q(z|x)||p(z))$. The first term of this expression maps to reconstruction likelihood while the second term captures how similar the learned distribution q is to the prior p .

Earlier we said that the optimization equation for naive auto encoder was

$$\phi, \varphi = \underset{\phi, \varphi}{\operatorname{argmin}} \|X - (\varphi \circ \phi)X\|^2 \text{ which we call } \mathcal{L}.$$

Our loss function for our VAE is now $\mathcal{L} + \sum_j KL(q_j(z|x)||p(z))$. β , a key hyper-parameter, is the regularization coefficient that impacts the reconstruction loss.

Chapter 3

Related Work

3.1 GloVe Embeddings

Reducing the dimensionality of information to improve the performance of downstream algorithm has been an important theme in machine learning research. This was famously explored in the original GloVe paper [7]. GloVe sought to accomplish this reduction via an unsupervised algorithm which tried to optimize the word-cooccurrence. Their insight was to use a matrix of word-cooccurrence; such a matrix would have the ability to encode meaningful information about how various words might be related. GloVe directly optimizes a training objective that stipulates that a word vector's dot product equals the logarithm of the word's probability of co-occurrence. With this clever insight, GloVe paved the way for large improvements in the field of neural machine translation as underlying encodings were much better. These embeddings have been critical in applying GRU and LSTM technology [8].

3.2 Chemical Embeddings

The utility of Representation learning is not limited to the domain of words and images alone. Representation learning has also proven to be extremely useful in other domains. For example, researchers have explored the use of encodings to represent chemical molecules. These embeddings have successfully captured properties of the molecules [9].

3.3 Broader Representation Learning

Representation Learning is a quickly evolving field with continual advances in the ability of deep neural-networks to encode data. This technique was pioneered by Bengio [10]. Bengio's insight was that the feature extraction that deep learning models perform in their initial tasks of classification or regression are themselves encoders. Since then, there has been an explosion in using representation learning techniques on data that is derived from both digital and physical properties. Continual developments in creating image and word embeddings have powered large advances in interdisciplinary uses of representation learning such as bio-medical retrieval [11]. Representation learning has proved to be a crucial method of allowing more data in the world to be digitized and operationalized. While in this paper we chose to use a variational autoencoder, the overall trend of utilizing deep learning to allow for broader scale machine learning stands to improve the world markedly.

Chapter 4

System Overview

4.1 Dataset

For this project, the initial data used to characterize circuits is from the BOMB dataset. The BOMB dataset is an open-source dataset which contains characterizations of various transistors from a variety of manufacturing technologies. The dataset includes data points created by measuring bias and other small-signal parameters from transistors across a range of temperatures, Monte Carlo variations, process types, etc. This is a $[N \times 10 \times 11 \times 11 \times 11]$ dataset where N is the number of datapoints in usage, 10 is the number of observed circuit characteristics, and each of the 11 dimensions corresponds to V_{ds} , V_{gs} , V_{bs} value. The data can be clustered with PCA. For example, in Figure 3, we see that PCA shows clear clustering by the process type. Process type refers to carrier mobilities for PMOS and NMOS transistors. In Figure 4, we can see the effect of clustering by the device type, which refers to what voltage the transistor prefers.

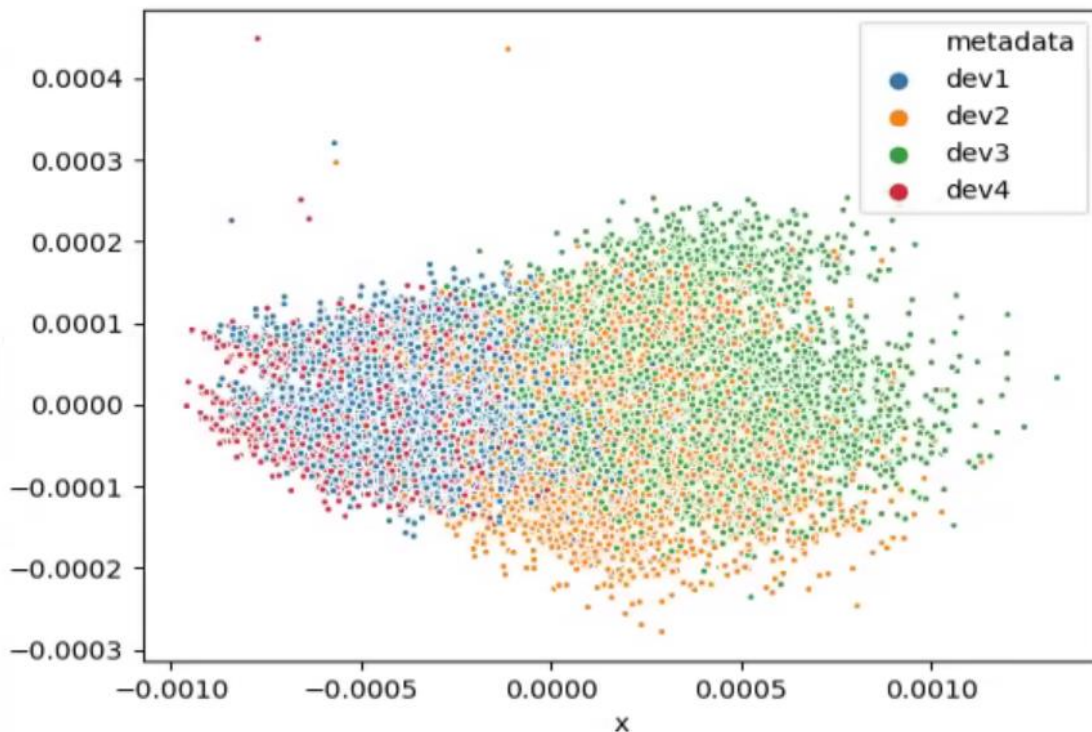


Figure 3: Clustering BOMB Dataset with PCA on Device Type. Dev 1 and Dev 4 are low-voltage, Devices 2 and 3 are high-voltage.

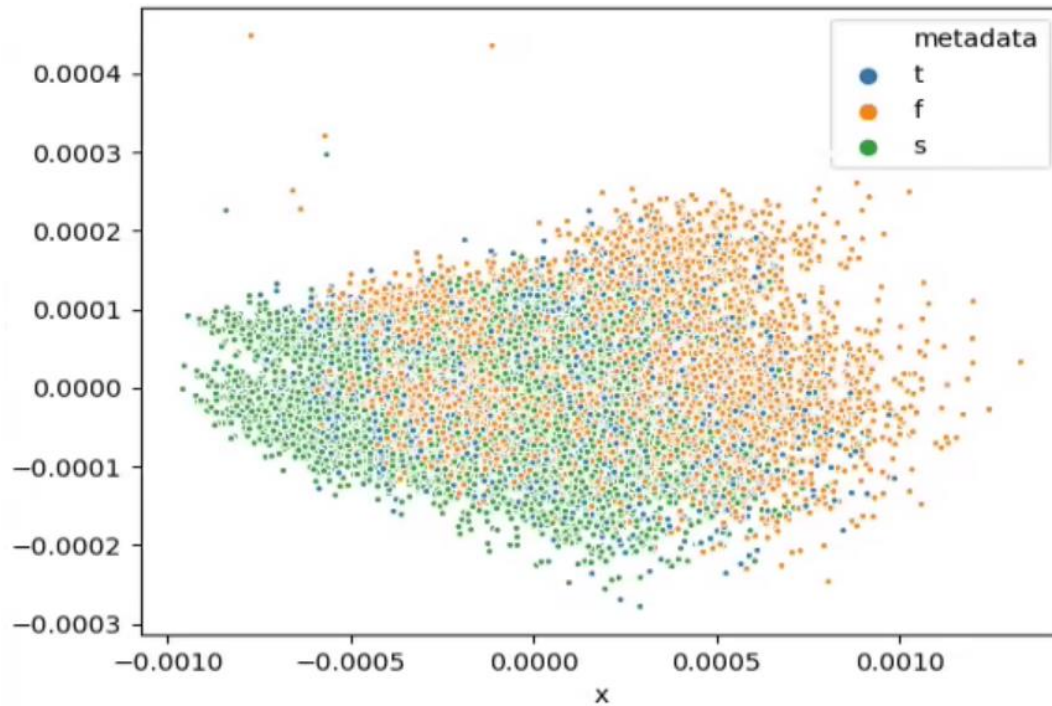


Figure 4: Clustering BOMB Dataset with PCA on Process Type. T refers to Typical, F to Fast, and S to slow

We segment the original dataset into a training set and test set. We want to test the ability of an embedding to generalize knowledge about the original dataset, so our testing set only includes high voltage devices and our training set contains only low and medium voltage devices. When we later perform a downstream prediction task, we want to show that the downstream task can be achieved more accurately with the embeddings as opposed to the raw data. Therefore, to ensure that we can show that the embeddings generalize and therefore perform better, we need to ensure that the test set contains data, high voltage devices, which neither the embeddings nor raw data have seen. To get a sense of how discernible voltage diverse devices are, see the T-SNE clustering in Figure 5.

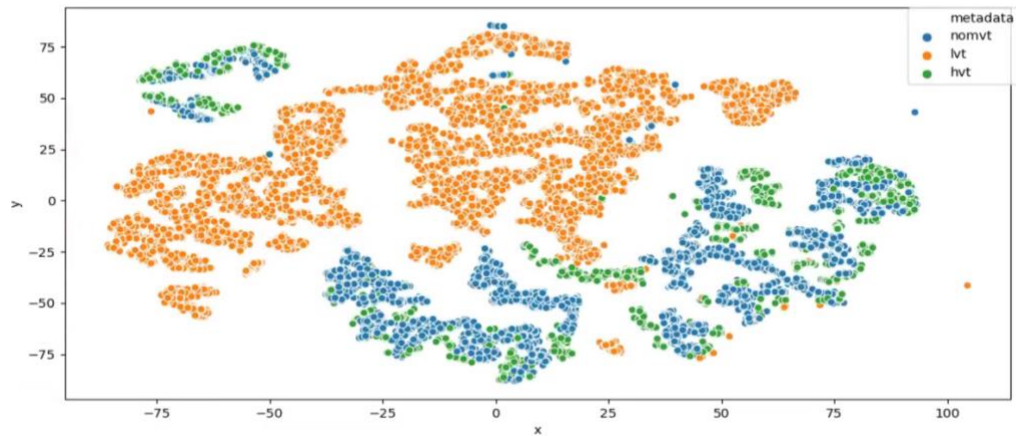


Figure 4: HVT refers to High Voltage Transistor, LVT to Low Voltage Transistor, and Nom to Normal Voltage Transistor

4.1 Encoding Flow

The training set is then fed into our variational-autoencoder (VAE). After the VAE is trained, all the training data points can be encoded. We then feed the encoded data points into a downstream task and train the downstream task's model. The downstream task here is to predict standard properties of a transistor using the Square Law. After the downstream task model has been trained using the encoded version of the original training dataset, it can then be used to predict the ibias of the test dataset. In parallel to this flow, there is another downstream task model that is trained off the raw, non-encoded, training data and is used to predict the ibias of the test dataset. Therefore, we can compare the test loss of these two separate downstream tasks and see whether the encodings improve the efficacy of the downstream model. A natural question may be why the raw dataset should go through a slightly different model architecture than the encoded dataset. This is because the dimensionality of the data entering the downstream model is different. Since neural networks use matrix multiplications to function, the number of parameters in the two models cannot be the same since the inner dimension of matrix multiplication must match. However, there is no way around this and overall, this doesn't impact experiment results. The overall flow can be seen in Figure 6 below.

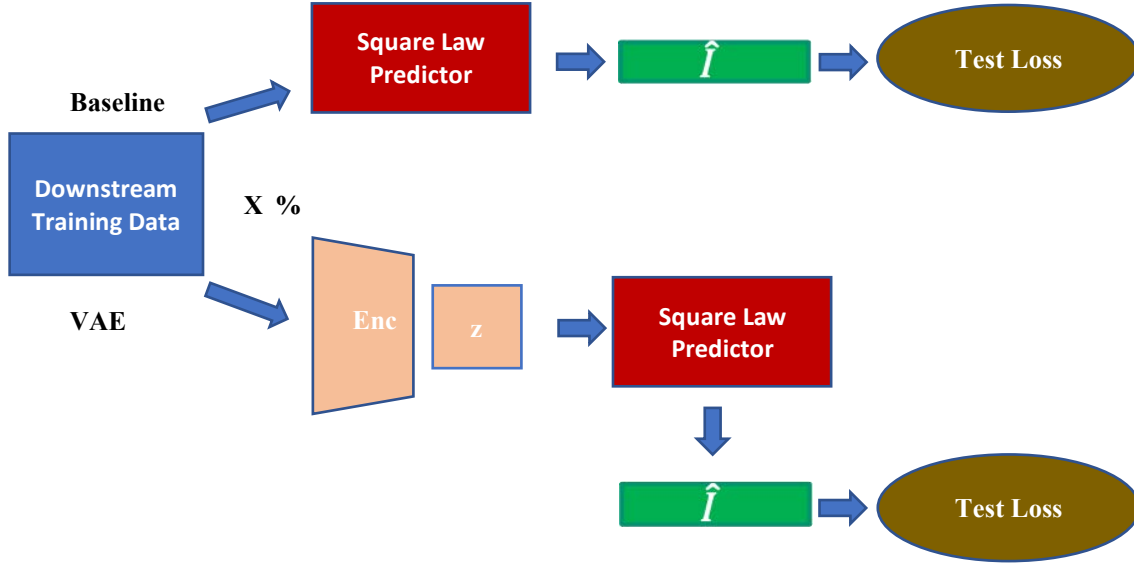


Figure 6

4.2 Training Process

The training process for the VAE was very standard and we performed normal hyperparameter optimization such as adjusting the regularization parameters for the VAE and varying batch sizes. The architecture of the VAE was discussed earlier, but the architecture of the downstream task is also straightforward. The downstream task is a standard fully connected network, with one hidden layer with size 200, and an input layer whose size depends on whether the input is the embeddings or the raw dataset. If embeddings are the input then this layer's dimension is 16, 32, and 64 but if it is the raw dataset then its dimension will be 11^3 . As a confirmation of understanding it should not be unexpected that the dimensionality of output of the downstream network is $[N \times 11 \times 11 \times 11]$ as the dimension of a single datapoint in the original dataset is $[N \times 10 \times 11 \times 11 \times 11]$ of which I_{bias} is one of the 10 small signal parameters so it should be $[N \times 1 \times 11 \times 11 \times 11]$.

The downstream task despite being a simple neural network is complex in that its training loop seeks to optimize the loss of the Square Law which allows for accurate estimates of a transistors I_{bias} . The Square Law has the following equation steps:

1. $\log isat = \log 0.5 + \log K_p + \log(1 + V_{ds} * \lambda) + \log \eta * threshold(V_{gs} - V_t, \epsilon)$
2. $\log ilin = \log K_p + \log(1 + V_{ds} * \lambda) + threshold((V_{gs} - V_t) * V_{ds} - 0.5 * V_{ds}^2, \epsilon)$
3. $\overline{I_{bias}} = e^{\log isat} + e^{\log ilin}$

The downstream task is explicitly learning the following 4 parameters $\{\log \lambda, \log \eta, \log V_t, \log K_p\}$ which can be used to estimate I_{bias} . The loss is then computed by finding the square loss between the *actual* I_{bias} and the estimated I_{bias} .

However, we *did* have some trouble getting the downstream networks to converge to a global minimum, so we helped the convergence process by initializing the first few steps of gradient descent so that the training of the downstream network can fine-tune and find the optimal values.

Chapter 5

Use Cases

5.1 Circuit Transfer

The primary benefit of the technology developed in this project is the ability for better prediction of circuit characteristics when adopting new manufacturing process technologies. Today, when new transistors are developed very little of the existing tools and knowledge can be leveraged.

5.2 Circuit Characterization

When circuits are designed, consisting of multiple transistors, we still rely mainly on human intuition to predict the behavior of these circuits - such as the gain or the current consumption. However, in a world where we can map individual circuit elements to embeddings, it becomes possible to predict the circuit characteristics, which improves circuit design iteration speed tremendously. Therefore, this technology has large potential for assistive debugging. Moreover, in scenarios where the circuit behavior does not map to the designer's intuition, these machine learning based technologies also aid with circuit comprehension.

5.3 Data and Computation Limitations

When datasets are forced to be size limited (e.g., due to storage constraints), embeddings are useful as a summarized form of the original dataset. Moreover, computations on the embeddings-based data representations are much less taxing than on the original datasets. For example any model which uses the raw data from BOMB versus embeddings will incur a penalty of $10 \cdot 11^3 / (\text{embedding size})$ per operation if a fully connected network was a consumer of this data - this ends up being around a factor of 1000 more expensive in computation and memory overhead. The exact overhead depends on the dimensionality of the original data; but as the example above shows, in the circuit domain this impact is huge.

Chapter 6

Implementation

6.1 Model Implementation

In this project, there are some novel implementations of machine learning code to integrate with the BOMB dataset.

The first is our implementation of being able to read the BOMB dataset into a format compatible with our VAE model. The PyTorch code used to achieve this is shown in the figure below [12].

```
class MosCharData(Dataset):  
  
    def __init__(self, data_path: str, last_idx: int = -3, stats=None):  
        # last_idx = 2 means vgs, vds and last_idx = 3 means vbs, vgs, vds  
        self.last_idx = last_idx  
  
        self.computed_mean, self.computed_std = None, None  
        self.sim_data = SimData.load(data_path)  
        # self.ss_params = self.sim_data.get_ss_params()  
        self.dataset_norm, self.dataset = self.process_simdata()  
  
        if stats is not None:  
            self.dataset_norm = (self.dataset - stats[0]) / stats[1]  
  
        self._vgs = torch.tensor(self.sim_data['vgs'])  
        self._vbs = torch.tensor(self.sim_data['vbs'])  
        self._vds = torch.tensor(self.sim_data['vds'])
```

Figure 5: Loading Data from the BOMB Open Dataset

The second is our implementation of the forward pass loop of the downstream task which predicts Ibias from the embeddings or the raw data.

```

def forward(self, x_in):
    x_in = torch.zeros_like(x_in)
    output = self.nn(x_in)
    output_dict = dict(zip(self.output_inits.keys(), output.T))
    output_dict = {k: v.unsqueeze(-1) for k, v in output_dict.items()}
    vgs_list = self.vgs_list.to(output)[None]
    vds_list = self.vds_list.to(output)

    ibias_hat = torch.zeros(x_in.shape[0], vgs_list.shape[-1],
len(vds_list)).to(output)

    threshold_fn = nn.Threshold(self.eps, self.eps)
    for vds_idx, vds in enumerate(vds_list):
        vov = vgs_list - output_dict['log_vt'].exp() #(Nx11)

        log_isat = torch.log(torch.tensor(0.5)) + output_dict['log_kp']
        #(Nx1)
        log_isat = log_isat + (1 + output_dict['log_lambda'].exp() *
vds).log() #(Nx11)
        log_isat = log_isat + output_dict['log_eta'].exp() *
threshold_fn(vov).log()
        isat = log_isat.exp() * (vov > 0) * (vds > vov)

        log_ilin = output_dict['log_kp'] + (1 +
output_dict['log_lambda'].exp() * vds).log() #(Nx1)
        log_ilin = log_ilin + threshold_fn(vov * vds - vds ** 2 / 2).log()
        #(Nx11)
        ilin = log_ilin.exp() * (vov > 0) * (vds <= vov)

        ibias_hat[..., vds_idx] = ilin + isat

    return ibias_hat

```

Figure 6: Computing the loss for a given loop iteration. Note that this is the code implementation of the formula described earlier.

Chapter 7

Evaluation

7.1 Model Evaluation

As described earlier, our objective is to show that embedding-based downstream learning has better performance than non-embedding based downstream learning. After training our models and performing some test trials we found that our embedding based models *did* perform better than the raw data. By looking at Figure 8 below, this is apparent.



Figure 7: Comparing test performance of embedding models vs the baseline

As shown in the chart, the latent values have lower test error across all the various scenarios upon which we ran the downstream data. The reason we tried with different sized downstream training sets was to investigate how summative the embeddings could be. Interestingly, the difference between the performance of the embedding based models and the baseline models stays consistent on almost all the variations of size in the downstream training set.

A number which may stand out is that the test loss seems incredibly low, but it's important to calibrate the test loss in the chart with what the maximum possible loss is in this predictive task. The maximum error in this predictive task is around $1e-6$, so the percentage errors that are obtained are reasonable ~10% error or less.

Chapter 8

Future Work

8.1 Model Evaluation

In the future, we want to expand to testing performance and generalizability across different technologies as the BOMB dataset matures, as well as discovering and benchmarking other useful downstream applications. There are other circuit tasks and characteristics which can be interesting downstream tasks such as training on one technology and testing on an explicitly different technology. If embeddings proved to be useful in this technology transfer process, that would be an immense contribution to the state of the art.

8.2 Model Training

Moreover, there can be improvements in how we train the embeddings. Currently this happens implicitly through manually tuning hyperparameters for a model which performs well on the downstream task. However, we could automatically tune the embeddings by including the downstream task in the loss function for the VAE as a form of regularization. This is a well-established method to improve model performance as seen in GANs [13]. There is also opportunity for further work in creating more complex network architectures to generate the embeddings that still use the general VAE loss function. One weakness of our encoding model is that we just stack the complex portion of the circuit dataset and extract their real values, but further improvement could be made if the complex portion of the circuits was maintained. Moreover, in the network architecture we did not use convolutions or any method to link the real and imaginary portions of the complex number once we separated them, which seems like a great opportunity to improve the encoding process.

Chapter 9

Conclusion

This project sought to prove that generating circuit embeddings using deep learning technologies was not only tenable but also *useful*. We have shown promise in both the BOMB dataset and the ability for machine learning to be used in the circuit domain. We have paved the way for more research to determine the extent to which technology transfer and circuit characterization is possible from embeddings generated from the BOMB dataset.

Bibliography

- [1] K. B. M. S. B. J. B. Bruce A. Draper, "Recognizing faces with PCA and ICA," *Computer Vision and Image Understanding*, Vols. Volume 91, Issues 1–2, no. ISSN 1077-3142, pp. Pages 115-137, 2003.
- [2] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, p. 2579–2605, 2008.
- [3] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, Red Hook, NY, USA, 2012.
- [4] J. Jordan, "Variational Auto-Encoders".
- [5] R. Lageweg, "Berkeley Open MOS dataBase (BOMB): A Dataset for Silicon Technology Representation Learning," BWRC Research Lab, Berkeley, 2021.
- [6] D. P. Kingma and M. Welling, "An Introduction to Variational Autoencoders," *CoRR*, vol. abs/1906.02691, 2019.
- [7] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, 2014.
- [8] J. Pennington, R. Socher and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.
- [9] M. Z. a. R. G. a. M. G. a. M. F. Mushtaq, "Efficient processing of GRU based on word embedding for text classification," 2019.
- [10] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams and A. Aspuru-Guzik, "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules," *ACS Central Science*, vol. 4, p. 268–276, January 2018.
- [11] Y. Bengio, A. Courville and P. Vincent, *Representation Learning: A Review and New Perspectives*, 2014.
- [12] Y. Zhang and Z. Lu, "Exploring Semi-supervised Variational Autoencoders for Biomedical Relation Extraction," *CoRR*, vol. abs/1901.06103, 2019.
- [13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, 2019.
- [14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, *Generative Adversarial Networks*, 2014.

- [15] M. Habibi, L. Weber, M. Neves, D. L. Wiegandt and U. Leser, "Deep learning with word embeddings improves biomedical named entity recognition," *Bioinformatics*, vol. 33, pp. i37-i48, July 2017.