When Noise Helps Learn: Approximate Impossibility in Identity Effects for Deep Neural Networks

Paul Glickman

A Thesis in
The Department of
Mathematics and Statistics

Presented in Partial Fulfilment of the Requirements
for the Degree of Master of Science (Mathematics) at
Concordia University
Montreal, Quebec, Canada

January 2025

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Paul Glickman

Entitled: When Noise Helps Generalize: Approximate Impossibility in Identity Effects for Deep Learning

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Mathematics)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

_____ Chair & Examiner

Dr. Junxi Zhang

_____ Examiner

Dr. Giuseppe Alessio D'Inverno

_____ Thesis Supervisor

Dr. Simone Brugiapaglia

Approved by _____

Dr. Lea Popovic, Graduate Program Director

_____

Dr. Pascale Sicotte, Dean of Faculty of Arts and Science

Date: December 17th, 2024

Abstract

Artificial intelligence, and especially deep learning, have had a major spotlight shined on them in recent years. Our goal is to mathematically and computationally explore some of its potential limits - specifically, those related to identity effects and generalization. Identity effects refer to the ability to recognize specific patterns, such as checking whether two sounds are identical, something human brains are very adept at handling from a very young age. It has been observed that depending on how the input data is "encoded", that is what type of vectors differentiate the pair of objects to be classified a deep neural net might fail to generalize identity effects outside the training set. Namely, given that matching objects are valid pairs, and non-matching objects are not, some types of encodings will allow the network to generalize to inputs it has never seen before, and some will not. Previous research shows that the existence of a specific orthogonal transformation on the encodings can predict impossibility of generalization. We explore mathematically and then test numerically whether approximate orthogonality leads to predictable loosening of the impossibility, eventually finding that approximate orthogonality leads to approximate impossibility. In particular, adding random noise to traditional encodings such as the "one-hot" encoding can help neural networks generalize outside the training set.

Acknowledgements

I would like to deeply thank my advisor Dr. Simone Brugliapaglia for all his knowledge, passion, and patience. This would not have been possible without him.

I would also like to express my sincere gratitude to Dr. Alessio D'Inverno and for his constructive criticism and he and Dr. Junxi Zhang for their guidance and for taking part in the Examining Committee. Their input allowed this thesis to improve far past what it could have been without them.

My parents and significant other are also worthy of praise for all of their support and patience during this journey. Their faith and eagerness to assist deal with non-academic problems were invaluable to the completion of this thesis.

Finally, I'd like to thank Dr. Galia Dafni, for seeing my potential and helping me while I figured out which specific direction I wanted to go in my graduate studies.

Contents

List of Figures

Chapter 1

Introduction

> "Artificial intelligence (AI) plays a crucial role in shaping the modern world, transforming industries and impacting daily life in unprecedented ways. By enabling machines to process vast amounts of data, learn from it, and make decisions, AI powers innovations in fields ranging from healthcare and finance to transportation and entertainment. In healthcare, AI aids in diagnosing diseases, predicting patient outcomes, and personalizing treatment plans, thus improving patient care and outcomes. In finance, it strengthens fraud detection, optimizes trading, and enhances risk management. Self-driving vehicles and smart city planning showcase how AI can make transportation safer and more efficient, while AI-driven recommendation systems in media and retail create more personalized experiences for users. Moreover, AI is increasingly used to tackle complex global challenges, such as climate change and energy optimization. As AI continues to advance, its responsible and ethical use becomes essential to ensure it benefits society while addressing concerns around privacy, bias, and security."
>
> — ChatGPT

Artificial intelligence (AI) has been a topic of great importance as of late. We can see in Figure 1.1 that Artificial Intelligence research has been growing incredibly fast over the last decade, with a specific focus on machine learning [17]. Even people with no knowledge or interest in the field are exposed to it through the electronic devices and applications they use every day. Facebook, one of the most popular social networks, has an AI ChatBot to "help" by making suggestions about what to search for or even what to type in a conversation. Multiple popular conversation-able Chatbots are used for leisure and even for education. Some graduate students even use AI as a replacement for Google or StackExchange for finding which commands and how to use them exist while coding. The incredibly popular search engine Google often suggests AI-written answers to queries, sometimes deeply inappropriate - for example, touting the health benefits of running with scissors [9].

It is not trivial to determine exactly when this "AI revolution" began. Humans have been curious about thinking and artificial thinkers for a very long time - even as far back as Ancient Greek Mythology there were writings of Talos, often described as an artificial life form designed by Hephaestus [25]. Jewish folklore has the Golem, an artificial thinking being made from some simple building material such as clay or mud [8]. AI has been a

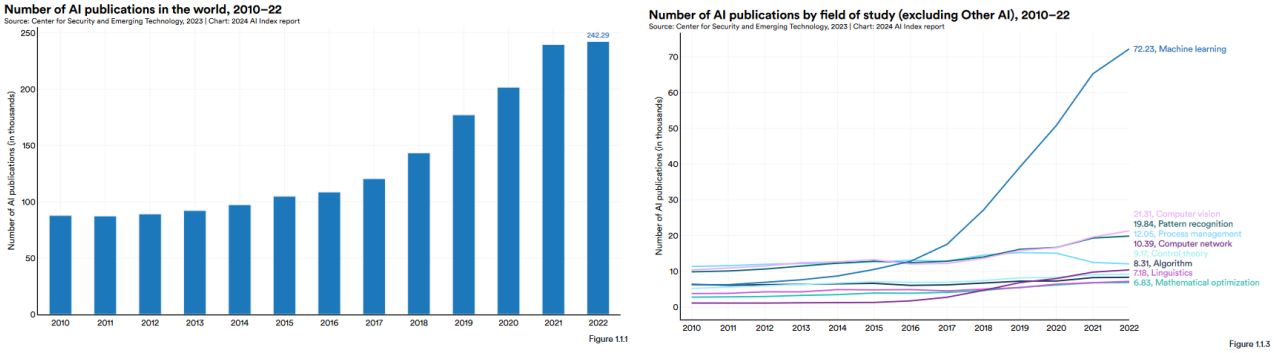

Figure 1.1: Number and field of AI publications in the world, 2010-22 (found in [17])

mainstay in science fiction for decades, often as an existential threat to human existence such as in popular films Terminator or the Matrix (which somehow has little to do with linear transformations).

Linear regression and the method of least squares appeared around the year 1800, which could be considered the first true artificial neural network - albeit a simple, linear one [24]. The preceptron, the first true (however simple) neural network for binary classification, dates back to the 1943 [18]. The first Feedforward Neural Networks (FNNs, see Section 2.2.2) deep learning systems appeared in the 1960s and early 1970s [24]. From there, many more types of neural networks were proposed, some of which are explored in Section 2.2.3.

It is difficult to overstate the impact AI is having on modern life. Apple's AI assistant Siri was used many times during this thesis to set reminds and alarms. AlphaGo has finally done the unthinkable, and has grown able to defeat even the best Go players (a popular abstract board game with a famously massive decision space) with some consistency [6]. Popular streaming service Netflix uses an AI driven recommendation system to direct viewers to media they might enjoy [20].

AI has even begun to encroach on fields many were not expecting to be possible - the visual arts. There are great controversies forming as to what is considered art and whether AI driven art is naturally infringing on artists' intellectual property through training sets. These controveries have begun to have a significant impact on some industries: particularly, board and video games with AI created art have become more and more common, a fact that many artists and consumers are not happy with.

It is easy to take for granted that machine learning is capable of a great deal, especially as it begins to seep into the consciousness of society. It becomes natural to ask what kinds of limits these powerful tools might have. In fact, mathematician Steven Smale, writer of "Mathematical Problems for the Next Century" ([26]), included as his 18th problem: "What are the limits of intelligence, both artificial and human?" For example, it is well known that deep neural networks have major vulnerabilities to "adversarial attacks" [12]. These "attacks" refer largely to specifically chosen inputs which lead to outputs that are very distant from the ground truth. For example, the *DeepFool* algorithm is put forth in [19], where very small changes to a picture, nearly invisible to the naked eye, completely change the categorization of an image for a specific learner. A picture of an elephant with a few discoloured pixels in the corner might suddenly be read as a whale. If this type of extreme failure is possible,

what other types of failures are there?

1.1 Thesis Objectives

In this thesis we will explore one particular example of another issue that may arise in the training and use of neural networks - that of *identity effects* (described in great detail below and in Section 2.3). Here we try to mathematically study one particular limitation of AI through theorems, then by reinforcing the theorems with numerical experiments.

Children learn to tell whether two sounds are identical at a very young age - even while still babies, they begin to develop the ability [16]. Because of the similarity between biological brains and neural networks, it seems reasonable to ask if a neural net has the same ability.

Instead of checking for identical sounds, though, for simplicty of the testing itself we will create two-letter (and eventually three-letter) "words" to see if a deep neural network has the ability to recognize whether or not both letters are the same. So an example "good" word might be 'AA' or 'BB', and a "bad" word could be 'AB' or 'CA'. For a human observer, it is trivial to guess that the proper pattern is probably about matching letters, and assume that 'XX' is good and 'ZY' is bad, even though none of those letters have been seen before. The question becomes: can a deep neural net generalize the matching letter pattern to letters it hasn't seen before?

This is not the first study into the topic of neural networks and identity effects. In fact, in Gary Marcus' book [16] strong claims were made empirically showing that neural networks can't generalize this kind of pattern.

It turns out that the answer is slightly more complicated than a simple "yes" or "no". Because of the structure of a neural net, one cannot simply input a letter, and must instead "encode" the information into vectors - and the generalization ability of the network depends on exactly how that data is encoded. We wish to study a specific relationship between how the data is encoded and how well the learner can generalize the pattern from limited data.

1.2 Thesis Outline

We begin in Chapter 2 by introducing the basic concepts of neural networks and encodings. We then flesh out the meaning of "Identity Effects" and explore several types of encodings and a specific transformation we call τ that appears linked to the issues with generalization. We also touch briefly on relevant modifications of τ and on patterns other than the simple two-letter-matching.

In Chapter 3 we then explore and confirm many of the experiments in [3] to establish a basis for the work. We then explore different types of encodings and the patterns that might cause some to allow some degree of generalization, exploring different types of what are known as "distributed" encodings and "noisy one-hot" encodings. Our main result is Theorem 3.4, which is a large inequality that places a limit on the ability of a neural network to generalize in certain common cases.

We conclude with Chapter 4, in which we review what has been found and then posit some open problems and how this research might best be continued into broader topics.

Chapter 2

Background

In this chapter, we describe the general setting and explore previously published theorems in order to properly set the stage for our own results. We also prove some minor auxiliary results.

To begin, we give some basic notation in Section 2.1. We then follow with a larger section expanding on deep neural networks and deep learning in Section 2.2, within which we explore the basic definition of a Neural Network in Section 2.2.1 and give an example of the type of network we use in this paper (Feedforward Neural Nets) in Section 2.2.2. We'll discuss some common activation functions and optimization methods (SGD and Adam) in Section 2.2.5, then go into the specifics of Identity Effects throughout Section 2.3, giving an overview and basic theorems in Sections 2.3.2 through 2.3.6. Then we proceed to stochastic methods, a relevant transformation (τ), and several of its variants in Sections 2.3.7-2.3.11. Section 2.3.9-2.3.11 contain original results stemming from the work done in this thesis.

2.1 Basic Notation

Throughout this thesis, several norms will be used, including:

The Frobenius matrix norm, for an $m \times n$ matrix M: $||M||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|m_{ij}|^2}$.

The L^p function norm, for $f : X \to V$, where V is a vector space equipped with norm $\|\cdot\|$: $||f||_{L^p(X)} = (\int_X \|f\|^p)^{1/p}$.

L^∞ is a special case defined as $\|f\|_{L^\infty(X)} = \sup_X \|f\|$.

2.2 Deep Learning

Machine learning is a huge topic in modern literature, and it spreads itself across many academic and corporate fields. Though it has many subfields, in this paper we will focus on neural networks - specifically *deep* neural networks, the training of which is called *deep learning* [15]. For a more thorough introduction to the topic, see for example [13] or [2].

2.2.1 Introduction to Neural Networks

A neural network is a mathematical object which can be thought of as a function, which takes in some input and gives an output in a precise way. The input is usually a vector, or a vectorized piece of data, and the output can be thought of as a vector as well - though the specific form the vector takes can be very different from problem to problem. Our main focus will be *supervised learning*, the most common scenarios therein being *classification* (differentiating inputs between different categories) and *regression* (analogous to function approximation). The underlying structure of neural networks made for classification and those made for regression are fundamentally the same, with only slight variations.

Neural networks were designed with direct inspiration from how the biological brain works [11] - hence the name. In a brain, a neuron is either firing (on) or not (off), and then that information is transmitted to the next set of neurons, and then so on and so forth. In an (artificial) neural net, a very similar process takes place over several of what are known as "hidden layers" between the input layer and the output layer. First, we will describe in detail how a neural net functions, specifically a feedforward network (the simplest). Second, we will describe in broad detail the adapting of a network to a problem, known as *training*. Finally, other examples of neural networks will be briefly mentioned, but they are not the focus of this paper.

2.2.2 Feedforward Neural Networks

The first and simplest form of a neural network is a feedforward network [24]. It begins by taking in some (ideally vectorized) input and running multiple affine transformations on it - one per node in the first hidden layer - and then some non-linear transformation is applied to it, known as an "activation function". These affine transformations have coefficients known as "weights" and the constant additive terms are known as "biases". There are many types of activation functions, to begin let us examine those most similar to what happens in the actual human brain - they take in some data, and then the neuron either fires, or it doesn't. This can be seen as a step function - it fires (1) for certain input values, and doesn't fire (0) for others.

So for one layer, we have our input $\mathbf{x}$, our weights W, our biases b, and our activation function σ. Note that together, $W\mathbf{x} + \mathbf{b}$ is an affine function of x. A single layer is given explicitly by

$$A(\mathbf{x}) = \sigma(W\mathbf{x} + \mathbf{b}),$$

where the activation function σ is applied component-wise.

A multilayer network comes from composing multiple layers. It is common for each activation function other than the last one to be the same, then the final function forces the data into a more appropriate shape. For example, if one is attempting to classify data into n categories, the final activation function will convert the previous layer's output into a vector of n probabilities using the softmax function (the softmax function takes a vector of values $\mathbf{v} \in \mathbb{R}^k$ and creates a vector of probabilities $\mathbf{z}$: $z_i = e^{v_i} / \sum_{j=1}^{k} e^{v_j}$). In sum, the full neural network is a function of the form A_{tot} below:

$$A_{tot}(\mathbf{x}) = \sigma_n(W_n(...(\sigma_2(W_2(\sigma_1(W_1(\sigma_0(W_0\mathbf{x} + \mathbf{b}_0)) + \mathbf{b}_1)) + \mathbf{b}_2)) + ...)) + \mathbf{b}_n).$$

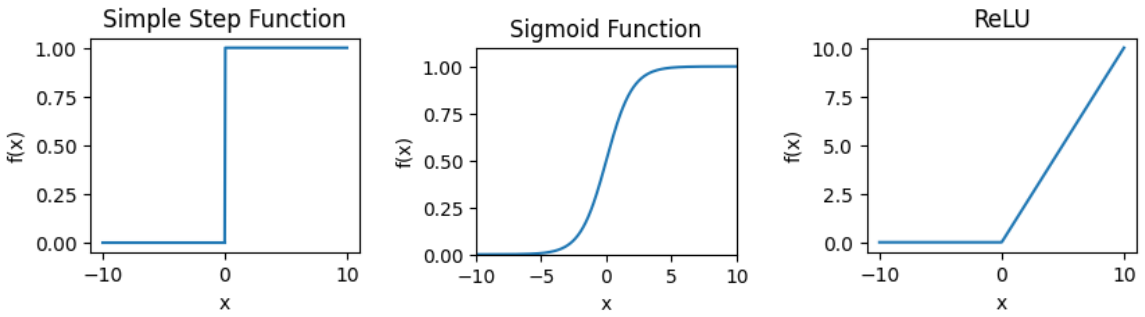

Figure 2.1: Step function compared to popular activation functions

It was very common early to use the sigmoid function as it acts as an approximation to a smooth step function: $\text{sigmoid}(x) = \frac{1}{1-\exp(-x)}$. See Figure 2.1 for a comparison to a step function. There are many activation functions, but the one we use the most often in this paper is the "Rectified Linear Unit" (ReLU), a simple function that acts as a linear function when $x \geq 0$, but is 0 elsewhere:

$$\text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0. \end{cases}$$

We now have enough to put together a toy network. Let's imagine a neural network that takes in three-dimensional data and outputs a simple binary (either 0 or 1). Our toy network has 3 hidden layers, with 2, 4, and 3 neurons each, respectively. See Figure 2.2 for a visual representation. At each node in the first hidden layer, the three inputs are each multiplied by a weight and summed with a bias. Then the ReLU above is applied. Note that since there are 2 neurons with 4 parameters each, that gives us 8 parameters in the first hidden layer. Now since we have two outputs in the first layer, any specific neuron in the second one will sum their weighted outputs and add a bias - making 3 parameters per neuron and 12 total in the second layer. The activation function (ReLU for us) is then applied once more, giving us four new outputs for the third layer to take in in a similar manner, so each neuron has 5 parameters, giving us a total of 15 in this layer. Lastly, we have our final layer, which will act similarly to any other neuron - taking in the previous outputs, multiplying them by weights, adding them to a bias - but won't have an activation function in the same sense. In our case, we might have a final output of 0 if the value of the final affine transformation is negative, and a final output of 1 otherwise. This final layer has a total of 4 parameters itself.

This small toy network has $8+12+15+4 = 39$ parameters. As one can imagine, a neural net in actual applications has far more layers and neurons, and hence many more parameters. This can cause major problems when it comes to a network "learning" what these parameters should be - and this is largely the topic of interest in neural networks. We discussed how a feedforward network takes in inputs and gives an output, but a far more interesting question is how to create a neural net to approximate some ground truth function - and to do that we must discuss loss functions and two very popular methods of learning based on gradient descent.

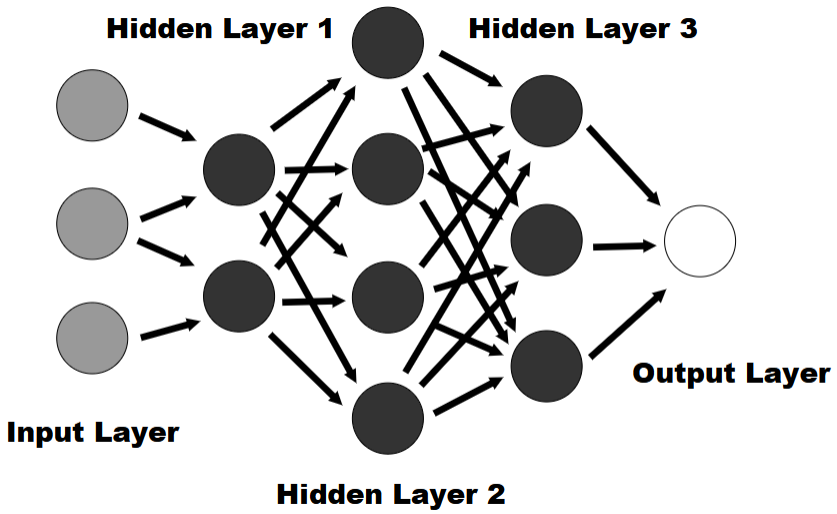

Figure 2.2: Example network

2.2.3 Other Common Network Types

There are many types of deep neural networks outside of the FNNs. Throughout the rest of this thesis we use FNNs due to their computational accessibility, but we provide descriptions for two other particularly relevant network types.

Recurrent Neural Networks, or RNNs, are neural networks that do not simply pass through their input data all at once, but instead do it in multiple steps. This allows them to be more powerful for processing sequential data. This is achieved by allowing neurons to pass information to one another even in a single hidden layer. In particular, *Long Short-Term Memory* (LSTM) networks are a particular type of RNN that allow promoting long term memory storage when it is useful, and forgetting data otherwise. For a more complete view of RNNs and LSTMs, see [10] and [22].

Graph Neural Networks [23], or GNNs, are neural nets that learn functions on graphs. Succinctly, a graph is a mathematical object made up of nodes (V) and edges (E) connecting some, all, or none of those nodes. In GNN terms we also require *node features* on each vertex - so in total, a graph can be written $G(V, E, \alpha)$. GNNs encode the data in graphs into vectors before learning, and so can have similar questions to our own FNN networks.

Identity effects and how they appear in LSTM networks and GNNs are explored specifically in [3] and [4], respectively.

2.2.4 Learning and Backpropagation

We consider a "loss function", which represents how well (or poorly) the neural network fulfills our criteria. Different loss functions exist, and can be used for different purposes, but each assigns a value to the output of the neural net when compared to the desired output.

Using this loss function, one applies "backpropagation" to calculate gradients (the details of which are beyond the scope of this thesis, but an extensive explanation can be found in [13]), and an iterative optimization algorithm is applied to reduce the loss and increase the

accuracy of the model.

2.2.5 SGD and Adam

Stochastic Gradient Descent (see [2]), or SGD, was once one of the most important tools used in deep learning. It has since been replaced with similar, more effective tools such as Adam (introduced in [14], but many operate on the same framework, so SGD is important to understand.)

SGD is an algorithm for optimization, and an evolution of Gradient Descent. The idea of Gradient Descent is that the gradient points in the direction of greatest increase, so if one were to move in the opposite direction, it would be towards a (local) minimum.

Let us begin by defining Gradient Descent. Say there is some differentiable function, $F(\Theta), F : \mathbb{R}^n \to \mathbb{R}$, and and a minimum is desired. We begin with a random input Θ_0. At each iteration i we compute:

$$\Theta_{i+1} = \Theta_i - \alpha_i \nabla F(\Theta_i)$$

where $\alpha_i > 0$ is the preset *learning rate* (which can vary from iteration to iteration). We continue this process until some stopping point has been reached - for example, if $F(\Theta_i)$ is below some threshold, or after some preset maximum number of iterations. It is worth noting that if the goal is to instead maximize, there is also Gradient Ascent - the same process, but we add the gradient instead of removing it ($\Theta_{i+1} = \Theta_i + \alpha_i \nabla F(\Theta_i)$).

Algorithm 1 (Gradient Descent)

Require: $F(\Theta)$: Objective function with input Θ
Require: α_i: Learning Rate Sequence
Require: Θ_0: Initial input
1: **procedure** *GD*
2: **while** Θ_i not converged **do**
3: $\Theta_{i+1} \leftarrow \Theta_i - \alpha_i \nabla F(\Theta_i)$
4: **end while**
5: **return** Θ_i

A common use of Gradient Descent is the case where the function is a loss function of some kind - a sum of functions over some data points $D = (\mathbf{d}_i)_{i=1}^n$. Since we are in the supervised case, each datapoint has had its output measured and can be written as $\mathbf{d}_i = (\mathbf{x}_i, y_i)$. In this case, the objective is to optimize the parameters of the function by "descending" towards the lowest total loss. Let us define a model

$$\hat{y} = \hat{f}(x, \Theta)$$

and a general loss function

$$F_D(\Theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(\hat{f}(x_i, \Theta), y_i),$$

Algorithm 2 (Stochastic Gradient Descent)

Require: $F(\Theta)$: Objective function with parameters Θ of the form $F_D(\Theta) = \sum_{d \in D} F_{\{d\}}(\Theta)$
Require: D: Dataset
Require: α_i: Learning Rate Sequence
Require: Θ_0: Initial parameters
1: **procedure** *SGD*
2: **while** Θ_i not converged **do**
3: Select batch (see below) $D_i \subset D$
4: $\Theta_{i+1} \leftarrow \Theta_i - \alpha_i \nabla \sum_{d \in D_i} F_{\{d\}}(\Theta)$
5: **end while**
6: **return** Θ_i

where Θ represents all trainable parameters. Common loss functions include the ever present *Mean Squared Error* (MSE) [7], often seen even in elementary statistics, and *binary cross-entropy*, which we use in all experiments [3]. The mean squared error corresponds to the choice

$$\ell(y, \hat{y}) = (y - \hat{y})^2,$$

and the binary cross-entropy loss is

$$\ell(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}).$$

Stochastic Gradient Descent (Algorithm 2) is a very similar process, but instead of using the entire dataset at every iteration, the dataset is looked at one random data point or set of data points at a time. This speeds up the process dramatically by reducing the number of computations that needs to be done, as since $\nabla F_D = \sum_{i=1}^{n} \nabla f_{\{\mathbf{d}_i\}}$ we need only compute $\nabla F_{\{d_i\}}$ for the datapoints in question. It is also possible to use *batching* to instead process the data more than one data point at time. Algorithm 2 refers to SGD with batching, for SGD without batching the batches D_i would instead be single datapoints $\mathbf{d}_i$.

A selected batch can be random or on a pretedermined schedule, so long as the selection is independent of the parameters in question Θ. An *epoch* is said to have been completed when each datapoint has been included in some D_i since the last epoch ended.

Lastly, one of the most common modern stochastic optimization processes is Adam, short for Adaptive Moments (see [14]). The precise details are beyond the scope of this paper, but seeing as Adam is the process used for the numerical experiments in this paper, we provide an overview.

Adam requires a step-size α_i, but it also requires two decay rates, β_1 and β_2. It begins as Gradient Descent, but instead of updating the parameters directly, it instead stores estimates of the biased first and second moments of the function $F_D(\Theta)$. At each step, it calculates them using the gradient, stores them, corrects the biases, then calculates the new parameters using the corrected biases. Then it begins anew. Once the estimates converge, the process stops. The algorithm is omitted for brevity.

2.3 Identity Effects

In this section the idea of Identity Effects in Neural Networks will be explored. "Identity effects" refers to a learner attempting to "learn" whether two things are identical, as described in the introduction. We will discuss basic identity effects and present an impossibility theorem (proving that for some learners and datasets the learner cannot truly distinguish between things being identical and not), and then expand that into more general situations. We define several encodings and describe how the they influence the impossibility, and then progress into the idea of "approximate" impossibility.

2.3.1 Introduction to Identity Effects - Literature Review

In natural learning situations, it turns out that humans have a really easy time distinguishing where or not two sounds are "identical" from a very young age [16]. Somewhat surprisingly, the same is not true of neural networks. This paper will focus on some specific situations in which neural networks fail to distinguish whether two things are identical - hence the name "Identity Effects".

2.3.2 The Setting

First, let us define a notation for the general setting. Let us define W as the vector space of all possible inputs w, where the choice of the letter "w" here is meant to invoke the idea of "words". For most purposes, including ours, $W = \mathbb{R}^d$, though we only use a small subset of it for viable inputs, as we only have a finite number of relevant objects. As our task is currently a simple binary classifier, each input has a rating $r \in [0, 1]$, describing how likely it is to be in the class "1".

We then have a dataset D, a finite list of individual data points that are comprised of a "word" and its rating $(w, r), w \in W, r \in [0, 1]$. For our dataset, the ratings are confined to the set $\{0, 1\}$, as they are created and sorted without the use of any learner. We define $\mathcal{D}$ as the set of all possible datasets.

Next we will consider a learning algorithm $L : \mathcal{D} \times W \to \mathbb{R}$, and what it learns given a given dataset D, and then apply it to an input w. This will give us a rating r. For consistency with the literature, we will refer to this whole function as $L(D, w)$. Note that we can also see this as a function $f(\Theta, w)$ that takes in some parameters, which we will define in total as Θ, as well as a datapoint in order to give us that data point's rating. In other words:

$$L(D, w) = f(\Theta, w) = r.$$

2.3.3 A Simple Impossibility Theorem

A given algorithm trained on such a dataset D should ideally be able to recognize whether or not objects are identical, even pairs of objects it has not seen before - those w in W but not in D. We now show that at least in one very specific instance, it is impossible for a learner to ever be able spot differences.

The method to approach this is to find some linear transformation $\tau : W \to W$ (with corresponding matrix T) we can apply to the data so that the learner cannot distinguish

between the original data and the transformed data. There are two relevant properties the pair of a learner $L(D, w)$ and a τ can have, and as we'll see in Theorem 2.1 any learner-dataset pair for which there exists a transformation with those properties will lead to impossibility.

1) *Invariance of the Data*: Simply, $\tau(D) = D$. As one might imagine, this means that the dataset and the transformed dataset are completely equivalent. For example, if our dataset includes all possible strings of 4 letters, and we were to swap each instance of letter A to the letter B, and each instance of the letter B to the letter A, it would still contain all possible strings of 4 letters, and thus be indistinguishable and hence invariant to the transformation.

2) *Invariance of the Algorithm*: $L(\tau(D), \tau(w)) = L(D, w)\ \forall w \in W$. The learning algorithm, when trained on the transformed dataset, assigns the same rating to the transformed input as it would have assigned to the original input when trained on the original dataset.

This brings us to our first theorem, the proof of which is so concise that it is being listed in the same matter as in the original text.

Theorem 2.1 ([3, Theorem 1])**.** *For any learning algorithm that satisfies both property 1 and property 2, the algorithm will not be able to differentiate between the transformed $\tau(w)$ and the original datapoint w, that is*

$$L(D, \tau(w)) = L(D, w).$$

Proof. $L(D, (\tau(w)) = L(\tau(D), \tau(w)) = L(D, w)$. □

As a toy example, one can consider W to be the set of words that are two letters long, taken from the first seven letters of the alphabet (A through G) with a rating of 1 if the two letters are identical, and 0 otherwise. (Note: this cannot be run through a neural network, as it is not a vector in $\mathbb{R}^n$, but our setting is still more general.) Our dataset D will be the set of all pairs that only contain the first five letters (A through E). To ease visualization, some example datapoints are (A, A, 1), (E, C, 0), and similar.

Next our transformation τ will replace any instances of the letter F with the letter G, and vice versa. It is easy to see that $D = \tau(D)$, as there are no Fs or Gs in D. The Invariance of the Algorithm is a much more complex topic and will be studied further deeper in this thesis.

2.3.4 τ as an Adversarial Attack

It is possible to view this τ as a form of adversarial attack (as mentioned in the Introduction). τ allows one to find specific examples $\tau(w)$ that lead to misclassification.

Let us take the toy example above, with no Fs or Gs present in the datase, and assume invariance of the algorithm. We see that $\tau(\text{FF}) = \text{FG}$, so since $L(D, \tau(w)) = L(D, w) \rightarrow L(D, \text{FF}) = L(D, \text{FG})$ either FF or FG must be misclassified. Clearly, τ has generated an adversarial example, and is therefore an adversarial attack.

2.3.5 Encodings

The purpose of an encoding is to take data and turn it into some form a neural net can take as an input - in other words, a simple numerical vector. Each encoding explored in this paper

is a matrix where each column corresponds to one letter's encoding, and where each letter's encoding is unique. To create a "word" encoding, we concatenate the letter encodings. The dimension of each encoding is largely independent from the number of letters, so long as there is enough information (see *distributed* encodings immediately below), and so we define this dimension to be n. There are several types of encodings used in this paper, including:

- *One-Hot*: A one-hot encoding assigns each input to a position in a vector. The vector that represents a single input has a 1 in that position, and a 0 in all others. This requires a vector of dimension at least equal to the number of inputs.

  Example: A $= (1, 0, 0, ...)^\top$, B $= (0, 1, 0, ...)^\top$, ..., Z $= (0, 0, ..., 1)^\top$

- *Distributed*: A distributed encoding involves having a vector of some dimension, and each letter is encoded using a combination of "activated bits" (ie bits set to 1, where all others are set to 0). If you have N different objects, an ambient dimension of d_a, and b activated bits, it is clear that one requires $d_a C b \geq N$. Which activated bits correspond to which letter are decided at random, while ensuring that no two letters have identical encodings. How different values of d_a and b affect the encodings are explored later.

  Example with $b = 3$, $d_a = 7$: A $= (1, 0, 1, 1, 0, 0, 0)^\top$, B $= (0, 0, 1, 0, 1, 0, 1)^\top$, ..., Z $= (1, 1, 0, 0, 0, 1, 0)^\top$

- *Noisy One-Hot*: Noisy One-Hot encodings are similar to One-Hot encodings, but each encoding has a random Gaussian vector added to it. The way different variances influence the encoding is also explored later.

  Example with variance $\mu^2 = 0.1$: A $= (1.03, -0.2, 0.1, ...)^\top$, B $= (0.2, 0.84, -0.3, ...)^\top$, ..., Z $= (-0.5, 0.4, ..., 1.1)^\top$

Note that one-hot encodings are a special case of distributed encodings where d_a equals the number of possible letters and b equals 1, and are a special case of noisy one-hot encodings where $\mu = 0$.

One-hot and distributed encodings are both examples of "binary encodings", as each of their entries is either 0 or 1.

Matrices corresponding to a few random encodings are included in Figure 2.3 to aid in visualization. Here each column represents 1 letter's encoding.

2.3.6 "Factorizable" Models

Recall that we can see our model that pairs each datapoint with its rating, given as a function of some parameters and our datapoint ($f(\Theta, w) = r$). Let us consider a model in which we can "factor" these functional inputs such that we can write $f(\Theta, w) = f(B, Cw)$, where C is a matrix which contains all multiplicative coefficients on w and B contains all other parameters. Though this type of setting appears to be quite specific, on the surface, it turns out that most neural network models use this structure, as the initial operation is nearly always a linear transformation, as discussed in Section 2.2.2.

Now we must discuss where B and C originate. We have a useful result if they arise from some loss function with no regularization term (see (2.2) below for the alternative), that is, one of the form:

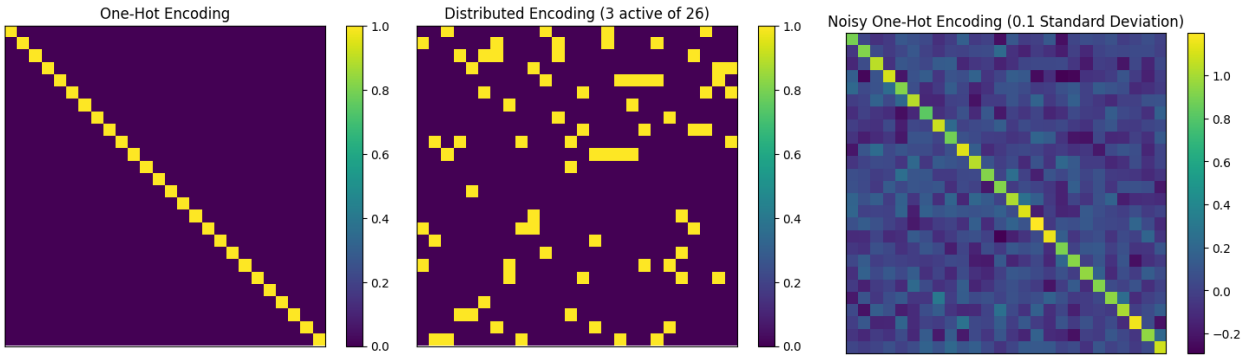

Figure 2.3: Matrix visualizations for select encodings

$$F(B, C) = \mathcal{L}(f(B, Cw_i), r_i | i = 1, 2, ...n) \tag{2.1}$$

A common example of this type of loss function could be the mean squared error as seen in statistics, in our case: $F(B, C) = \sum_{i=1}^{n}(r_i - f(B, Cw_i))^2$.

Theorem 2.2 ([3, Theorem 2])**.** *For any learning algorithm based on the form $f(B, Cw)$ optimized with parameters set by loss function of the form* (2.1) *that has and attains a unique minimizer $(\hat{B}, \hat{C})$ for a given dataset D, i.e.,*

$$L(D, w) = f(\hat{B}, \hat{C}w),$$

we have that L is invariant to all invertible linear transformations τ:

$$L(\tau(D), \tau(w)) = L(D, w), \quad \forall w \in W.$$

Proof. As discussed earlier, we have that $\tau(x) = Tx$, for some matrix T, which has an inverse T^{-1} by hypothesis. Then, upon optimizing for our transformed dataset we have

$$L(\tau(D), \tau(w)) = f(B', C'(Tw))$$

for some B' and C' obtained by minimizing the loss $\mathcal{L}(f(B, CTw_i), r_i, i = 1, 2, ...n)$. Now since we had a unique minimizer, we have that

$$\hat{C} = C'T \implies C' = \hat{C}T^{-1}$$

$$\text{and } B' = \hat{B}.$$

Therefore

$$L(\tau(D), \tau(w)) = f(B', C'Tw) = f(\hat{B}, \hat{C}w) = L(D, w).$$

□

Remark 2.3. *It is worth noting that the ambient space is not relevant in this theorem - it applies even to non-linear spaces.*

Next, let us consider a similar case except where our loss function has a regularization term:

$$F_{Reg}(B,C) = F(B,C) + \lambda\mathcal{R}(B,C) = \mathcal{L}(f(B,Cw_i), r_i, i = 1,2,...n) + \lambda\mathcal{R}(B,C) \tag{2.2}$$

Where $\lambda \geq 0$ is a tuning parameter which decides how much the regularization term "penalizes" the loss function: where $\lambda = 0$, there is no penalty, but as λ increases, whatever our regularization term is will penalize more and more . There are several forms this regularization term can take, the simplest of which is similar to *ridge regression* [7]:

$$\mathcal{R}(B,C) = \sum_i B_i^2 + \sum_{i,j} C_{ij}^2.$$

Here, the regularization term penalizes large parameters. As λ goes to 0, the penalty for large parameters becomes small (and becomes non-existent at $\lambda = 0$, removing the regularization term), but as it goes to ∞, the magnitude of the parameters is penalized and is driven towards 0.

For the remainder of this section, we will refer to $F_{Reg}(B,C)$ as $F(B,C)$, for convenience.

Theorem 2.4 ([3, Theorem 3])**.** *Consider the same setting as Theorem 2.2, but where the loss function follows form* (2.2) *instead, and the transformation matrix T, when right-multiplied by the second input of $\mathcal{R}$, has no effect on the regularization term: $\mathcal{R}(B,CT) = \mathcal{R}(B,C)$. We have that L is invariant to all invertible linear transformations τ:*

$$L(\tau(D), \tau(w)) = L(D,w) \quad \forall w \in W$$

Proof. As in Theorem 2.2, we transform the data and optimize once again, finding optima B' and C'. Then we have:

$$F(B',C') = \mathcal{L}(f(B',C'Tw_i), r_i, i = 1,2,...n) + \lambda\mathcal{R}(B',C')$$

But since, $\hat{B}$ and $\hat{C}$ are unique minimizers, we must also have:

$$F(\hat{B},\hat{C}) = \mathcal{L}(f(\hat{B},\hat{C}w_i), r_i, i = 1,2,...n) + \lambda\mathcal{R}(\hat{B},\hat{C})$$

So then, combining these and the fact that $\mathcal{R}(B',C'T) = \mathcal{R}(B',C')$ by hypothesis, we have:

$$B' = \hat{B}, \quad C' = \hat{C}T^{-1}.$$

The rest of the proof continues as in Theorem 2.2. □

2.3.7 Stochastic Methods

So far, we have only shown that invariance of the algorithm occurs in the case of unique minimizers, which is an idealized scenario. Now we will enter into more realistic applications - such as Stochastic Gradient Descent (SGD, see Algorithm 2). We will also need the concept of equality in distribution. We continue to assume that our learning algorithm input is "factorizable" as $f(B,Cw)$, as in Section 2.3.6. We assume our loss function to have the form:

$$F(B,C) = \mathcal{L}(f(B,Cw), r|(w,r) \in D) + \lambda(\mathcal{R}_1(B) + \|C\|_F^2) \tag{2.3}$$

Where here $\lambda \geq 0$ (the weak inequality allows for unregularized loss functions) and $\|\cdot\|_F$ is the Frobenius norm, defined in Section 2.1.

Definition 2.5. *Two real-valued random variables are considered to be equal in distribution (denoted "$\stackrel{d}{=}$") when their distribution functions are equal, that is to say*

$$X \stackrel{d}{=} Y \iff P(X \leq x) = P(Y \leq x) \forall x$$

Theorem 2.6 ([3, Theorem 4])**.** *Let T be an orthogonal matrix (i.e., one such that $T^\top T = TT^\top = I$.) Suppose one uses SGD (as defined in Algorithm 2) with a pre-determined step size sequence $\{\alpha_i\}_{i=1}^k$ and a loss function in the same form as (2.3), where $F(B, C)$ is differentiable with respect to B and C, to generate a sequence of parameters starting at (B_0, C_0) and ending at (B_k, C_k). Next, assume that $C_0 \stackrel{d}{=} C_0 T$ and that the random initialization of B_0 and C_0 were independent. Such learners, $L(D, w) = f(B_k, C_k w)$, are invariant to such transformations, that is*

$$L(\tau(D), \tau(w)) \stackrel{d}{=} L(D, w).$$

Proof. Let us find (B'_k, C'_k). Since the initialization is random and independent of our dataset, $(B'_0, C'_0 T) \stackrel{d}{=} (B_0, C_0)$. We will use mathematical induction on the sequence $(B'_i, C'_i)_{i=0}^k$. Next, defining F_X as the loss function when applied only on data in X, and D_i to be the data in the i'th batch we can see that

$$\begin{aligned} F_{\tau(D_i)}(B, C) &= \mathcal{L}(f(B, Cw), r | (Tw, r) \in D_i) + \lambda(\mathcal{R}_1(B) + \|C\|_F^2) \\ &= \mathcal{L}(f(B, CTw), r | (w, r) \in D_i) + \lambda(\mathcal{R}_1(B) + \|CT\|_F^2) \\ &= F_{D_i}(B, CT). \end{aligned}$$

Next, we use the chain rule

$$\begin{aligned} \frac{\partial F_{\tau(D_i)}}{\partial B}(B, C) &= \frac{\partial F_{D_i}}{\partial B}(B, CT) \\ \frac{\partial F_{\tau(D_i)}}{\partial C}(B, C) &= \frac{\partial F_{D_i}}{\partial C}(B, CT) T^\top. \end{aligned} \tag{2.4}$$

To see that $B'_{i+1} = B_{i+1}$, using the inductive hypothesis $B'_i \stackrel{d}{=} B_i$ and the definition of SGD:

$$\begin{aligned} B'_{i+1} &= B'_i - \alpha_i \frac{\partial F_{\tau(D_i)}}{\partial B'}(B', C) \\ &= B'_i - \alpha_i \frac{\partial F_{D_i}}{\partial B'}(B', CT) \\ &\stackrel{d}{=} B_i - \alpha_i \frac{\partial F_{D_i}}{\partial B}(B, CT) \\ &= B_{i+1}. \end{aligned}$$

Similarly, for C'_{i+1}:

$$\begin{aligned} C'_{i+1} &= C'_i - \alpha_i \frac{\partial F_{\tau(D_i)}}{\partial C'}(B', C') \\ &= C'_i - \alpha_i \frac{\partial F_{D_i}}{\partial C'}(B', C'T) T^\top \\ &\stackrel{d}{=} C_i T^{-1} - \alpha_i \frac{\partial F_{D_i}}{\partial C}(B, CT) T^\top \\ &= C_{i+1} T^{-1}. \end{aligned}$$

Combined, and plugging into our basic equation for L, we have

$$L(\tau(D), \tau(w)) = f(B'_k, C'_k T w) \stackrel{d}{=} f(B_k, C_k w).$$

□

An Example Transformation τ

A commonly useful example transformation that many learner-encoding-datasets are invariant to simply swaps two "letters" (or analogues) that weren't included in the dataset, and leaves the dataset as it was. So if our set W is a pair of objects, (x, y), and we have two specific objects A and B that are left out of our dataset D, our transformation τ does the following:

$$\tau(x, y) = \begin{cases} (x, B) & \text{if } y = A, \\ (x, A) & \text{if } y = B, \\ (x, y) & \text{otherwise.} \end{cases} \tag{2.5}$$

Let us define this τ's transformation matrix in the one-hot case $T^{OH} : \mathbb{R}^{52} \to \mathbb{R}^{52}$. First, we define S, a matrix that swaps the final two components. Without loss of generality, we assume that the final two letters are the ones that are not seen in the dataset.

$$S = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \in \mathbb{R}^{26 \times 26} \tag{2.6}$$

Now T^{OH} just applies S to the second "letter" in our encoding, and the first letter is unchanged:

$$T^{OH} = \begin{pmatrix} I_{26} & \mathbf{0} \\ \mathbf{0} & S \end{pmatrix},$$

where I_n denotes the $n \times n$ identity matrix and $\mathbf{0}$ denotes a zero matrix of suitable dimension.

This is an orthogonal matrix: and so Theorem 2.6 applies even in many real world settings when using SGD. Adam is beyond the scope of this paper, but behaves similarly, as seen in the Appendix of [3].

2.3.8 Modifying T^{OH}

It is clear that for other types of encodings, τ's transformation matrix T^{OH} must be modified, in that each letter's encoding is non-zero in more than one component. We assume that the encoding matrix is invertible.

The process is similar, but with an additional step. We define $R' \in \mathbb{R}^{26\times 26}$ so that each letter's encoding is transformed into the simple one-hot's case using elementary linear algebra. We then create a larger matrix $R \in \mathbb{R}^{52\times 52}$:

$$R = \begin{pmatrix} R' & \mathbf{0} \\ \mathbf{0} & R' \end{pmatrix}$$

For another encoding, denote the new τ's transformation matrix $\tilde{T}$. By transforming our vector input into a one-hot encoding by multiplying by R, then transforming it through T^{OH}, then transforming it back into the original, we can apply τ to any dataset. When our encoding is a basis, R is invertible and we have:

$$\tilde{T} = R^{-1}TR.$$

This matrix will not be orthogonal in most non-one-hot cases.

2.3.9 Properties of τ

A relevant question one might ask is whether the τ defined in equation 2.5 is linear or not. We prove that it is, under necessary and sufficient conditions on the encoding vectors. We will require a quick definition to aid in the proof.

Definition 2.7 (Tensor Product of Linear Maps)**.** *The tensor product of linear maps, denoted $\otimes$, is defined as follows, where F_1, F_2, ..., F_n are linear maps and x_1, x_2, ..., x_n are vectors:*

$$(F_1 \otimes F_2 \otimes ... \otimes F_n)(x_1, x_2, ..., x_n) = (F_1(x_1), F_2(x_2), ..., F_n(x_n)).$$

Proposition 2.8 (Linearity of τ)**.** *For a given encoding, we list the columns v_A through v_Z (all unique by the definition of an encoding, see Section 2.3.5), where v_Y and v_Z are not seen in the dataset, τ (as defined in equation 2.5) can be extended to a linear transformation if and only if all of the following conditions are met:*

1. $v_Y \notin Span(v_A, v_B, ..., v_X)$
2. $v_Z \notin Span(v_A, v_B, ..., v_X)$
3. $v_Y \notin Span(v_Z)$ *or* $v_Y = -v_Z$

Proof. τ takes in a concatenated pair of encodings. Since it has no effect on the first encoding, we can write its input as $x\mathbf{V}$, where x represents an arbitrary letter's encoding and $\mathbf{V}$ represents the specific second vector that may change.

$$\tau(x\mathbf{V}) = T(x\mathbf{V}) = \begin{cases} xv_Z & \text{if } \mathbf{V} = v_Y, \\ xv_Y & \text{if } \mathbf{V} = v_Z, \\ x\mathbf{V} & \text{otherwise.} \end{cases}$$

We now prove the two implications of the "if and only if".
$\Leftarrow$ We have that $v_A, v_B, ...v_Z \in \mathbb{R}^n$.

Consider the subspaces $S_1 = \text{Span}(v_A, v_B, ..., v_X) \subset \mathbb{R}^n$ and $S_2 = \text{Span}(v_Y, v_Z) \subset \mathbb{R}^n$. Let $b_1, b_2, ..., b_k$ be a basis of S_1. We define $k := \dim(S_1)$.

Now, given assumption 3, we have two cases.

• Case 1: $v_Y \notin \text{Span}(v_Z)$. Here, by assumption $v_Y \notin \text{Span}(v_A, v_B, ..., v_X)$ (and similar for v_Z), we have $k+2$ unique basis vectors for $S_1 + S_2$ in $\mathbb{R}^n$. We then add arbitrary linearly independent vectors $b_{k+1}, ..., b_{n-2}$ to $\mathbb{R}^n$ to complete this set to be a basis of $\mathbb{R}^n$: $B_1 = \{b_1, b_2, ..., b_k, b_{k+1}, ..., b_{n-2}, v_Y, v_Z\}$. Now we consider the change of basis transformation $S \in \mathbb{R}^{n \times n}$ from B_1 to B_2 defined by $B_2 = \{b_1, b_2, ..., b_k, b_{k+1}, ..., b_{n-2}, v_Z, v_Y\}$. This is a linear transformation by definition.

• Case 2: $v_Y = -v_Z$. We proceed similarly, though here as $\dim(\text{Span}(v_Y, v_Z)) = 1$ we require an additional basis vector: $B_1 = \{b_1, b_2, ..., b_k, b_{k+1}, ..., b_{n-1}, v_Y\}$. Now because if $\tau(v_Y) = v_Z = -v_Y$, we have that $\tau(v_Z) = \tau(-v_Y) = -\tau(v_Y) = -v_Z = v_Y$, we can consider the change of basis matrix $S \in \mathbb{R}^{n \times n}$ from B_1 to B_2 defined by $B_2 = \{b_1, b_2, ..., b_k, b_{k+1}, ..., b_{n-1}, -v_Y\}$, again, a linear transformation by definition.

In either case, τ can be extended to a linear transformation on all of $\mathbb{R}^{2n}$ as follows

$$\tau(x, y) = (x, Sy).$$

$\Rightarrow$ We prove Condition 1. Condition 2 is analogous.

Observe that τ is of the form $\tau = I \otimes \tau_2$, that is to say $\tau(x, y) = (x, \tau_2(y))$. Note that τ_2 is linear, as otherwise τ would not be.

We use proof by contradiction and assume otherwise: $v_Y \in \text{Span}(v_A, v_B, ..., v_X)$. Then there exists some c_js such that:

$$v_Z = \tau_2(v_Y) = \tau_2\left(\sum_{j \in \{A,B,...,X\}} c_j v_j\right) = \sum_{j \in \{A,B,...,X\}} c_j \tau_2(v_j) = \sum_{j \in \{A,B,...,X\}} c_j v_j = v_Y$$

which contradicts the uniqueness assumption that $v_Z \neq v_Y$.

For condition 3: We use contradiction again, assuming that $v_Y \in \text{Span}(v_Z)$ and $v_Y \neq -v_Z$. $v_Z = \tau_2(v_Y) = \tau_2(c_Z v_Z) = c_Z \tau_2(v_Z) = c_Z v_Y = c_Z^2 v_z$ which implies $c^2 = \pm 1$, so $v_Z = v_Y$, which again contradicts the uniqueness assumption. □

Remark 2.9. *Note that the second part of condition 3 is impossible in the case of a binary encoding, such as the one-hot or distributed encodings that we encounter.*

Remark 2.10. *Though we use two unseen letters, these theorems and proof generalize fairly trivially to other counts.*

2.3.10 More General Patterns: AAB

We have discussed the idea of attempting to train a model to check for matching components, but what about other patterns? There are many interesting ones that seem like they might have the same "identity effects", even if there is more than just an identity involved.

Say for example we consider words of the form "xxy" to be "good" words, that is two matching letters then one non-matching. It turns out that a very similar τ^{AAB} has the same properties:

$$\tau^{AAB}(x\mathbf{V}y) = T(x\mathbf{V}y) = \begin{cases} xv_Zy & \text{if } \mathbf{V} = v_Y, \\ xv_Yy & \text{if } \mathbf{V} = v_Z, \\ xv_?y & \text{otherwise.} \end{cases}$$

Since one can view τ as $\tau^{AB} = I \otimes \tau_2$, it is analogous to say that $\tau^{AAB} = I \otimes \tau_2 \otimes I$, and as such τ^{AAB}'s matrix representation is orthogonal for encodings in which τ's were orthogonal, and that the dataset is invariant to it. Numerical results about τ^{AAB} are included in Section 3.2.2.

There are many other possible patterns, but for any similarly simple "identity with a twist", a similar strategy for finding an appropriate τ will work.

2.3.11 A Specific Encoding: One-Cold Encoding Orthogonality

We now discuss a specific encoding that will appear again in the distributed encoding part of Section 3.2.5. It is of particular note as it exhibits numerical properties analogous to the one-hot encoding case.

It turns out that one-cold encodings (i.e., distributed encodings with $n-1$ active bits) always lead to orthogonal transformation matrices T. We prove it directly, though first we'll require a quick lemma.

Lemma 2.11. *Define* $A \in \mathbb{R}^{n\times n}$

$$A = \begin{pmatrix} 0 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 0 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 0 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & 0 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 0 \end{pmatrix}$$

and $S \in \mathbb{R}^{n\times n}$ *as an appropriately sized swap matrix first mentioned in equation* (2.6).

Then $A^2S = SA^2$.

Proof. For each component of A^2 we see that

$$(A)^2_{ij} = \sum_{k=1}^{n} a_{ik}a_{kj} = \sum_{k=1}^{n}(1-\delta_{ik})(1-\delta_{kj})$$

where δ represents the Kronecker delta. Then each summand is 1, unless $i = k$ or $j = k$. On the diagonal:

$$(A)^2_{ii} = \sum_{k=1}^{n}(1-\delta_{ik})(1-\delta_{ki}) = n-1,$$

because of the fact that $k = i$ only in one summand. When not along the diagonal, it is necessarily in two, as we instead have, for $\imath \neq j$

$$(A)^2_{i\neq j} = \sum_{k=1}^{n}(1-\delta_{ik})(1-\delta_{kj}) = n-2.$$

So,

$$A^2 = \begin{pmatrix} n-1 & n-2 & n-2 & \cdots & n-2 & n-2 \\ n-2 & n-1 & n-2 & \cdots & n-2 & n-2 \\ n-2 & n-2 & n-1 & \cdots & n-2 & n-2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ n-2 & n-2 & n-2 & \cdots & n-1 & n-2 \\ n-2 & n-2 & n-2 & \cdots & n-2 & n-1 \end{pmatrix}.$$

As we know from elementary linear algebra, left multiplying by S swaps the final two rows, whereas right multiplying swaps the final two columns. So,

$$SA^2 = \begin{pmatrix} n-1 & n-2 & n-2 & \cdots & n-2 & n-2 \\ n-2 & n-1 & n-2 & \cdots & n-2 & n-2 \\ n-2 & n-2 & n-1 & \cdots & n-2 & n-2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ n-2 & n-2 & n-2 & \cdots & n-2 & n-1 \\ n-2 & n-2 & n-2 & \cdots & n-1 & n-2 \end{pmatrix} = A^2S,$$

as desired. □

Theorem 2.12. *Consider any square matrix B where each column has one 0 entry, and all other entries as 1. Our standard transformation τ, defined by its associated block matrix*

$$T = \begin{bmatrix} B & 0 \\ 0 & B \end{bmatrix}\begin{bmatrix} I & 0 \\ 0 & S \end{bmatrix}\begin{bmatrix} B^{-1} & 0 \\ 0 & B^{-1} \end{bmatrix}$$

is orthogonal. (Here I is the appropriately-sized identity matrix and S is in Lemma 2.11.)

Proof. First, define A as as above and note that $A^\top = A$ and that $S^{-1} = S$:

Then for some permutation matrix P (where $P^{-1} = P^\top$, by the definition of a permutation matrix),

$$B = PA.$$

Now:

$$\begin{aligned} T^\top T &= \begin{bmatrix} B^{-\top} & 0 \\ 0 & B^{-\top} \end{bmatrix}\begin{bmatrix} I & 0 \\ 0 & S \end{bmatrix}\begin{bmatrix} B^\top & 0 \\ 0 & B^\top \end{bmatrix}\begin{bmatrix} B & 0 \\ 0 & B \end{bmatrix}\begin{bmatrix} I & 0 \\ 0 & S \end{bmatrix}\begin{bmatrix} B^{-1} & 0 \\ 0 & B^{-1} \end{bmatrix} \\ &= \begin{bmatrix} B^{-\top}IB^\top BIB^{-1} & 0 \\ 0 & B^{-\top}SB^\top BSB^{-1} \end{bmatrix} \\ &= \begin{bmatrix} I & 0 \\ 0 & B^{-\top}SB^\top BSB^{-1} \end{bmatrix} \end{aligned}$$

So if $B^{-\top}SB^{\top}BSB^{-1} = I$, then $T^{\top}T = I$. Now, observe that

$$\begin{aligned} B^{-\top}SB^{\top}BSB^{-1} &= (PA)^{-\top}S(PA)^{\top}(PA)S(PA)^{-1} \\ &= P^{-\top}A^{-\top}SA^{\top}P^{\top}PASA^{-1}P^{-1} \\ &= PA^{-1}SAP^{-1}PASA^{-1}P^{-1} \\ &= PA^{-1}SAASA^{-1}P^{-1} \\ &= PA^{-1}SA^2SA^{-1}P^{-1} \end{aligned}$$

So now we manipulate:

$$\begin{aligned} PA^{-1}SA^2SA^{-1}P^{-1} &= I \\ A^{-1}SA^2SA^{-1} &= P^{-1}P \\ SA^2SA^{-1} &= A \\ SA^2S &= A^2 \\ SA^2 &= A^2S^{-1} \\ SA^2 &= A^2S. \end{aligned}$$

The last equation holds thanks to Lemma 2.11. This concludes the proof. □

Chapter 3

Approximate impossibility in theory and practice

Now we will explore cases where we do not have true orthogonality of τ (as defined in Section 2.3.7), but instead *approximate* orthogonality. We begin with a theoretical analysis in Section 3.1, setting up prerequisites for and explaining the main theorem of this thesis (Theorem 3.4), and then show many numerical experimentations exploring the idea further in Section 3.2. We reproduce the results from [3] in Section 3.2.1 and explore another possible pattern in Section 3.2.2. We discuss noisy one-hot encoding experiments in Section 3.2.3, then a variety of different distributed encodings in Section 3.2.4. We move on to examining the broader view of distance from orthogonality in Section 3.2.5 and finally test some preselected encodings with particularly high and low distances from orthogonality in Section 3.2.6.

3.1 Theoretical Analysis

We have shown that identity effects, when one uses encodings that have a transformation that the dataset is invariant to (with an orthogonal transformation matrix), cannot be properly learned by a learner, under specific (but common) conditions. That leads to the next question quite naturally: what happens when the transformation matrix is not precisely orthogonal, but not very distant from it? The study of this sprang from the idea of studying "noisy one-hot" encodings, as described in Section 2.3.5.

To begin, we'll need to reference a basic definition in analysis: Lipschitz Continuity and Lipschitz Constants.

Definition 3.1 (Lipschitz Continuity)**.** *A function with a vector or matrix output, that is* $f : X \to \mathbb{R}^{n\times m}$ *(where* m *is 1 in the case of a vector output) is Lipschitz Continuous if, for an appropriate norm (we use* $\|\cdot\| = \|\cdot\|_2$ *for vectors and* $\|\cdot\| = \|\cdot\|_F$ *for matrices),*

$$\exists C > 0 : \forall x, y \in D, \|f(x) - f(y)\| \leq C\|x - y\|.$$

Intuitively, this states that the growth of the function is bounded in absolute growth - there is some constant that limits how much it can change for any given change in inputs.

Definition 3.2 (Lipschitz Constants for Real-Valued Functions)**.** *Each C in Definition 3.1 is a "Lipschitz Constant". For a function f and an input x, we define the infimum of such values to be* $\text{Lip}(f, x)$.

Proposition 3.3. *For any $F : X \to \mathbb{R}^m$,* $\text{Lip}(F, x) \leq \|\frac{\partial F}{\partial x}\|_{L^\infty}(x)$.

Proof. This is a standard result (seen for example in [5, Section III, Theorem 2]), but the proof is included for competeness. First, note that $F = [F_1, F_2, ..., F_m]^\top$. Next, we apply the *Mean Value Theorem* (MVT) from calculus to the functions $g_1, g_2, ..., g_m$ defined by

$$g_i(t) = F_i(tx + (1-t)y).$$

Now, $g_i'(t) = \frac{\partial F_i}{\partial x}(tx + (1-t)y)(x-y)$, and by the MVT we have $\frac{g_i(1)-g_i(0)}{1-0} = g_i'(\xi)$ for all $i = 1, 2, ..., m$ for some ξ in the line segment $\overline{xy}$. This gives

$$F(x) = F(y) + \frac{\partial F}{\partial x}(\xi)(x-y),$$

where $\frac{\partial F}{\partial x}$ is the Jacobian matrix

$$\left[\frac{\partial F}{\partial x}\right]_{ij} = \frac{\partial F_i}{\partial x_j}.$$

Here we assume that x, $F(x)$ are vectors so $\|x\|_2 = \|x\|_F$ and $\|F(x)\|_2 = \|F(x)\|_F$ (and otherwise, we reshape them to be as such). We also require [27, Theorem 5.3] which states $\|A\|_{2\to 2} = \sup_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} \leq \|A\|_F$. This leads to

$$\begin{aligned}
\|F(x) - F(y)\|_2 &= \left\|\frac{\partial F}{\partial x}(\xi)(x-y)\right\|_2 \\
&\leq \left\|\frac{\partial F}{\partial x}(\xi)\right\|_{2\to 2} \|(x-y)\|_2 \\
&\leq \max_{\xi\in X} \left\|\frac{\partial F}{\partial x}(\xi)\right\|_F \|x-y\|_2 \\
&= \left\|\frac{\partial F}{\partial x}\right\|_{L^\infty(X)} \|x-y\|_2.
\end{aligned}$$

We thus know the smallest of these $\left\|\frac{\partial F}{\partial x}\right\|_{L^\infty(X)}$ values is the Lipschitz constant, and this completes the proof. □

We will also require a simple arithmetic inequality and several of the norms defined in Section 2.1. We use the same notation as chapter 2.

We now introduce the main theorem of this thesis: the connection between approximate orthogonality and ratings impossibility.

Theorem 3.4 (Approximate Rating Impossibility)**.** *Let us consider a learner $L(D, x)$ whose model takes the form $r = \Phi(B, C_w)$ and a linear transformation τ with corresponding matrix T and invariance of the dataset ($\tau(D) = D$). We proceed using SGD, starting with initial parameters all equal to zero (i.e. $C_0 = 0$ and $B_0 = 0$ in all components) and a bounded*

learning rate ($0 < \theta_i \leq \theta_{\max} \forall i$). Assume also that all partial derivatives of the loss function with respect to the parameters up to second order are bounded and that the learner is Lipschitz continuous with respect to said parameters.

Then for some constant $C_{\Phi,k,F_D,\theta_{max}}$ depending on the model Φ, the number of iterations k, the loss function for the specific dataset F_D, and a learning rate bound θ_{max} we have

$$|L(D, \tau(x)) - L(D, x)| \leq C_{\Phi,k,F_D,\theta_{max}} \|T^\top T - I\|_F$$

with probability 1. (It is key to note that $C_{\Phi,k,F_D,\theta_{max}}$ does not depend on the specific T, so all relevant information about T is contained in $\|T^\top T - I\|_F$.)

Proof. Recall the notation of Chapter 2. Let us consider the learned functions $\Phi(B_k, C_k x) = L(D, x)$ and $\Phi(B'_k, C'_k T x) = L(\tau(D), \tau(x))$. Here $\Theta = (B, C)$ represents the set of all parameters in our model Φ and $\text{Lip}(\Phi, \Theta)$ represents the Lipschitz constant of the function $\Phi(B, C_w)$ with respect to Θ. Assume, up to vectorization, that $\Theta \in \mathbb{R}^p$.

First, we observe that

$$\begin{aligned}
|L(D, \tau(x)) - L(D, x)| &= |L(\tau(D), \tau(x)) - L(D, x)| \\
&= |\Phi(B'_k, C'_k T x) - \Phi(B_k, C_k x)| \\
&\leq \text{Lip}(\Phi, \Theta) \sqrt{(\|B'_k - B_k\|_F^2 + \|C'_k T - C_k\|_F^2)} \\
&\leq \text{Lip}(\Phi, \Theta) \sqrt{2} (\|B'_k - B_k\|_F + \|C'_k T - C_k\|_F) \qquad (3.1)
\end{aligned}$$

where here we used the inequality $\sqrt{x^2 + y^2} \leq \sqrt{2}(|x| + |y|)$.

Now let us define some constants M_{XY} and M_X, as they will appear commonly for the rest of the section.

$$M_{XY} := \left\| \frac{\partial^2 F_D}{\partial X \partial Y} \right\|_{L^\infty(\mathbb{R}^P)}, M_X := \left\| \frac{\partial F_D}{\partial X} \right\|_{L^\infty(\mathbb{R}^P)}$$

which are bounded by assumption. Now, using the partial derivatives from equation 2.4, we obtain

$$\begin{aligned}
B'_{i+1} - B_{i+1} &= B'_i + B_i - \theta_i \left(\frac{\partial F_{\tau(D)}}{\partial B}(B'_i, C'_i) - \frac{\partial F_D}{\partial B}(B_i, C_i) \right) \\
&= B'_i - B_i - \theta_i \left(\frac{\partial F_D}{\partial B}(B'_i, C'_i T) - \frac{\partial F_D}{\partial B}(B_i, C_i) \right).
\end{aligned}$$

Taking the norm and using the matrix chain rule, and using Proposition 3.3:

$$\begin{aligned}
\|B'_{i+1} - B_{i+1}\|_F &\leq \|B'_i - B'_i\|_F + \theta_i \left\| \frac{\partial F_D}{\partial B}(B'_i, C'_i T) - \frac{\partial F_D}{\partial B}(B_i, C_i)) \right\|_F \\
&\leq \|B'_i - B_i\|_F + \theta_i \left(\left\| \frac{\partial^2 F_D}{\partial B^2} \right\|_{L^\infty(\mathbb{R}^P)} \|B'_i - B_i\|_F + \left\| \frac{\partial^2 F_D}{\partial B \partial C} \right\|_{L^\infty(\mathbb{R}^P)} \|C'_i T - C_i\|_F \right) \\
&= (1 + \theta_i M_{BB}) \|B'_i - B_i\|_F + \theta_i M_{BC} \|C'_i T - C' i\|_F \qquad (3.2)
\end{aligned}$$

Now for C, which will require an extra step to handle the matrices T and $T^\top$, we have

$$C'_{i+1} = C'_i - \theta_i \frac{\partial F_{\tau(D)}}{\partial C}(B'_i, C'_i) = C'_i - \theta_i \frac{\partial F_D}{\partial C}(B'_i, C'_i T)T^\top$$

which implies

$$C'_{i+1}T - C_{i+1} = C'_i T - C_i - \theta_i \left(\frac{\partial F_D}{\partial C}(B'_i, C'_i T)T^\top T - \frac{\partial F_D}{\partial C}(B_i, C_i) \right).$$

Taking the norm and using the triangle inequality and the Lipschitz property, we obtain

$$\begin{aligned}
\|C'_{i+1}T - C_{i+1}\|_F &\leq \|C'_i T - C'_i\|_F + \theta_i \left(\left\| \frac{\partial F_D}{\partial C}(B'_i, C'_i T)T^\top T - \frac{\partial F_D}{\partial C}(B_i, C_i)) \right\|_F \right) \\
&= \|C'_i T - C'_i\|_F + \theta_i \Bigg(\Bigg\| \frac{\partial F_D}{\partial C}(B'_i, C'_i T)T^\top T - \frac{\partial F_D}{\partial C}(B'_i, C'_i T) \\
&\quad + \frac{\partial F_D}{\partial C}(B'_i, C'_i T) - \frac{\partial F_D}{\partial C}(B_i, C_i) \Bigg\|_F \Bigg) \\
&\leq \|C'_i T - C'_i\|_F + \theta_i (M_C \|T^\top T - I\|_F + M_{BC}\|B'_i - B_i\|_F + M_{CC}\|C'_i T - C_i\|_F) \\
&= (1 + \theta_i M_{CC})\|C'_i T - C_i\|_F + \theta_i M_{BC}\|B'_i - B'_i\|_F + \theta_i M_C \|T^\top T - I\|_F.
\end{aligned} \tag{3.3}$$

Now, let us combine (3.2) and (3.3):

$$\begin{aligned}
\|C'_{i+1}T - C_{i+1}\|_F + \|B'_{i+1} - B_{i+1}\|_F &\leq (1 + \theta_i M_{BB})\|B'_i - B_i\|_F + \theta_i M_{BC}\|C'_i T - C'_i\|_F \\
&\quad + (1 + \theta_i M_{CC})\|C'_i T - C_i\|_F \\
&\quad + \theta_i M_{BC}\|B'_i - B'_i\|_F + \theta_i M_C \|T^\top T - I\|_F \\
&= (1 + \theta_i (M_{BB} + M_{BC}))\|B'_i - B_i\|_F \\
&\quad + (1 + \theta_i (M_{BC} + M_{CC}))\|C'_i T - C_i\|_F + \theta_i M_C \|T^\top T - I\|_F \\
&\leq (1 + \theta_i (\tilde{M} + M_{BC}))(\|C'_i T - C_i\|_F + \|B'_i - B_i\|_F|) \\
&\quad + \theta_i M_C \|T^\top T - I\|_F,
\end{aligned}$$

where $\tilde{M} := \max(M_{BB}, M_{CC})$. Now we define

$$\mathcal{C} := (1 + \theta_{\max}(\tilde{M} + M_{BC})) \tag{3.4}$$

and

$$E_i := \|C'_i T - C_i\|_F + \|B'_i - B_i\|_F,$$

then use the fact that $E_0 = 0$ by the assumption of all initial parameters being 0 to obtain

$$\begin{aligned}
\|C'_{i+1}T - C_{i+1}\|_F + \|B'_{i+1} - B_{i+1}\|_F &= E_{i+1} \\
&\leq \mathcal{C}E_i + \theta_i M_C \|T^\top T - I\|_F \\
&\leq (\mathcal{C}(\mathcal{C}E_{i-1} + \theta_i M_C \|T^\top T - I\|_F) + \theta_i M_C \|T^\top T - I\|_F \\
&\leq \mathcal{C}^{i+1} E_0 + \left(\sum_{j=0}^{k} \mathcal{C}^j \right) \theta_i M_C \|T^\top T - I\|_F \\
&\leq \left(\sum_{j=0}^{k} \mathcal{C}^j \right) \theta_i M_C \|T^\top T - I\|_F
\end{aligned}$$

Continuing from (3.1)

$$\begin{aligned}|L(D,\tau(x)) - L(D,x)| &\leq \sqrt{2}\,\mathrm{Lip}(\Phi,\Theta)(\|B'_k - B_k\|_F + \|C'_k T - C_k\|_F) \\ &\leq \sqrt{2}\,\mathrm{Lip}(\Phi,\Theta)\left(\sum_{j=0}^{k}\mathcal{C}^j\right)\theta_i M_C\|T^\top T - I\|_F \\ &\leq C_{\Phi,k,F_D,\theta_{max}}\|T^\top T - I\|_F\end{aligned}$$

The bound holds with probability 1 since the only randomness in the theorem comes from the choice of datapoints in SGD, and so while $L(D,w)$ and $L(D,\tau(w))$ are both random variables, the above upper bound holds with probability 1 as $P(\theta_i \leq \theta_{\max}) = 1$ by assumption. □

Remark 3.5. *An inspection of the proof reveals that*

$$C_{\Phi,k,F_D,\theta_{max}} = \sqrt{2}\,\mathrm{Lip}(\Phi,\Theta)\left(\sum_{j=0}^{k}\mathcal{C}^j\right)\theta_{max}M_C,$$

where $\mathcal{C}$ is defined in Equation (3.4)*.*

The discussion of $\mathrm{Lip}(\Phi,\Theta)$ goes beyond the scope of this thesis, but one could search the literature (for example [21]) for more details. Because of the assumption of Lipschitz Continuity, though, we know that it must exist and be finite - and so for a small enough $\|T^\top T - I\|_F$, we have approximate impossibility. A detailed study of $\mathcal{C}$ is left as a topic of future investigation.

It is also worth noting that this case generalizes the original impossibility theorem: when the transformation τ is orthogonal, $\|T^\top T - I\|_F = 0$ and therefore we have true impossibility. Now we'll explore several types of encodings and the corresponding quantity $\|T^\top T - I\|_F$ compared to how well they generalize experimentally.

3.2 Numerical Results

We begin by reproducing some of the results from [3] in a similar framework and then by adding several more steps.

We use Keras to implement a multilayer feedforward neural network trained using Adam (see 2.2.5 for an overview and below for details). In all cases, we use an input layer of dimension $2n$, where n is the length of the encoding vector. We use ReLU as the activation function for the hidden layers and a sigmoid activation for the final layer. We decided to use ReLU as it is by far the most commonly used, though technically this precludes Theorem 3.4 from applying as it is not differentiable and as such prevents the loss function from being so. However, it is differentiable almost everywhere, and so this should not pose too large a problem. Our weights and biases are all initialized according to individual random Gaussian distributions: $\mathcal{N}(\mu = 0, \sigma^2 = 0.0025)$. An architecture with 2 hidden layers was found to achieve a good balance between accuracy and speed, and so it is used in all experiments.

The Adam optimizer (see Section 2.2.5 and [14] for further details) was used with default Keras settings: learning rate $\alpha = 0.001$ and decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

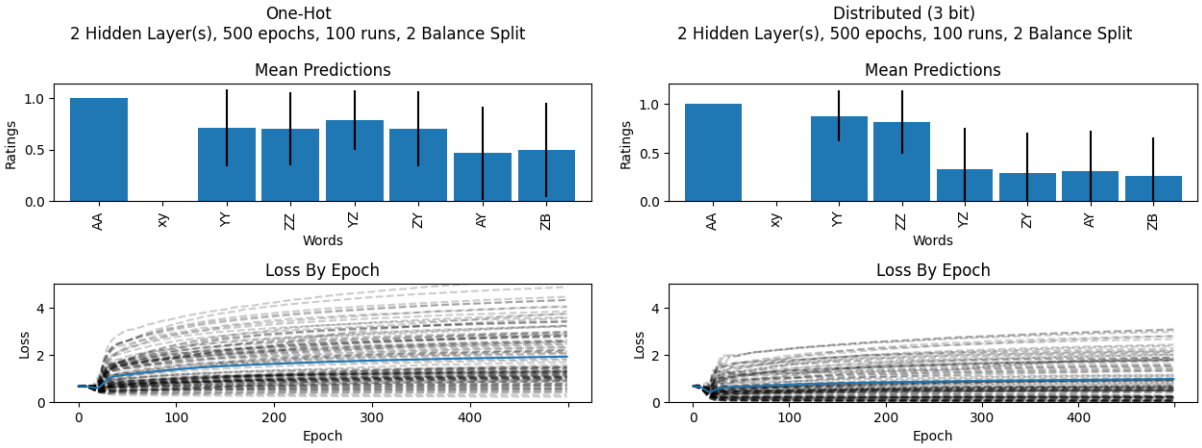

Figure 3.1: One-hot and distributed encoding results

Each figure represents the mean of 100 runs (except where specified otherwise) and is presented in the same manner. The bar graph shows the average prediction over some number of runs for some inputs, where the sample did not contain any Y or Z: AA, xy (a non-matching word the learner has seen already), YY, ZZ, YZ, ZY, AY, ZB. It is worth noting here that this set includes 2 examples from the training set, and so isn't a true validation set - but their inclusion allows for an easier visualization of the learner's predictive power. The line graph underneath shows the validation loss per epoch. The black dotted lines represent each of the runs, and the blue line represents the mean validation loss at each epoch across all 100 runs.

The validation loss on the test set over the epochs was also included, to give a general idea of the tendency to plateau (so more epochs likely would not have been useful).

Here, "Balance Split" refers to the specific words chosen to be a part of the dataset. A "Balance Split" of X means that for every "good" word (i.e., one matching the desired pattern), X "bad" words (i.e., those that do not) is added to the dataset. In all cases we present, a balance split of 2 was used, so exactly one third of the words in the dataset are good.

3.2.1 Standard Identity Effects

Here we reproduce figures from [3] and explore further into more general patterns than identity effects and into new encodings not seen in that paper.

One-Hot Encodings

In Figure 3.1, we see the predictions for YY, ZZ, YZ, ZY, AY, and ZB are all approximately 0.5. In fact, over these 100 runs the model on average predicts that YZ is most likely to be a good word. As predicted, the One-Hot encoding does not allow the learner to generalize well to new letters, in accordance with results from [3].

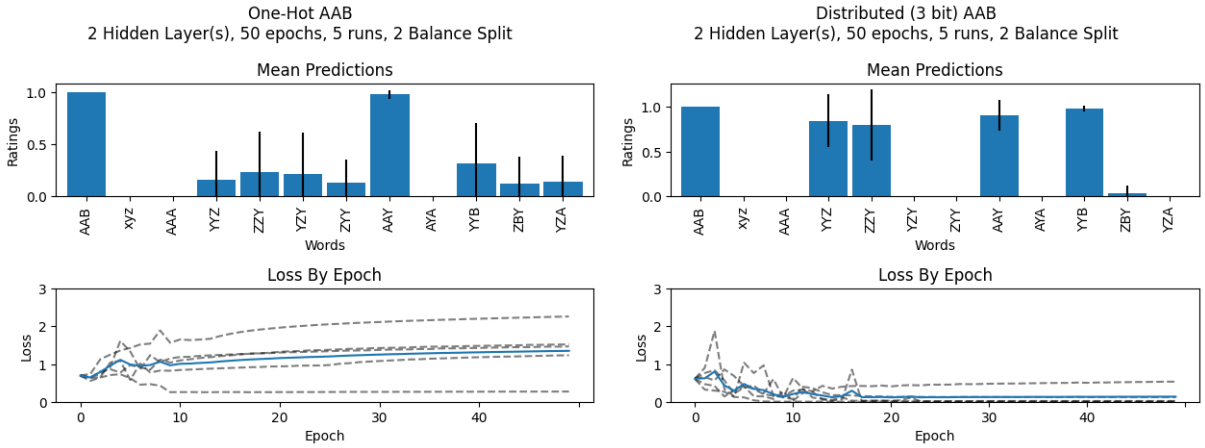

Figure 3.2: AAB one-hot and distributed encoding results

Distributed Encodings

For the 3-bit distributed encodings, again as expected, we can see in Figure 3.1 that the model does a much better job of generalizing. YY and ZZ are both high, and all the non-matching words are all given ratings under 0.5, again in accordance with results from [3].

3.2.2 More General Patterns: AAB

Up to now we have just reproduced figures from [3]. Now let us see how well the pattern continues into the AAB rule, as first discussed in Section 2.3.10. Here we use only 5 runs with each encoding type, as there are many more three letter words than two and this causes these experiments to be quite lengthy.

We see in the first plot in Figure 3.2 that in the one-hot case, as expected, the learner is unable to fully generalize to letters it has never seen before. Interestingly, it appears able to properly categorize words of repeated letters, plus an unknown letter (AAY) quite well - this seems logical, as the learner has witnessed many such words (AAB, AAC, etc) with only one such being a bad word (AAA), so its "priors" lead it to believing AAY is a good word. Still, it does a very poor job with every other good word, though marginally better with YYB than the others - maybe due to the fact that the learner has seen many words ending with B, one third of which are good.

In the second plot in Figure 3.2 we see that the distributed encoding allows the learner to generalize quite well. It seems that the AAB question seems to have very similar properties to the simple matching situation.

3.2.3 Noisy One-Hot

Now we begin what may be the most important section of this thesis - how a learner can generalize while using Noisy One-Hot encodings. As a reminder from section 2.3.5, Noisy One-Hot encodings are similar to One-Hot encodings, but each encoding has a random Gaussian vector added to it. This new encoding no longer has an orthogonal transformation matrix,

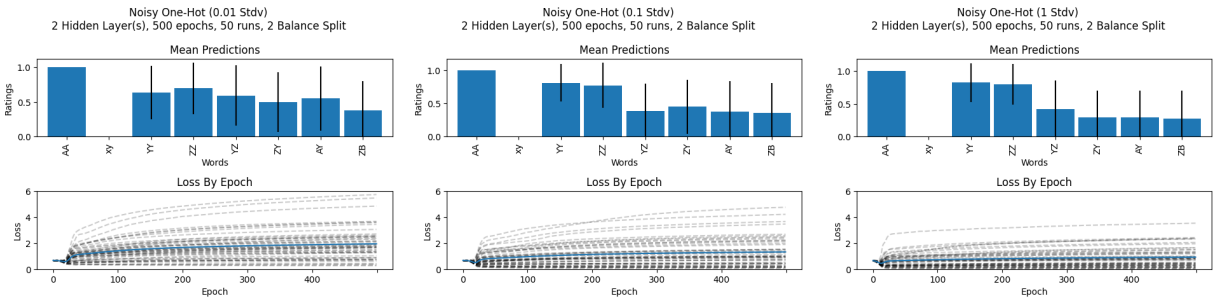

Figure 3.3: Noisy one-hot encoding results for standard deviations 0.01, 0.1, and 1

but the smaller the variance of the noise, the closer to orthogonal the transformation matrix will be. Therefore we have run multiple experiments, going through several variances - though given Keras' native language we use standard deviation (σ) instead of the variance itself (σ^2).

We see in the first second plot in Figure 3.3 that with a standard deviation of 0.1, the learner is already doing a significantly better job than with the one-hot encoding. It is not perfect, but the learner is seeing YY and ZZ as good words, and the others each have ratings below 0.5 - though ZY is coming a little too close for comfort.

The third plot shows an improvement when our standard deviation is 1, but still not at the same level as the distributed encodings we've seen previously.

We can also see in the first plot that when our standard deviation is 0.01, there is not much generalization taking place - it appears to be too similar to the normal one-hot encoding.

Although looking at each experiment one at a time is somewhat illuminating, it is also rather inefficient. Here we introduce a new type of figure: the pixel plot in Figure 3.4 shows multiple experients in one place. Each column represents one standard deviation of the noise in a noisy one-hot encoding for 50 runs. Each cell shows what rating that experiment (so a specific standard deviation of noise) assigns on average to the word on the left. The figure here shows that as the noise increases, the ability of the model to predict whether a word is good or bad also increases, as expected. It appears that there is a soft cutoff at a standard deviation of 0.1, where at $\sigma \geq 0.1$ the ratings assigned to the bad words is visibly lower than that assigned to the good words, but it appears to be a relatively smooth improvement.

In the second plot of Figure 3.4, we can see the average validation loss at the end of each experiment.

3.2.4 Other Distributed Encodings

In our 26 ambient dimension world, we have seen two distributed encodings already: 3 active bits of 26, and 1 active bit of 26 (the one-hot). Now let us see several other encodings, changing either the number of active bits or the ambient dimension.

This time we skip directly to the full prediction plots.

In Figure 3.5 we can see that as expected, the 1 active bits (one-hot) and the 25 active bits ("one-cold", see Section 2.3.11) encodings do not allow the learner to generalize well. The one-cold encoding is predictably similar to the one-hot, as swapping 1s and 0s in the initial inputs should not change the final outputs - in fact, one-cold encodings also lead to similar orthogonalities (see Appendix 2.3.11 for a more detailed study). Between the two

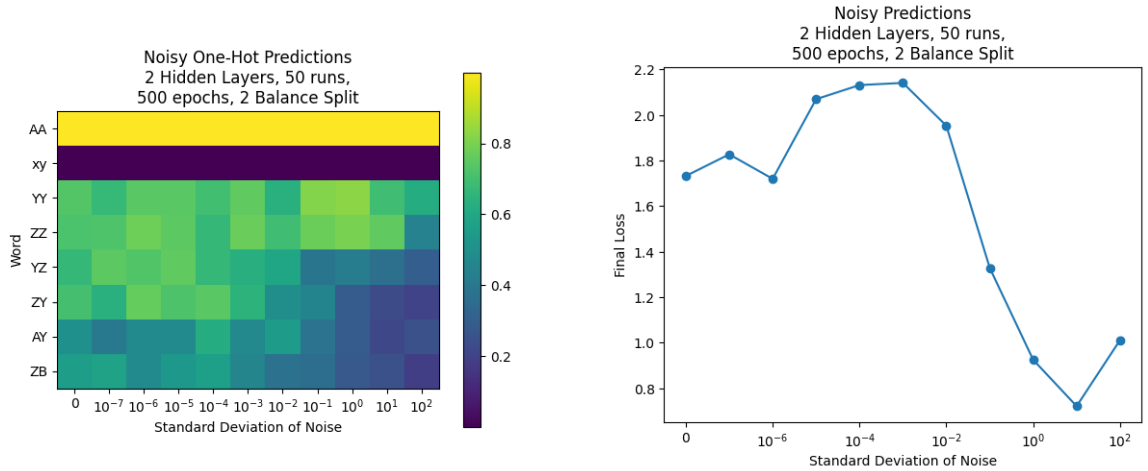

Figure 3.4: Noisy one-hot predictions

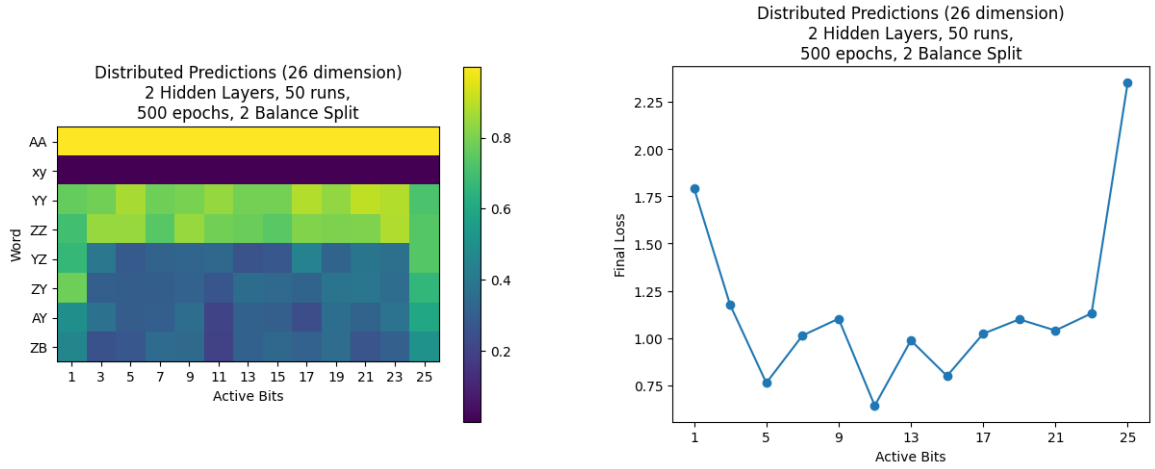

Figure 3.5: Distributed (with differing active bits) predictions

extremes, it does not appear that there is much difference in the ability to generalize from a given numer of active bits.

In Figure 3.6, we see what happens as we edit the ambient dimension, but leave the number of active bits at 3. As we have 26 letters, we require at least 7 ambient dimension (as $\binom{6}{3} = 20 < 26$, and therefore without an ambient dimension of 7 we cannot uniquely represent each letter). In this case it appears that the fewer ambient dimensions, the better the learner can generalize. This aligns with our theory, as with a smaller ambient dimension, you have more linear dependence (with the same number of active bits), which implies less orthogonality.

3.2.5 Numerical Study of Distance from Orthogonality

Now, in order to explore theorem 3.4's assertion about the connection between approximate impossibility and distance from orthogonality (that is, $\|T^\top T - I\|_F$), we will generate a

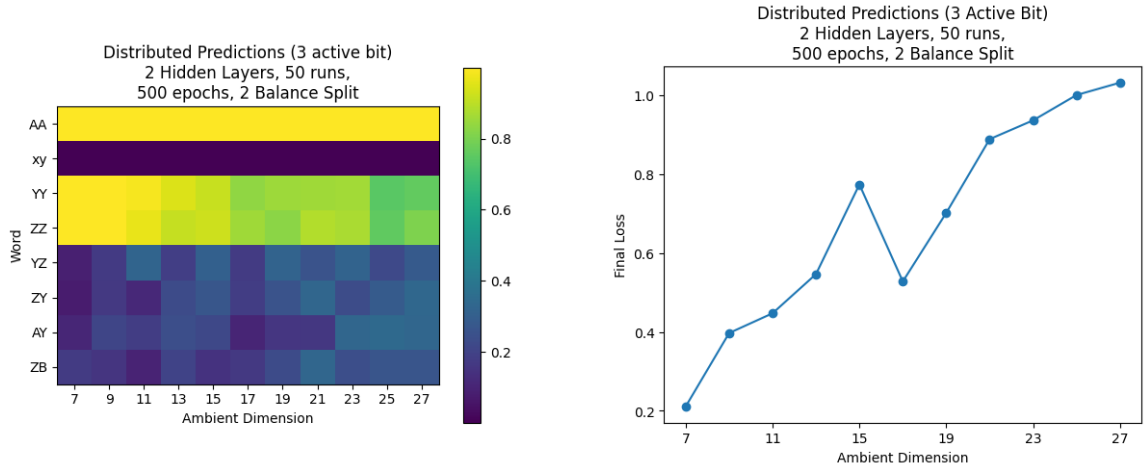

Figure 3.6: Distributed (with differing ambient dimensions) predictions

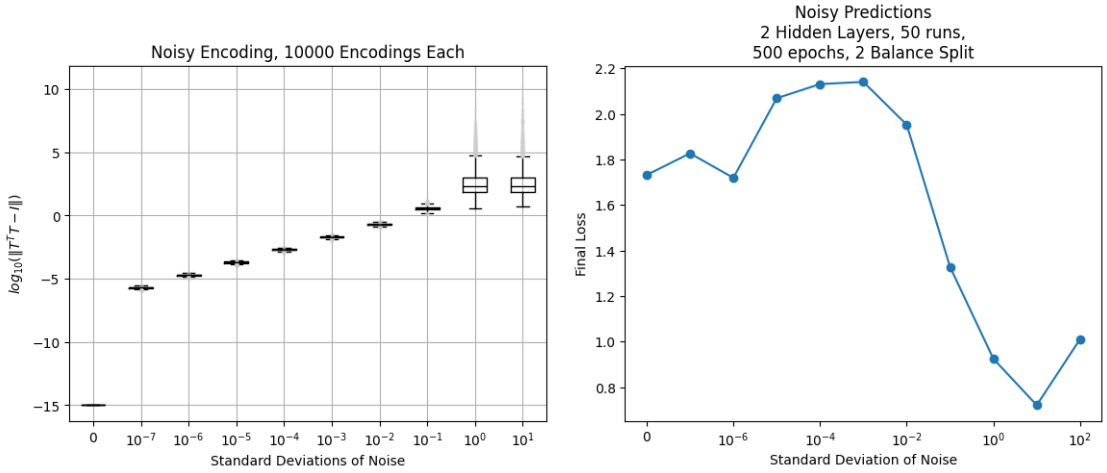

Figure 3.7: Noisy one-hot deviations from orthogonality vs validation losses

sample of encodings. Then we calculate their transformation's distance from orthogonality and compare to both mean predictions and mean final validation loss. In each plot we create a certain number of random encodings of each variable of interest (10,000 for each of the noisy encodings, 5000 for each of the distributed encodings) and calculate $\|T^\top T - I\|_F$ for each. Then we take the log of the data (as seen in [1, Appendix A.1.3]), and create box plots of the data. (To avoid singularities at $\log 0$ when plotting $\|T^\top T - I\|_F$ in a log scale, we take instead $\max(10^{-15}, \|T^\top T - I\|_F)$.

One might intuit that the higher the standard deviation of the added noise, the higher the distance from orthogonality. What we see in Figure 3.7 is that while this does appear to be the case, it only remains true until a standard deviation of 10^1 - after which the distance from orthogonality plateaus. We can see that the predictions seem to follow a similar pattern - as the noise increases, the validation loss largely decreases. After 10^1, it appears that the validation loss begins to grow - this is examined further in the Conclusion (Section 4.1, particularly Figure 4) as a potential avenue for future research.

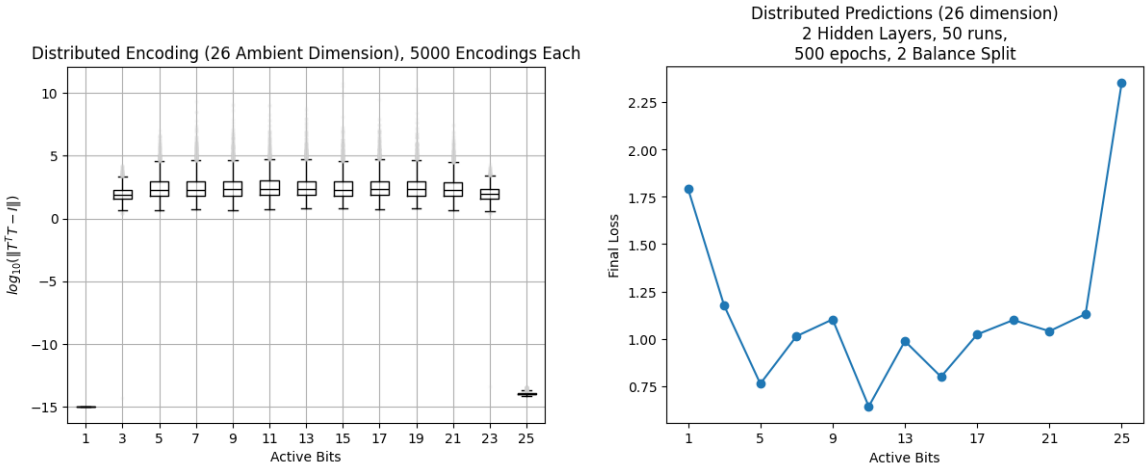

Figure 3.8: Distributed (26 dimension) deviations from orthogonality vs validation losses

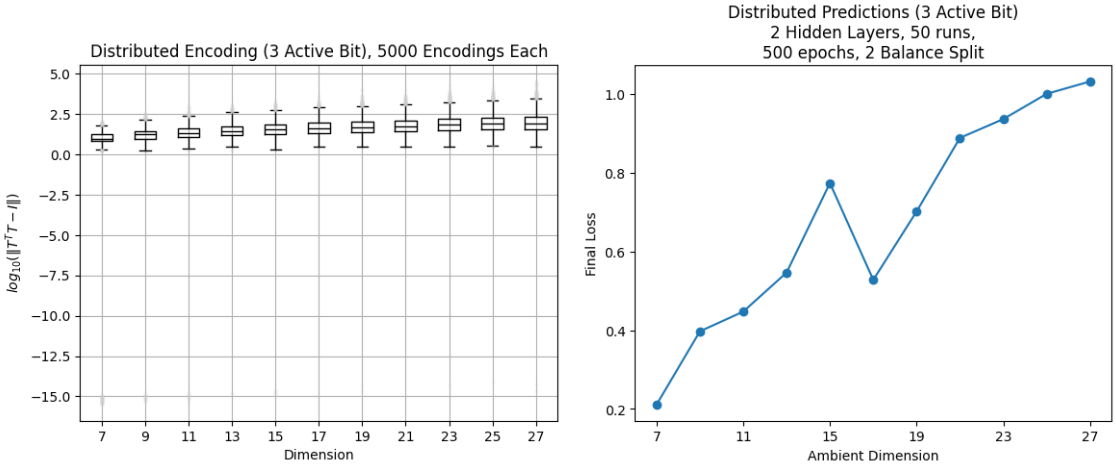

Figure 3.9: Distributed (3 active bits) deviations from orthogonality vs validation losses

Here in Figure 3.8 we see that outside of the one-hot and the one-cold case, the distance from orthogonality of the matrices appears to be fairly unchanging. This matches the validation loss: one-hot and one-cold both do fairly poorly, but all others do similarly (given that the experiments are random).

Lastly, in Figure 3.9 here we see something somewhat unexpected - although the validation loss goes up with the ambient dimension as one might expect, the distance from orthogonality does not notably go down. Note that this does not conflict with our major theorem (Theorem 3.4, but it does potentially imply there is another important relationship that is not captured by it.

3.2.6 Specific Low/High Distance Matrices

During this process, at some point it became clear that one could save the lowest and highest distance from orthogonality encodings, and then verify if they seem to follow the predictions

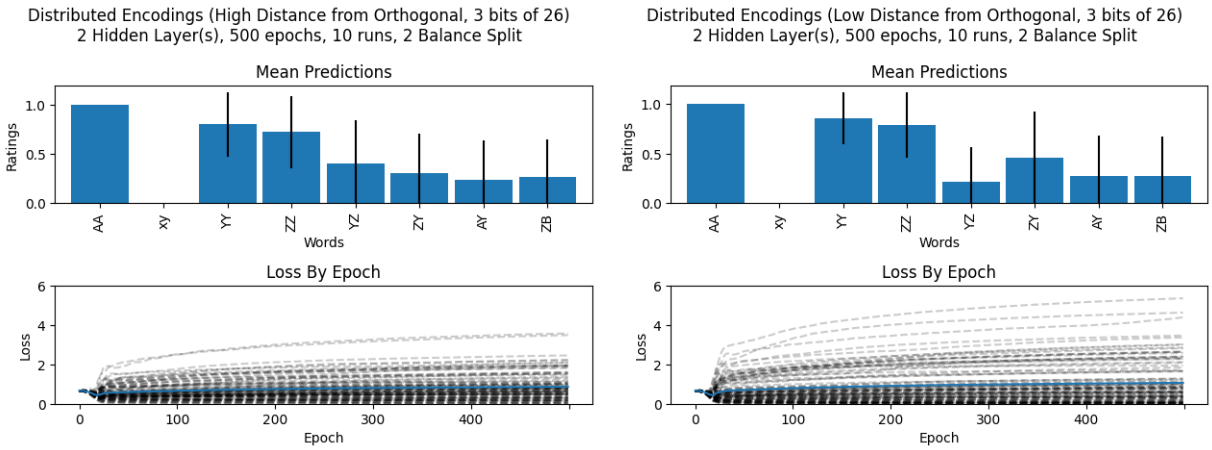

Figure 3.10: Specific encoding experiments (high/low distance)

of the theorem. There were many possible types of encodings one could have chosen, but the distributed encodings with 3 active bits out of 26 were selected as they were deemed to be the most iconic of our work. Using the same 3-active-of-26 encodings as referenced in Figure 3.8, we selected the top 10 distance from orthogonality encodings and the bottom 10, and ran experiments: 10 runs with each encoding, 500 epochs each.

We can see in Figure 3.10 that the results are not as we might have expected. Both the close matrices and the far matrices have similar validation losses and similarly poor predictions.

Fully understanding this issue is beyond the scope of this thesis, but we conjecture that it has to do with the specific transformation T. The original theorem, (Theorem 3.4), stated that there existed some transformation τ that would limit the distance between the predicted output of the learner and the true rating: it may be the case that our assumed τ is not the most effective for some of these encodings, and as such, our calculated distance from orthogonality is not the smallest possible. It is also clearly possible that our bound is simply not sufficiently tight for the distance from orthogonality to overcome the other constant in all cases. In any case, it could be a fruitful area of further research.

Chapter 4

Conclusions

It appears that identity effects in deep learning have far more to do with the way the network takes data in than anything else - specifically, which encodings are chosen. Throughout this thesis we experimented with several types of encodings (namely one-hot, distributed, and noisy one-hot), and saw that the transformation matrix τ's orthogonality is a very important indicator of identity effect impossibility. When τ is instead only approximately orthogonal, there still appears to be some degree of approximate impossibility as evidenced by Theorem 3.4, a conclusion reinforced in many experiments in Section 3.2.

That is not to say that the relationship is perfectly described through this distance from orthgonality of τ, though. There are clearly other effects in play. For true impossibility, this orthogonality is an extremely strong indicator, but "approximate impossibility", as one can tell from the title, is a much more nebulous concept. When specific low and high distance-from-orthogonality matrices were deliberately chosen, there did not appear to be any major difference in their ability to generalize, as we saw in Section 3.2.6. We also saw that the standard deviation of the noise in the noisy one-hot encodings tended to increase generalization ability up to 10^1, but after that did not perfectly correspond to the distance from orthogonality of the related τ. In fact, more experiments were run and as can be seen in Figure 4.1, the distance from orthgonality remained in the same order of magnitude, but the generalization ability of the learner dropped significantly - strongly hinting that another measure will be important to fully understand this approximate impossibility.

4.1 Open Problems

There are many open problems remaining in this study of identity effects.

A major topic for future investigation is the study of the constant $C_{\Phi,k,F_D,\theta_{max}}$ from Remark 3.5, and in particular the Lipschitz constant $\text{Lip}(\Phi, \Theta)$. Although its apparent independence from τ allows the theory to hold and have value, there could be a great deal of insight to be found within it.

We gave theory for initializers starting at 0, and while it seems intuitive that random initializations would follow the same rules, that remains to be seen through theory and experimentation.

Though we used SGD during the theory, we used Adam for the experimentation, and so

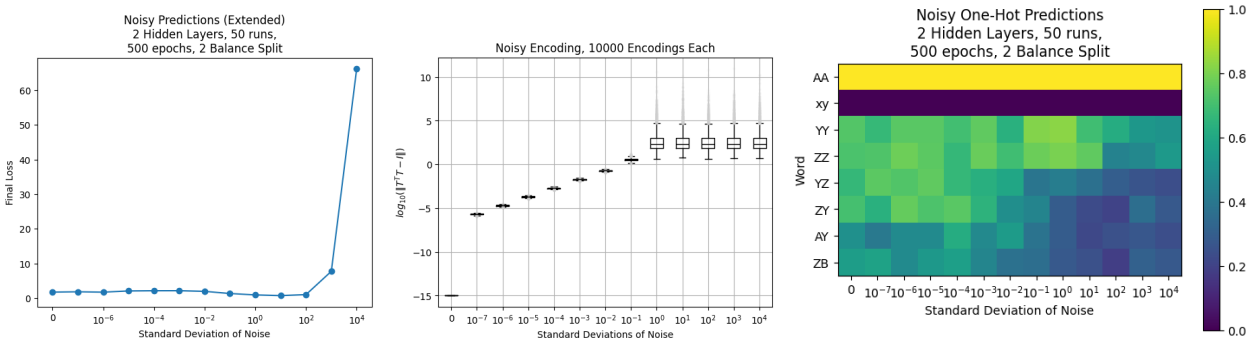

Figure 4.1: Extended noisy one-hot figures

finding a way to extend Theorem 3.4 to include Adam as an optimizer would be another logical next step.

Though we showed results for ReLU, further investigation into other fully differentiable activation functions is needed.

There are also many other types of architectures one can test. In [3], LSTM models were touched on and experimented with, and in [4], GNNs were discussed and in both papers it was found that orthgonality leads to impossibility. It remains to be explored if approximate orthogonality leads to the same approximate orthogonality as it does in our FNN theories and experiments. There are other types of neural networks such as Convolutional Neural Networks (CNNs) where it seems that no identity effect research exists, as of yet, and this could also be a fruitful next question.

We also touched briefly on patterns other than basic matching in Section 3.2.2, but there are a great deal more possibile patterns one can explore, including more complex ones. There is a great deal of research to be done to see if "identity effects" should potentially instead be known as "pattern effects", as one might intuit by examining the types of patterns that cause impossibility.

Lastly, all assumptions in this thesis included this idea of "Factorizable Models" described in Section 2.3.6. What might be found if this assumption is violated?

As we can see, though many questions were touched about the relationship between distance-from-orthogonality and approximate ratings impossibility, as always, there are many more left to explore. It also remains to be seen what the practical effects of these identity effects might be on major applications of DNNs - what hidden problems are linked to a neural net's seeming inability to generalize some major types of patterns?

Bibliography

[1] Ben Adcock, Simone Brugiapaglia, and Clayton G Webster. *Sparse Polynomial Approximation of High-Dimensional Functions*, volume 25. SIAM, 2022.

[2] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep Learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

[3] Simone Brugiapaglia, Matthew Liu, and Paul Tupper. Invariance, Encodings, and Generalization: Learning Identity Effects with Neural Networks. *Neural Computation*, 34(8):1756–1789, 2022.

[4] Giuseppe Alessio D'Inverno, Simone Brugiapaglia, and Mirco Ravanelli. Generalization Limits of Graph Neural Networks in Identity Effects Learning. *Neural Networks*, 181:106793, 2025.

[5] Charles Henry Edwards. *Advanced Calculus of Several Variables*. Courier Corporation, 2012.

[6] Attila Egri-Nagy and Antti Törmänen. The Game is Not Over Yet — Go in the Post-AlphaGo Era. *Philosophies*, 5(4):37, 2020.

[7] James Gareth, Witten Daniela, Hastie Trevor, and Tibshirani Robert. *An Introduction to Statistical Learning: With Applications in R*. Spinger, 2013.

[8] Cathy S Gelbin. The Golem: From Enlightenment Monster to Artificial Intelligence. *Bulletin of the German Historical Institute*, 69(2022):79–94, 2021.

[9] Danny Goodwin. Google AI Overviews Under Fire for Giving Dangerous and Wrong Answers, 2024. Accessed: 15 November 2024.

[10] Greff, Klaus and Srivastava, Rupesh K and Koutník, Jan and Steunebrink, Bas R and Schmidhuber, Jürgen. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2016.

[11] Larry Hardesty. Explained: Neural Networks. *MIT News*, 14, 2017.

[12] Douglas Heaven. Why Deep-Learning AIs are so Easy to Fool. *Nature*, 574(7777):163–166, 2019.

[13] Catherine F Higham and Desmond J Higham. Deep Learning: An Introduction for Applied Mathematicians. *SIAM Review*, 61(4):860–891, 2019.

[14] Diederik P Kingma. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015.

[16] Gary F Marcus. *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. MIT Press, 2003.

[17] Nestor Maslej, Loredana Fattorini, Raymond Perrault, Vanessa Parli, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, and Jack Clark. Artificial Intelligence Index Report 2024, 2024.

[18] Warren S McCulloch and Walter Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[19] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[20] Netflix. Netflix Research Areas: Machine Learning, 2024. Accessed: 14 November 2024.

[21] Moreno Pintore and Bruno Després. Computable Lipschitz Bounds for Deep Neural Networks. *arXiv preprint arXiv:2410.21053*, 2024.

[22] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning Internal Representations by Error Propagation, Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Ed. de Rumelhart and J. Mcclelland. vol. 1. 1986. *Biometrika*, 71(599-607):6, 1986.

[23] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[24] Juergen Schmidhuber. Annotated History of Modern AI and Deep Learning. *arXiv preprint arXiv:2212.11279*, 2022.

[25] Alex Shashkevich. Stanford Researcher Examines Earliest Concepts of Artificial Intelligence, Robots in Ancient Myths, 2019.

[26] Steve Smale. Mathematical Problems for the Next Century. *The Mathematical Intelligencer*, 20:7–15, 1998.

[27] Lloyd N Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 2022.