Learning Rate Estimation for Stochastic Gradient Descent

Nadia Hyder
Gerald Friedland

Learning Rate Estimation for Stochastic Gradient Descent

by Nadia Hyder

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:

Gerald Friedland

Professor Gerald Friedland
Research Advisor

May 19th, 2022
(Date)

* * * * * * *

[illegible]

Professor Kannan Ramchandran
Second Reader

May 19, 2022
(Date)

Learning Rate Estimation for Stochastic Gradient Descent

by

Nadia Hyder

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Spring 2022

Learning Rate Estimation for Stochastic Gradient Descent

Abstract

Learning Rate Estimation for Stochastic Gradient Descent

by

Nadia Hyder

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Gerald Friedland, Chair

State-of-the-art gradient descent optimizers all attempt to tune learning rate such that we can find the minimum of the loss function without overshooting or approaching it so slowly that we fail to reach it by the end of training. Yet, current approaches fail to consider what the shape of the error function means. In this work, we conduct experiments to better understand complexity of error functions and develop systematic methods of measuring learning rate using concepts from information theory and fractal geometry. Experiments conducted suggest a few findings: (1) resulting loss curves from training over random, unlearnable data resemble exponential decay, (2) oversized networks are less sensitive to hyperparameters, and (3) fractal dimension can be a useful heuristic for learning rate scaling. Together, these 3 findings solidify that the underlying complexity of the learning problem should be accounted for when measuring– rather than selecting– learning rate.

Contents

List of Figures

List of Tables

Acknowledgments

I am so grateful to my friends, family, and UC Berkeley faculty who have inspired, supported, and encouraged me in my journey to pursuing and completing my masters degree. I would like to thank Professor Gerald Friedland for his wisdom, his guidance throughout my masters, and most importantly, for being the reason I decided to pursue this degree. If I had not taken his course 'Experimental Design for Machine Learning' two years ago, I would not be here today. I would like to thank Professor David Bamman as well, who gave me my first research opportunity at UC Berkeley. I would also like to thank Professor Kannan Ramchandran, my second reader. Finally, I would like to thank my family for their unconditional love, support, and encouragement. To my parents, thank you for always believing in me, guiding me, and encouraging me to pursue my passions and put my best foot forward. To my siblings, thank you for being such positive forces in my life and for always being there for me.

Chapter 1

Introduction

Traditional neural networks are trained using stochastic gradient descent, an iterative optimization technique that computes an expected gradient using minibatches of data. Network weights are computed at each timestep: $\theta_{t+1} = \theta_t - \eta \nabla_i \mathcal{L}(\theta_t)$, where η is learning rate and $\mathcal{L}$ is the loss function. Learning rate controls how quickly the model is adapted to the problem and moves toward a minimum of the loss landscape. Smaller learning rates mean slower convergence to a minimum (necessitating longer training) and possibly getting stuck, while larger learning rates may result in rapid changes and possibly overshooting minima or converging to a suboptimal solution (with shorter training).

As a result, the choice of learning rate can have a large effect on model performance and so new methods of learning rate selection, adjustment, and adaptation have emerged. These include using exponentially weighted averages of previous weight updates (momentum), varying learning rate over iterations using learning rate scheduling, and adapting learning rate based on performance.

Existing methods mostly treat learning rate as a hyperparameter. By definition, hyperparameters are values manually selected to control the learning process whereas parameters are derived from the learning process. Ideally, learning rate should be a parameter– not a hyperparameter– measured and adjusted accordingly during training. Machine learning is a scientific process and thus requires measurements beyond accuracy, along with parameter configuration motivated by these measurements.

In this work, we (1) develop better intuition about error functions of data, and (2) take an information theoretical and measurement-based approach to configure learning rate using the complexity of the error function. We analyze loss curves and their complexity versus accuracy, observe how network capacity affects the choice of learning rate, and tie these ideas back to the fractal geometry of error functions over various learning rates and network capacities. With these considerations, we propose a method of learning rate scaling using fractal dimension.

Chapter 2

Related Work

Existing work related to learning rate selection is motivated by the following issue: the magnitude of gradients can change drastically between iterations as parameters change. This makes it both difficult and suboptimal to choose a single, global learning rate. Approaches to address this include learning rate scheduling and various gradient descent optimization algorithms. Optimizers like Adagrad, Adadelta, RMSprop, and Adam implement adaptive learning rates using momentum and/or scaling. Fractal geometry has been touched upon, with current research showing promising results and room for further work.

2.1 Learning Rate Scheduling

Learning rate scheduling is a simple, yet common approach. There are several existing predefined frameworks in which the learning rate is adjusted between iterations as training progresses.

Time-based decay

Time-based decay takes on the following form:

$$\eta = \frac{\eta_0}{(1 + kt)}$$

where η and k are hyperparameters and t is the current iteration number.

Exponential decay

Another common schedule is exponential decay, computed using the following equation:

$$\eta = \eta_0 * e^{(-kt)}$$

Cosine Annealing

This learning rate schedule begins with a large learning rate and rapidly decreases it to a minimum value before increasing it rapidly, then repeats this process [9]. The learning rate is reset to mimic restarting the learning process, but better-performing weights are maintained so as to give the learning process a “warm restart”.

$$\eta_t = \eta^i_{min} + \frac{1}{2}(\eta^i_{max} - \eta^i_{min})(1 + cos(\frac{T_{cur}}{T_i}\pi))$$

where η^i_{min} and η^i_{max} are the minimum and maximum learning rates, and T_{cur} is the current iteration number since the last restart.

Learning-Rate Annealing Methods for Deep Neural Networks

Nakamura et al. [11] discuss various different approaches to scheduled-annealing for SGD and experiment with deep neural networks on image classification datasets. Nakamura et al. propose an annealing approach combined with the sigmoid function with warmup, and argue that this approach overtakes both adaptive methods and existing schedules in terms of accuracy.

The primary shortcoming associated with learning rate schedules is their dependence on pre-defined hyperparameters, the data, and the type of model used.

2.2 Gradient Descent Optimization

Stochastic Gradient Descent with momentum

SGD with momentum (SGDM) [12] is an optimization method which adds a momentum term to regular SGD. Momentum is a moving average of gradients used to update network weights. This approach works better and faster than regular SGD as it helps accelerate gradient vectors in the right direction for faster convergence. In SGDM, parameters are updated as follows:

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta)$$

$$\theta_t = \theta_{t-1} - v_t$$

where γ is another hyperparameter that needs to be configured and typically has a value of 0.9.

Adagrad

Adagrad [4] is a gradient descent optimization algorithm in which each parameter has its own learning rate. The optimizer automatically adapts learning rates in each dimension based on gradients seen, assigning high learning rates for parameters related to infrequent features and low learning rates for frequent ones. Overall, Adagrad modifies the general learning rate at time step t for every parameter θ, based on its past gradients.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i}$$

Since Adagrad performs smaller updates, it is well-suited for sparse data. A major drawback of using Adagrad is that it accumulates squared gradients in the denominator. Since every added term is positive, the accumulated sum continues to grow during training, potentially diminishing the learning rate until it is infinitesimal.

Adadelta

Adadelta [18] is an extension of Adagrad that reduces the effects of accumulation in the denominator. Instead of accumulating all past gradients, Adadelta restricts the window of accumulated gradients to a fixed size. The sum of gradients is instead a decaying average of all past squared gradients, and the running average $E[g^2]_t$ at time step t is computed using the running average at the previous time step and the current gradient:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

γ is traditionally set to 0.9, and SGD updates as follows:

$$\Delta\theta_t = -\eta * g_{t,i}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

with the equation for Adadelta:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

where the denominator is equivalent to the root mean squared error criterion of the gradient. Adadelta provides the advantage of not needing to set a default learning rate while also eliminating the major drawback of diminishing learning rate associated with Adagrad.

RMSprop

RMSprop [7] is a similar adaptive learning rate method also developed as a way to resolve Adagrad's diminishing learning rates. RMSprop has an identical update rule.

$$E[g^2]_t = 0.9 E[g^2]_{t-1} + 0.1 g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Adam

Adam (Adaptive Moment Estimation) [8] utilizes both momentum and scaling, combining methodologies from SGD with momentum and RMSprop. This optimizer works well for data with noisy and sparse gradients. The weight update is performed using the equation:

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

and η is the learning rate (1e-3 in the original paper), ϵ is a small number like 1e-10 to prevent division by 0 and β_1 and β_2 are forgetting parameters typically set to 0.9 and 0.99, respectively.

Hypergradient descent

Baydin et al. [2] discuss a method for improving gradient descent optimizers by dynamically updating learning rate during training. This is achieved by updating the gradient with respect to the learning rate of the update itself, which they define as a "hypergradient." The update rule is the following:

$$\theta_t = \theta_{t-1} - \eta_t \Delta f(\theta_{t-1})$$

and the learning rate is computed using the formula:

$$\eta_t = \eta_{t-1} + \beta \Delta f(\theta_{t-1}) * \Delta f(\theta_{t-2})$$

where β is the hypergradient learning rate. This algorithm requires setting both η_0 and β.

2.3 Learning Rates and Fractal Geometry

Learning Figures with the Hausdorff Metric by Fractals

Sugiyama et al. [16] present a method of learning figures, defined as "nonempty compact sets in Euclidean space" in which a machine learner takes as input discretized vectors and outputs discrete representations of the target figure as self-similar sets, or fractals. The generalization error of outputs is measured with the Hausdorff metric (fractal dimension). Sugiyama et al. argue there is a connection between fractal geometry and learning by measuring complexity using the Hausdorff and VC dimensions, which give a lower bound on the number of positive examples.

The Fractal Geometry of Backpropagation

Rojas [13] highlights the shapes of iteration paths in linear associator training using back-propagation with momentum. Rojas shows that regardless of the learning rate selected, structure of error functions (or the "iteration path for on-line learning") is a fractal.

Acceleration via Fractal Learning Rate Schedules

Agarwal et al. [1] discuss a Chebyshev learning rate scheduler for gradient descent in which step sizes are ordered fractally. They claim that a step-size schedule derived from Chebyshev polynomial roots gives unstable optimization trajectories, but when a fractal permutation of Chebyshev step sizes is selected, stable acceleration is achieved.

Chapter 3

Conceptual Background

3.1 Stochastic Gradient Descent

Stochastic Gradient Descent is the standard optimization technique for finding network parameters that minimize loss. Gradients of the loss function $\mathcal{L}$ are computed over minibatches of data at each iteration using backpropagation until the algorithm converges.

$$\theta_{t+1} = \theta_t - \eta \nabla_i \mathcal{L}(\theta_t)$$

θ_t represents the network parameters (weights) at timestep t, i refers to the minibatch sample, and η is learning rate. The learning rate is the step size in the loss curve and determines how quickly the optimization algorithm moves towards a minimum of the loss function.

3.2 Information Theory

Information theory provides a powerful framework for measurement in machine learning.

$\vec{x}$	$f(\vec{x})$
$\vec{x_1}$	0
$\vec{x_2}$	1
$\vec{x_3}$	1
...	...
$\vec{x_n}$	1

Table 3.1: Table containing input to a machine learner where $\vec{x_i}$ is an instance containing input features and $f(\vec{x_i})$ is the corresponding output.

In the table above, each outcome under $f(\vec{x})$ is one bit of information if and only if the learner is a binary classifier and outcomes are equiprobable. If there are more than two classes or outcomes are non-uniform, the amount of information in the outcome is adjusted according to Shannon entropy [15],

$$H = -\Sigma\, p(x)\, \log p(x).$$

Supervised machine learners have a memory equivalent capacity which can be measured and computed in bits. Below we discuss how to measure this capacity and determine architectural requirements for a good learner.

Memory-Equivalent Capacity

Intellectual capacity refers to the number of unique target functions a machine learner is able to represent, as a function of model parameters [10]. When a machine learner can represent all 2^N binary labelling functions of any N inputs, its intellectual capacity is memory-equivalent to N bits. This is defined as memory-equivalent capacity (MEC). Given uniformly random data points, MEC is equal to VC dimension [6]. VC dimension is a well-known measure of network capacity, defined as the cardinality of the largest set of points the learner can shatter (the largest set of points for which there exists a perfect classifier f for at least one configuration of the data points) [17].

The capacity of a dataset has two bounds [5]:

1. Capacity Requirement (upper bound), or the size of the dataset's lookup table,
2. Expected Capacity Requirement (lower bound), achieved when weights are set to identity and biases are learned, which in the best case is logarithmic.

Meanwhile, artificial neural networks have capacity upper limits that can be determined analytically using the following engineering principles [6]:

1. The output of a perceptron is maximally 1 bit.
2. Maximum memory capacity of a perceptron is the number of parameters (including bias) in bits.
3. Maximum memory capacity of perceptrons in parallel is additive.
4. Maximum memory capacity of a layer of perceptrons depending on a previous layer of perceptrons is limited by the maximum output (in bits) of the previous layer.

Measuring capacity allows us to determine optimal network architecture, including number of neurons and layers.

3.3 Fractal Geometry

Fractals are non-geometric, infinitely complex mathematical shapes in which patterns repeat (self-similar). A self-similar figure can be split into parts, each of which is a scaled copy of the whole. The study of fractals, or fractal geometry, provides a way of understanding complexity in not just shapes, but also in systems.

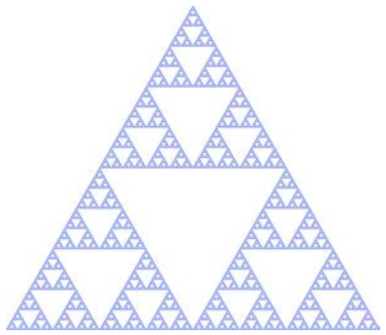


Figure 3.1: Sierpinsky triangle, a well-known fractal

Fractal Dimension

Fractal dimension measures the complexity of a self-similar figure; more specifically, it is a ratio measuring how a fractal pattern changes with the scale it is measured at.
Simple figures provide better intuition of fractals and their dimensions.

1. A line segment of length n can be split into n^1 smaller lines when each is $\frac{1}{n}$ the length of the original (here, the scaling factor is n). The fractal dimension of a line is 1.

2. An $n \times n$ square can be split into n^2 unit squares. Each square has a magnification factor of n, as scaling it by n gives back the size of the original square. The fractal dimension of a square is 2.

3. An $n \times n \times n$ cube can be split into n^3 unit cubes. Each cube has a scaling factor of n to generate the original cube. The fractal dimension of a cube is 3.

The following formulas are used in fractal geometry, where N is the number of sub-pieces in a fractal, S is the scaling factor, and D is the dimension of the figure.

$$N = S^D$$

$$D = \frac{logN}{logS}$$

Looking at our equations for fractal dimension computation, we can see that there exists a relationship between MEC and fractal dimension. N bits of MEC is equivalent to a fractal dimension in which S = 2.

Minkowski Dimension

The Minkowski dimension, or box-counting dimension, is a method of determining the fractal dimension of a set in R^n. Formally, the Minkowski dimension of a set is defined as follows: For set $F \subset R^d$ and $\delta > 0$, $N_\delta(F)$ denotes a collection containing the smallest number of closed balls with diameter less than or equal to δ which cover F. The upper limit of the Minkowski dimension of F is:

$$\dim_m F = \limsup_{\delta \to 0} \left[\frac{\log\left(|N_\delta(F)|\right)}{\log\left(\frac{1}{\delta}\right)} \right]$$

Fractal dimension measures how a pattern fills space– the dimensionality of the pattern need not be the same as the dimension of the space it exists in. One well-known application of fractal dimension is measuring the complexity and irregularity of coastlines. This emerged from the observation that the scale at which a coastline is measured affects the resulting measurement. The fractal dimension of a coastline is a value between 1 and 2, which corresponds to the range between a line that is so straight and a line that is so irregular it completely fills up all of 2D space. The higher the fractal dimension, the more complex and irregular the figure is, whereas the lower the fractal dimension, the smoother it is. These same principles hold for fractal dimensions of error functions.

3.4 Main Hypotheses

The complexity of the original problem (3.1) is reflected in the complexity of the model we must design (MEC, 3.2) and this complexity should be reflected in the observable loss function (Minkowski dimension, 3.3). Complexity is assumed to never go up or down without loss, according to the Law of Conservation of Complexity [14]. For this reason, we speculate that the fractal dimension of the error curve tells us how learning rate should be scaled. Highly complex functions (as reflected by the error curve complexity) should be approached slowly with a small learning rate, less complex functions can have higher learning rates.

From this theory, we formulated the following hypotheses:

1. The error function reflects the complexity of the function implied by the data.
2. The learning rate should therefore by adapted according to the complexity of the function modeled.
3. Fractal dimension measures said complexity.

Chapter 4

Experimental Setup

To test our hypotheses, we ran a series of experiments to better understand what fractal dimension tells us about the error function, the effect of network capacity, and the effect of varying both network capacity and fractal dimension on performance. In this section, we discuss these experiments and their setup.

4.1 List of Experiments

1. Examining loss curves of random data
2. Examining the effect of learning rate when network capacity is varied
3. Measuring and adapting learning rate based on fractal dimension of the error function and analyzing performance compared to other optimizers

4.2 Classifier Setup

For each of the experiments, we used a simple binary classifier implemented in PyTorch and cross-entropy loss to compute error. The following is the model implementation:

```
class Classifier(torch.nn.Module):

    def __init__(self, input_size, hidden_size):
        super(Classifier, self).__init__()
        self.layer_1 = torch.nn.Linear(input_size, hidden_size)
        self.layer_out = torch.nn.Linear(hidden_size, 1)
        self.sigmoid = torch.nn.Sigmoid()

    def forward(self, inputs):
        x = self.sigmoid(self.layer_1(inputs))
        x = self.layer_out(x)
        return x

    def predict(self,x):
        pred = self.forward(x)
        return (pred.detach().numpy() > 0)
```

For each experiment, we trained with the following learning rates: 10, 5, 3, 1, 0.1, 0.01, 0.001. Optimizer selection and number of iterations varied by experiment.

4.3 Datasets

We ran our experiments on several datasets including generated data that took on the following shapes/ forms: random data, circle data, and spiral data. Finally, we ran our experiments on MNIST data– from which we arbitrarily selected 2 classes to perform binary classification– to see how our experiments performed on more widely utilized datasets. Results were approximately the same regardless of which two classes we selected from MNIST. All datasets used were balanced.

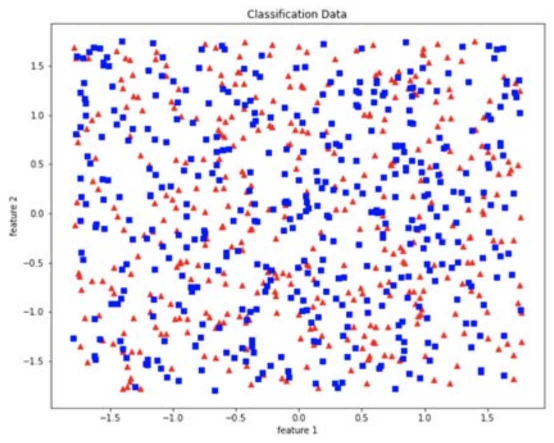


Figure 4.1: Randomly generated, 2 feature data

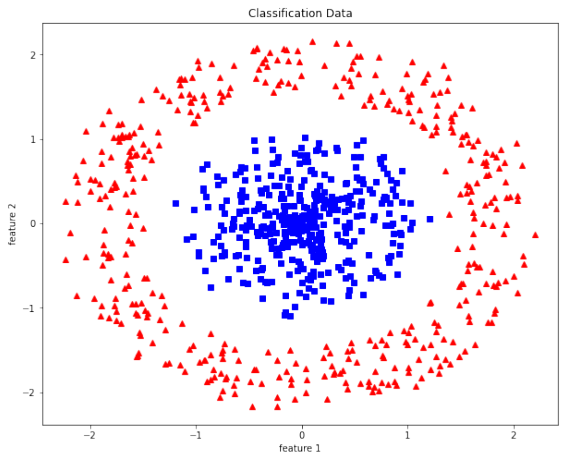


Figure 4.2: Circle data

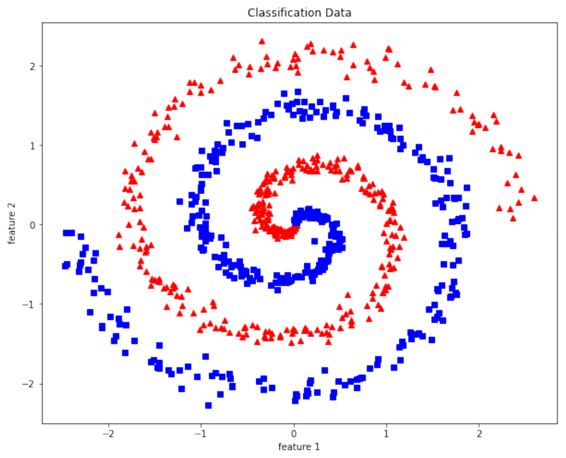


Figure 4.3: Spiral data

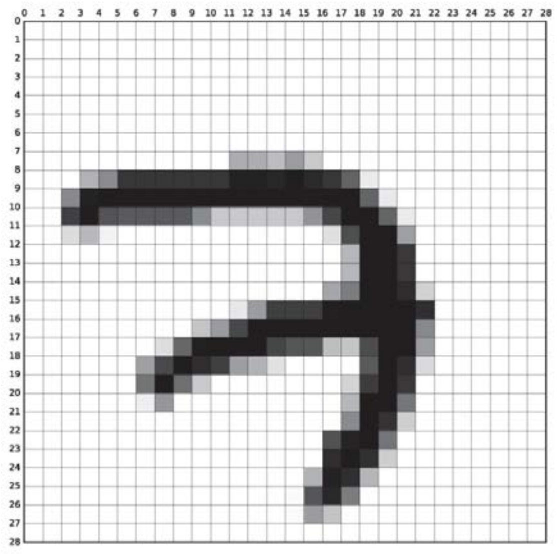


Figure 4.4: MNIST sample for digit 7. Contains 28x28 features, 1 for each pixel.

4.4 Measurement Tools

Network architecture was constructed carefully so as to either be at or above MEC (depending on the experiment). We determined the appropriate network capacity for each dataset using Brainome [3], a tool that computes capacity expectations for datasets using different machine learning algorithms. Using Brainome and the aforementioned engineering principles for neural networks, we determined appropriate network architecture (number of hidden layer neurons) to achieve the desired capacity.

4.5 Experiment 1: Randomness and Exponential Decay

To gain insight into complexity of error functions, especially from a poor learner on unlearnable data, we analyzed loss curves resulting from training on randomly generated data. We then compared these to exponential decay curves.

In this experiment, we generated uniform random data with 1000 data points, 10 features, and binary outcomes. We found the capacity expectations using Brainome and built a model with appropriate capacity of 102 bits using 10 features and 1 hidden layer with 2 neurons. We trained over 2000 iterations using stochastic gradient descent and learning rates 10, 5, 3, 1, 0.1, 0.01, 0.001. After training, we plotted (a) the training and test loss, and (b) the running average of the loss curve. We then fit an exponential decay curve to the running average curve, which took on the form $a * 2^{-bt} + c$. Finally, we measured the area between the two curves as a measure of similarity.

4.6 Experiment 2: Network Size and Learning Rate Sensitivity

State-of-the-art machine learners like GPT-3 have many layers and billions of parameters. Consequently, these networks are incredibly powerful. To better understand the effect of learning rate on overparametrized networks, we trained several different models of varying capacities over a binary class MNIST dataset.

We trained each network multiple times with learning rates 10, 5, 3, 1, 0.1, 0.01, 0.001. For each model, we tried different optimizers including stochastic gradient descent, RMSProp, and Adam. Our MNIST dataset has a memory equivalent capacity of 1573 bits (as determined by Brainome). We constructed networks with capacities of 1572 bits, 7860 bits (5x MEC), and 157200 bits (100x MEC) and analyzed how learning rate affected training for networks which were (a) at capacity, (b) larger than capacity, and (c) substantially larger than capacity.

4.7 Experiment 3: Fractal Dimension and Loss Curves

To compute the complexity of error functions, we implemented the box counting algorithm for fractal dimension. We then used this computation to perform learning rate scaling.

Box Counting Algorithm

In practice, the Minkowski dimension is computed using box counting over the data. Box counting is performed by overlaying a grid on the data, counting how many boxes of the grid are covering part of the data, and redoing this process with a finer grid with smaller boxes. The fractal dimension is the slope of the line when we plot the value of $\log_2 N$ on the y-axis against the value of $\log_2 R$ on the x-axis, where N is the number of boxes covering the curve and R is the magnification (inverse of box size). This gives the rate of change in complexity with scale.

Algorithm 1 Box counting algorithm

```
1: procedure BOXCOUNT($Z$)
2:     transform Z into binary array
3:     $p \leftarrow$ minimal dimension of data
4:     $n \leftarrow$ greatest power of $2 \geq p$
5:     build $boxsizes$ from $2^n$ to $2^1$
6:     $counts \leftarrow []$
7:     for size in $boxsizes$ do
8:         $currcount \leftarrow$ number of non-empty and non-full boxes in $Z$
9:         add $currcount$ to $counts$
10:    end for
11:    fit successive $log_2(boxsizes)$ with $log_2(counts)$ using OLS
12:    return slope of fitted linear regression line
13: end procedure
```

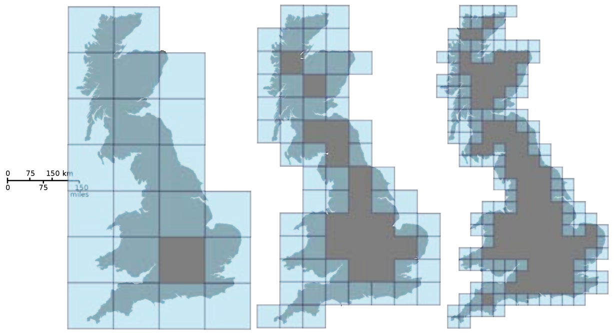


Figure 4.5: Boxes overlaid on the coast of Great Britain

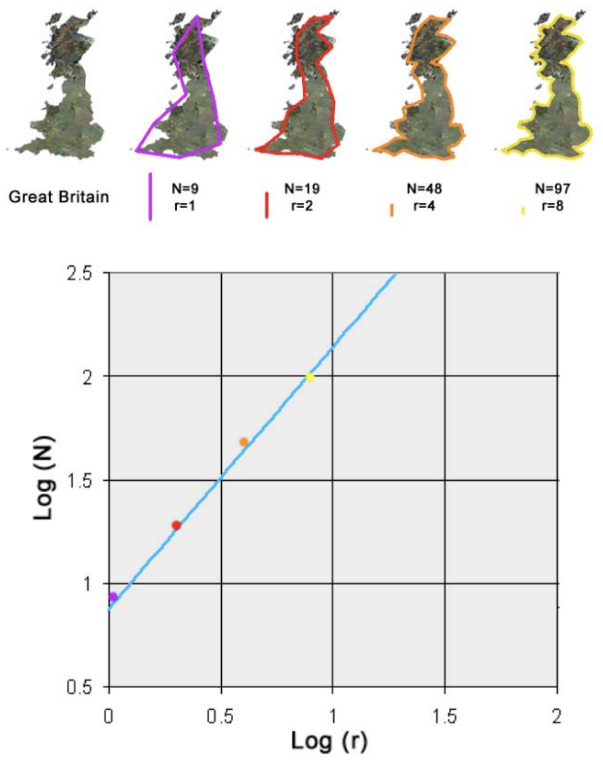


Figure 4.6: Computing the box counting dimension of the coast of Great Britain

Learning Rate Scaling using Fractal Dimension

The goal of this experiment was to develop an adaptive learning rate algorithm that uses fractal dimension of the error function. We drew on ideas from related work, particularly the concept of a "warm restart" [11]. Using the box counting algorithm and the same MNIST dataset, capacities, and learning rates described before, we developed an algorithm for learning rate scaling. At every iteration, the learning rate is scaled according to the computed Minkowski dimension. The closer the fractal dimension is to 1, the greater the increase in learning rate. This is to increase complexity in the error function. In this implementation, the learning rate begins with a specified initial value η_{min} and increases every iteration until it reaches a maximum learning rate η_{max}, after which the learning rate is reset to the initial minimum, and the process repeats. This differs from learning rate scheduling as learning rate is not adjusted by a constant value but instead by a factor of the fractal dimension. Learning rate scaling is performed with the following equations:

$$\Delta_{t-1} = \left\{ \begin{array}{ll} |1 - boxcount(Z)|, & \text{if } |1 - boxcount(Z)| \geq \delta \\ \delta, & \text{otherwise} \end{array} \right\} \tag{4.1}$$

$$\eta_t = \eta_{t-1}(1 + \frac{1}{T_n \Delta_{t-1}})$$

$$\text{if } \eta_{t-1} > \eta_{max} \text{ or } \eta_t > \eta_{max} : \eta_t := \eta_{min}$$

Where Z is data computed from our error function up until iteration t, $\delta = 0.01$ so η does not increase rapidly, and T_n is the total number of training iterations.

Algorithm 2 Fractal Dimension Learning Rate Scaling

```
1: procedure FRACTALSCALING($Z, \eta_{min} = \eta_0, \eta_{max} = 10, \delta = 0.01$)
2:     $fd \leftarrow boxcount(Z)$
3:     $\Delta \leftarrow |1 - fd|$
4:     if $\Delta \leq \delta$ then
5:         $\Delta \leftarrow \delta$
6:     end if
7:     $\eta_{new} = \eta * (1 + \frac{1}{T_n \Delta})$
8:     if $\eta > \eta_{max}$ or $\eta_{new} > \eta_{max}$ then
9:         $\eta \leftarrow \eta_{min}$
10:    else
11:        $\eta \leftarrow \eta_{new}$
12:    end if
13: end procedure
```

Additionally, we ran fractal scaling on the spiral dataset to see how it stacked up in performance against other optimizers.

Chapter 5

Results

We found the following results, which together verified our hypotheses.

1. Loss curves of random data resemble exponential decay.
2. The greater the network capacity, the less the learner is sensitive to learning rate.
3. Fractal dimension measures the complexity of the error function and can be used as a method of adapting the learning rate.

Below we discuss these findings in detail by experiment.

5.1 Exponential Decay in Loss Curves

When we trained our learner at MEC on random data, the classifier achieved near 50% accuracy on the test set as expected with each learning rate and SGD. For each learning rate, our moving average loss closely resembled exponential decay. Furthermore, the area between the moving average and fitted exponential decay curve was at maximum 10, meaning the two curves were quite similar.

We speculate that when we do not get exponential decay loss curves across learning rates, our data is learnable and likely just needs hyperparameter tuning or improvements in network architecture for better performance. In this case, on the other hand, our selected network architecture was appropriate but the data was unlearnable. It is worth noting that loss curves resembling exponential decay are quite smooth and have fractal dimension close to 1. This shows a strong relationship between the complexity of the error function and the complexity of the function implied by the data. The ‘function’ was simply random guessing and the smoothness of the error function reflected this lack of complexity.

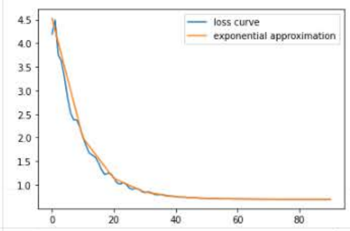
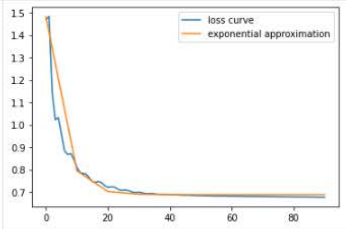
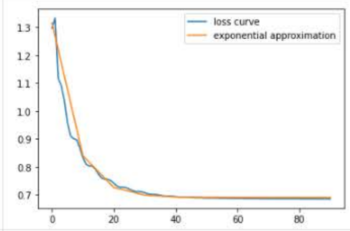
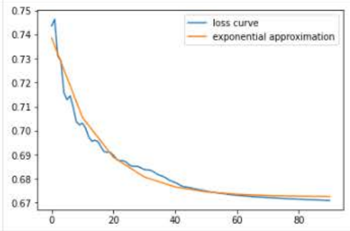
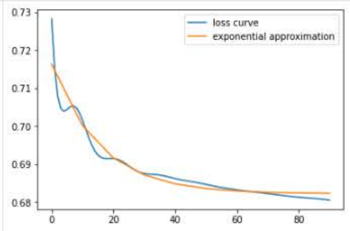
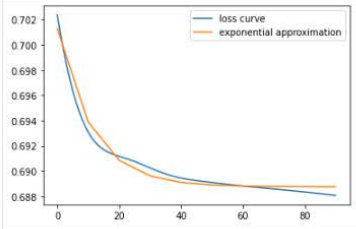
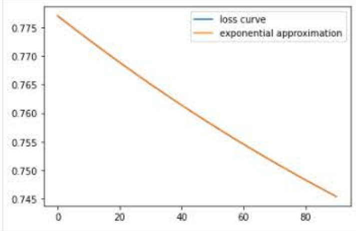
Learning Rate	running average of train loss and exponential decay curves	Exponential Decay function: A2^-kt	Area between	Train Accuracy	Test Accuracy
10		$3.831 * 2^{-0.156x} + 0.695$	10.238	49.88%	48.00%
5		$0.793 * 2^{-0.29x} + 0.688$	2.834	52.38%	52.00%
3		$0.626 * 2^{-0.207x} + 0.690$	1.883	55.75%	45.50%
1		$0.066 * 2^{-0.100x} + 0.673$	0.247	56.88%	52.50%
0.1		$0.034 * 2^{-0.093x} + 0.682$	0.155	57.13%	52.50%
0.01		$0.013 * 2^{-0.128x} + 0.689$	0.048	55.00%	52.00%
0.001		$0.096 * 2^{-0.006x} + 0.681$	0.005	50.22%	52.00%

Figure 5.1: Plot of loss curve + exponential decay approximation, fitted equation for decay, area between the curves, and accuracy for decreasing learning rates

5.2 Oversized Networks

When we varied network capacities using the MNIST dataset and stochastic gradient descent, we achieved the following accuracies:

MEC	$\eta = 10$	$\eta = 5$	$\eta = 3$	$\eta = 1$	$\eta = 0.1$	$\eta = 0.01$	$\eta = 0.001$
1572 bits	99.83%	99.83%	99.7%	99.76%	99.46%	54.03%	46.68%
7860 bits	99.86%	99.83%	99.83%	99.80%	99.59%	98.85%	46.72%
157200 bits	99.76%	99.59%	99.83%	99.63%	99.39%	98.95%	95.67%

Table 5.1: Test accuracy by network capacity and learning rate using SGD

MEC	mean	SD
1572 bits	85.61%	22.39
7860 bits	92.06%	18.52
157200 bits	98.97%	1.38

Table 5.2: Mean and standard deviation of accuracy by capacity using SGD

Learners performed well for a greater number of learning rates as their capacity increased. By increasing capacity from 1572 to 7860 bits, the learner increased its ability to learn at one more specified learning rate than the previous model (at learning rate 0.01). When we further increased the capacity to 157200 bits, the learner could learn across all learning rates used. This model clearly outperformed the other two. These outcomes make it clear that larger networks are less sensitive to learning rate.

To confirm this holds regardless of optimizer choice, we ran this experiment again using RMSprop and Adam and got similar results.

MEC	$\eta = 10$	$\eta = 5$	$\eta = 3$	$\eta = 1$	$\eta = 0.1$	$\eta = 0.01$	$\eta = 0.001$
1572 bits	98.65%	98.71%	98.99%	99.05%	99.09%	99.32%	48.77%
7860 bits	99.09%	98.95%	98.68%	99.02%	99.39%	99.29%	99.42%
157200 bits	99.22%	99.15%	99.15%	98.71%	99.19%	99.32%	99.49%

Table 5.3: Test accuracy by network capacity and learning rate using RMSprop

MEC	mean	SD
1572 bits	91.80%	17.57
7860 bits	99.12%	0.25
157200 bits	99.18%	0.22

Table 5.4: Mean and standard deviation of accuracy by capacity using RMSprop

MEC	$\eta = 10$	$\eta = 5$	$\eta = 3$	$\eta = 1$	$\eta = 0.1$	$\eta = 0.01$	$\eta = 0.001$
1572 bits	98.65%	98.78%	99.05%	99.12%	99.15%	99.39%	46.41%
7860 bits	99.19%	98.85%	99.36%	99.56%	99.36%	99.36%	99.39%
157200 bits	99.49%	99.46%	99.36%	99.46%	99.42%	99.32%	99.36%

Table 5.5: Test accuracy by network capacity and learning rate using Adam

MEC	mean	SD
1572 bits	91.51%	18.41
7860 bits	99.30%	0.21
157200 bits	99.41%	0.06

Table 5.6: Mean and standard deviation of accuracy by capacity using Adam

The two optimizers performed better than SGD at learning rate 0.01 but performed just as poorly at learning rate 0.001. We can still see a clear relationship between model capacity and learning rate sensitivity: the smaller the network, the more sensitive to learning rate, and the larger the network, the better its performance regardless of learning rate. These findings also show the shortcomings of current adaptive optimizers; though Adam and RMSprop adjust learning rate based on performance, initial selected learning rate can have a large effect on performance at MEC.

5.3 Learning Rate Scaling using Fractal Dimension

Fractal dimension of an error function can be approximately between 1 and 2, where higher fractal dimension means more irregularity and complexity and lower fractal dimension corresponds to smoother, simpler curves. Furthermore, as determined in experiments with random data, poor learners have more predictable and smooth loss curves (i.e. exponential decay) with fractal dimension close to 1. These are the motivating principles behind the algorithm for fractal scaling. The proposed algorithm adds complexity to error functions to match the complexity of the original problem.

MNIST Data

We first trained learners using fractal scaling on the MNIST dataset with the same aforementioned learning rates and varying capacities. The results are shown below:

MEC	$\eta_{min} = 10$	$\eta_{min} = 5$	$\eta_{min} = 3$	$\eta_{min} = 1$	$\eta_{min} = 0.1$	$\eta_{min} = 0.01$	$\eta_{min} = 0.001$
1572 bits	96.49%	97.03%	98.49%	98.10%	95.20%	95.34%	98.60%
7860 bits	97.06%	98.57%	97.85%	98.53%	95.45%	98.89%	98.67%
157200 bits	97.96%	98.75%	98.92%	98.49%	98.71%	99.28%	99.28%

Table 5.7: Test accuracy by network capacity and learning rate for MNIST data using fractal scaling

MEC	mean	SD
1572 bits	97.04%	1.32
7860 bits	97.86%	1.14
157200 bits	98.77%	0.42

Table 5.8: Mean and standard deviation of accuracy by capacity using fractal scaling

Fractal scaling gave significant improvements in model performance, namely for the 1572 bit capacity model with learning rate 0.001 and 0.01. We achieve, on average, much higher and more consistent accuracies than we did using vanilla stochastic gradient descent. Additionally, unlike RMSprop and Adam, the learner performed similarly across learning rates and network capacities. There are slight fluctuations, but these appear to be independent of learning rate.

Spiral Data

Next, we trained with fractal scaling over the spiral data at MEC over 3000 iterations to see how it performed compared to SGD, RMSprop and Adam. We added the following features to make our data more learnable: $sin(x)$ and $sin(y)$. The table below shows how performance compared over several learning rates and optimizers.

Optimizer	$\eta = 10$	$\eta = 5$	$\eta = 3$	$\eta = 1$	$\eta = 0.1$	$\eta = 0.01$	$\eta = 0.001$
SGD	61.50%	69.00%	61.50%	71.50%	59.50%	59.50%	45.50%
RMSprop	57.00%	45.40%	54.50%	57.00%	69.00%	70.50%	62.00%
Adam	45.50%	56.50%	58.50%	73.50%	73.50%	80.50%	68.50%
Fractal Scaling	79.50%	82.50%	83.50%	81.50%	83.00%	74.00%	79.00%

Table 5.9: Test accuracy by optimizer and learning rate for spiral data

Optimizer	mean	SD
SGD	61.14%	7.74
RMSprop	59.34%	8.05
Adam	65.21%	11.28
Fractal Scaling	80.42%	3.06

Table 5.10: Mean and standard deviation of accuracy by optimizer

Again, we see more consistent accuracies with our fractal scaling algorithm and performance did not depend on the choice of learning rate. Loss curves generated using fractal scaling versus other optimizers are notable as well.

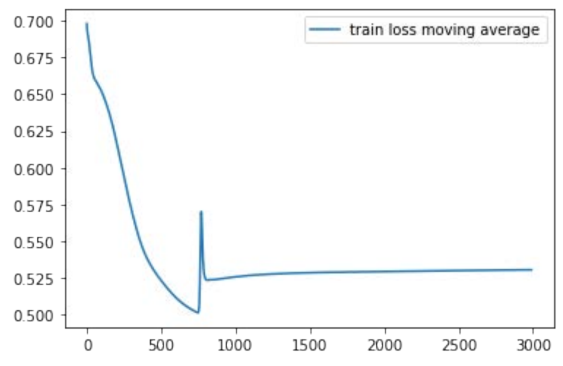


Figure 5.2: Loss curve using SGD and η = 3

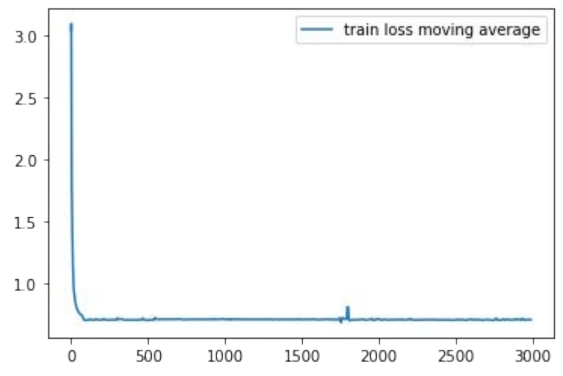


Figure 5.3: Loss curve using RMSprop and $\eta = 3$

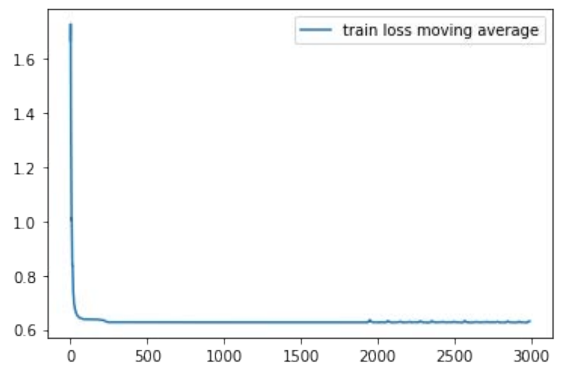


Figure 5.4: Loss curve using Adam and η = 3

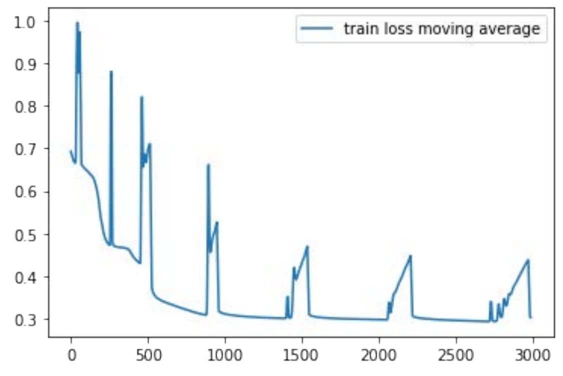


Figure 5.5: Loss curve using fractal scaling and $\eta = 3$

These loss curves have several important implications. Firstly, the loss curves corresponding to RMSprop and Adam for learning rate 3 both resemble exponential decay, and both achieve accuracy in the 50% range, only slightly better than random guessing. This illustrates our previous finding that exponential decay loss is indicative of a poor learner that is at best performing random guessing and needs hyperparameter tuning if the data is not random. Secondly, the curve corresponding to SGD is a little more varied and less smooth. This model has better accuracy than the other two (61.5%). Finally, the curve corresponding to fractal scaling shows exactly what we intended it to– it reflects the complexity we have added to the curve by accounting for fractal dimension. This proves empirically that by adding complexity to the loss function, fractal dimension scaling improves performance.

Chapter 6

Conclusion

In this work, we drew on concepts from information theory, fractal geometry, and related work to develop a methodical way of measuring and scaling learning rate. Our experiments showed the following results: (1) error functions resemble exponential decay when the learner performs random guessing, (2) oversized networks are less sensitive to learning rate, and (3) fractal dimension is a useful heuristic for learning rate scaling. Perhaps the most interesting finding is that empirically, fractal scaling outperforms SGD, RMSprop, and Adam as it is more consistent across learning rates and network capacity. These results ultimately give a clear takeaway: the error function reflects the complexity of the function implied by the data and for this reason complexity is a valuable measurement for learning rate computation.

6.1 Future Work

While we concluded that larger networks are less sensitive to learning rate, further explanation is needed. Particularly, how increasing network size changes loss landscape in such a way that gradient descent always converges to a minimum. Next, it could be helpful to find a better upper limit for fractal dimensions of error functions. Although in theory the fractal dimension of an error function can be between 1 and 2, in practice the curves seen in this work did not have fractal dimension greater than approximately 1.3. This could be a useful limiting measure in an optimizer which uses fractal dimension. It is also worth exploring whether fractal scaling performs better simply because it varies the learning rate or because we scale the learning rate appropriately, though our result on spiral data suggest that fractal scaling is a better method of adaptation. Another consideration is that the fractal scaling algorithm we developed uses η_{min} and η_{max} which adds two more hyperparameters to set. Although η_{min} corresponds to the initial specified learning rate (which had little effect) and η_{max} empirically performs well as 10, there may be other, better optimization methods using fractal dimension which do not need additional hyperparameters. Finally, because of the fluctuation in loss using fractal scaling, the learner may need to stop training at the right time or risk an increase in loss and decrease in accuracy. There are possible drawbacks to introducing complexity and fluctuation, which have not been observed in this work.

Bibliography

[1] Naman Agarwal, Surbhi Goel, and Cyril Zhang. "Acceleration via fractal learning rate schedules". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 87–99.

[2] Atilim Gunes Baydin et al. "Online Learning Rate Adaptation with Hypergradient Descent". In: *CoRR* abs/1703.04782 (2017). arXiv: `1703.04782`. URL: `http://arxiv.org/abs/1703.04782`.

[3] *Brainome - solve machine learning problems with Zero Code*. Feb. 2022. URL: `http://www.brainome.ai/`.

[4] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[5] Gerald Friedland. *Lecture notes in Experimental Design for Machine Learning on Multimedia Data*. Sept. 2020.

[6] Gerald Friedland, Alfredo Metere, and Mario Krell. *A Practical Approach to Sizing Neural Networks*. 2018. DOI: `10.48550/ARXIV.1810.02328`. URL: `https://arxiv.org/abs/1810.02328`.

[7] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on* 14.8 (2012), p. 2.

[8] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`.

[9] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Restarts". In: *CoRR* abs/1608.03983 (2016). arXiv: `1608.03983`. URL: `http://arxiv.org/abs/1608.03983`.

[10] David J. C. MacKay. *Information Theory, Inference Learning Algorithms*. USA: Cambridge University Press, 2002. ISBN: 0521642981.

[11] Kensuke Nakamura et al. "Learning-Rate Annealing Methods for Deep Neural Networks". In: *Electronics* 10.16 (2021). ISSN: 2079-9292. DOI: `10.3390/electronics10162029`. URL: `https://www.mdpi.com/2079-9292/10/16/2029`.

[12] Yurii E Nesterov. "A method for solving the convex programming problem with convergence rate O (1/kˆ 2)". In: *Dokl. akad. nauk Sssr.* Vol. 269. 1983, pp. 543–547.

[13] Rad Rojas. "The fractal geometry of backpropagation". In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. Vol. 1. IEEE. 1994, pp. 233–238.

[14] Dan Saffer. *Designing for interaction: creating innovative applications and devices.* New Riders, 2010.

[15] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: `10.1002/j.1538-7305.1948.tb01338.x`.

[16] Mahito Sugiyama et al. "Learning figures with the Hausdorff metric by fractals—towards computable binary classification". In: *Machine learning* 90.1 (2013), pp. 91–126.

[17] V. N. Vapnik and A. Ya. Chervonenkis. "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities". In: *Theory of Probability & Its Applications* 16.2 (1971), pp. 264–280. DOI: `10.1137/1116025`. eprint: `https://doi.org/10.1137/1116025`. URL: `https://doi.org/10.1137/1116025`.

[18] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *CoRR* abs/1212.5701 (2012). arXiv: `1212.5701`. URL: `http://arxiv.org/abs/1212.5701`.