The Computational Advantages of Intrinsic Plasticity in Neural Networks

by

Nolan Peter Shaw

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2019

Author's Declaration

Abstract

In this work, I study the relationship between a local, intrinsic update mechanism and a synaptic, error-based learning mechanism in ANNs. I present a local intrinsic rule that I developed, dubbed IP, that was inspired by the Infomax rule. Like Infomax, this IP rule works by controlling the gain and bias of a neuron to regulate its rate of fire. I discuss the biological plausibility of this rule and compare it to batch normalisation.

This work demonstrates that local information maximisation can work in conjunction with synaptic learning rules to improve learning. I show that this IP rule makes deep networks more robust to increases in synaptic learning rates, and that it increases the average value for the slope of the activation functions. I also compare IP to batch normalisation and Infomax, whose family of solutions were shown to be the same.

In addition, an alternative rule is developed that has many of the same properties as IP, but instead uses a weighted moving average to compute the desired values for the neuronal gain and bias rather than the Adamised update rules used by IP. This rule, dubbed WD, demonstrates universally superior performance when compared to both IP and standard networks. In particular, it shows faster learning and an increased robustness to increases in synaptic learning. The gradients of the activation function are compared to those in standard networks, and the WD method shows drastically larger gradients on average, suggesting that this intrinsic, information-theoretic rule solves the vanishing gradient problem. The WD method also outperforms Infomax and a weighted moving average version of batch normalisation.

Supplementary analysis is done to reinforce the relationship between intrinsic plasticity and batch normalisation. Specifically, the IP method centers its activation over the median of its input distribution, which is equivalent to centering it over the mean of the input distribution for symmetric distributions. This is done in an attempt to contribute to the theory of deep ANNs.

Analysis is also provided that demonstrates the IP rule results in neuronal activities with levels of entropy similar to that of Infomax, when tested on a fixed input distribution. This same analysis shows that the WD version of intrinsic plasticity also improves information potential, but fails to reach the same levels as IP and Infomax. Interestingly, it was observed that batch normalisation also improves information potential, suggesting that this may be a cause for the efficacy of batch normalisation—an open problem at the time of this writing.

Acknowledgements

I would like to thank Tyler Jackson, who has been a helpful in the development of this work, both through helping me learn how to work with the libraries required to program my models, as well as through many helpful conversations.

I would also like to thank Masoumeh Shafieinejad, Josh Jung, and Ryan Hancock for their support throughout all the ups and downs of this research project.

Dedication

This is dedicated to the one I love:

Azathoth

Blind Idiot God, Daemon Sultan, Lord of All Things

Table of Contents

List of Figures

Chapter 1

Introduction

The study of how neural learning occurs in the biological brain has led to the development of artificial neural networks (ANNs) in computer science. The resulting research has largely focused on the ability of networks to learn through altering the strength of their synapses. This learning mechanism has a variety of different forms, such as Hebbian learning [1] and its variants [2], as well as error-based learning, particularly back propagation [3]. I refer to this family of mechanisms generally as synaptic plasticity.

However, there are other biological mechanisms in neural networks that remain far less studied. In this thesis, I aim to extend research on one such mechanism: intrinsic plasticity. Intrinisic plasticity, or IP, refers to the phenomenon of neurons modulating their firing rate in response to changes in the distribution of their stimuli. This mechanism seems to have two primary benefits for biological neural networks. The first is that it controls the energy consumption of a neuron. A biological neuron that is firing all the time consumes far more calories than a neuron with a low firing rate. This advantage is lost in ANNs, where the cost of storing different firing rates is constant. This may be why machine learning researchers have largely focused on synaptic mechanisms rather than intrinsic ones.

The second benefit is computational and founded in information theory [4]. A neuron that never fires cannot propagate a signal, but a neuron that fires all the time also fails to propagate any information about its inputs. Researchers such as Jochen Triesch have built single-neuron models demonstrating that neurons can effectively learn the ideal activation function that maximises the information potential of a neuron for a fixed mean firing rate [5, 6]. As previously noted, artificial networks are unconcerned with maintaining a low firing rate, and so a rule can be used that strictly maximises information potential. Bell and Sejnowski's Infomax rule [7] does exactly this.

Previously, Li and Li used synaptic, error-based algorithms to update the weights in ANNs in combination with local, intrinsic rules that maximise the information potential of each neuron by tuning its activation function [8]. This method, which they referred to as "synergistic learning" has demonstrated that an intrinsic update rule improves performance when used in conjunction with the error-entropy maximisation (MEE) algorithm, but their study was limited to networks with only one hidden layer. Recently, deep learning has shown that networks with multiple hidden layers can greatly improve performance across domains [9]. For this reason, it remains an important open area of research to evaluate the impact of IP on learning in deep network architectures.

In this work, I present two novel IP rules, inspired by—but distinct from—the Infomax rule. I demonstrate that these rules can improve learning when combined with an error propagating algorithm for learning weights for shallow networks, mirroring the results of Li and Li for the MEE algorithm.

I then test the impact that these IP rules have on learning in deep neural networks. The results show that IP makes deep networks more robust to increases in synaptic learning rates, indicating that this IP rule may solve the vanishing gradient problem through its influence on synaptic weight updates. I also compare intrinsic plasticity to batch normalisation (BN) [10], showing that both rules have the same family of solutions. Unlike batch normalisation, the IP rules are more biologically plausible, as they do not require a neuron to look ahead in time to adjust the activation function, nor do they perfectly shift the activation function for every distribution of inputs. I build an incremental version of batch normalisation that is biologically plausible and compare it to these rules. The results suggest that batch normalisation may work because of its impact on the information potential of neurons, rather than the previously proposed reasons of reducing internal co-variate shift [10], smoothing the loss function [11], or performing length-direction decoupling [12].

Computational Neuroscience and Machine Learning

Traditionally, computational neuroscience seeks to understand biological brains and neurological systems through the implementation of mathematical models and tools. While some researchers believe this to include only biologically plausible models, it is my view that strict adherence to biological plausibility may hinder the rate at which researchers discover the mathematical principles that underlie brain function and the connection between neural networks and theory of mind. Simplifying abstractions can behoove the understanding of large-scale network behaviour and macroscopic phenomena without the need for a perfect model of neurons found in the brain.

The work presented here is centered on the study of neural networks, and implementing mechanisms that improve the ability of a network to perform a given task. While this may be viewed more appropriately as machine learning, this research draws heavily from biological inspiration, and the methods implemented are biologically plausible wherever possible. Network properties that are not the focus of this work, such as synaptic learning rules, are thus abstracted in a manner that diminishes biological plausibility. Nevertheless, I feel that this work is best viewed through the lense of computational neuroscience, as its end is not strictly to improve how a computer may learn to solve a problem, but rather to view neural behaviour—both biological and artificial—in the context of information theory. So, rather than being solidly described as either machine learning or theoretical neuroscience, this work aims to be firmly placed at the intersection of the two.

Outline

This work will first provide the background knowledge that underlies my research in Chapter 2. I begin with Section 2.1 by reviewing the biological brain; first focusing on the fundamental computational unit, the neuron, then extending the scope of focus to networks of neurons and what properties characterise them. I then discuss artificial neural networks in Section 2.2 and Section 2.3, providing an brief overview of their history as well as the algorithms that drive their learning.

Section 2.4 then focuses more specifically on the regulatory mechanisms that locally control a neuron's rate of fire. These mechanisms, broadly refered to as "intrinsic plasticity" are central to this work, and the computational advantages provided by them are intimately related to information-theoretic principles. I then relate these mechanisms to related work in machine learning in Section 2.5, by comparing and contrasting them to the method of batch normalisation.

I then present my own contributions to this area of research in Chapter 3 and Chapter 4. I present the hypotheses I formulated, regarding the effects of intrinsic plasticity on network behaviour. I then describe the intrinsic plasticity mechanism I developed and discuss its biological plausibility. After, I describe the experiments designed to test my hypotheses. Chapter 3 performs the above experiments for a rule that has a stronger theoretical basis, while the rule presented in Chapter 4 has stronger results, but has less theoretical support.

Supporting analysis is presented in Chapter 5, where I provide proof that the IP rule presented in Chapter 3 centers the activation function over the median of its input distribution, demonstrate that the IP rule improves entropy, and expand upon the derivation of the IP rule.

Chapter 2

Background

What follows will be an overview of the key areas of research that this thesis extends. It will be assumed that the reader has a background in high school biology, a basic understanding of implementating and analysing simple algorithms, and university-level mathematics. In particular, a background in calculus, linear algebra, basic probability theory, and introductory ordinary differential equations are important for the reading of this thesis.

2.1 The Biological Brain

The skills of learning, problem solving, and cognition in most complex organisms are almost entirely attributed to the functions of our brain. It is important to note that certain intelligent organisms, such as the octopi, exhibit non-centralised, distributed nervous systems [18]. However, even in these creatures, the foundations of biological computation are largely built upon the computational unit of the neuron. Neurons are interesting in how they facilitate neuron-to-neuron communication, so it is also useful to describe the connection between neurons, called synapses. A synapse is a very small gap between two neurons that facilitates the transmission of chemicals, called neurotransmitters. The convention used here will be to refer to neurons in terms of "pre-synaptic" neurons sending signals to "post-synaptic" neurons. A more detailed explanation of neurons and synapses follows.

2.1.1 The neuron

Neurons are cells that are specialised to process and propagate electrical signals. They are comprised of three main components: dendrites, the soma, and the axon.

Dendrites, or dendrons, are protrusions from the main body of the neuron that receive stimulus from other neurons. This stimulus arrives in the form of various neurotransmitters, which are molecules sent from pre-synaptic neurons and bind to the membrane of the post-synaptic neuron, causing ion channels to open or close. These ion channels allow ions to flow both in and out of the neuron to change the neuron's membrane potential—the difference between the electrical potential inside the neuron as compared to outside the neuron. Both the dendrites and the soma integrate signals across the membrane, and both introduce non-linearities into the overall propagation of the signal [19].

The soma is the main body of the neuron. It houses various organelles, each of which performs a specialised task to maintain the healthy functioning of the cell. Such tasks include energy production, cellular blueprinting, and the production of the chemicals required to manufacture the neurotransmitters that send signals between neurons. The computational role of the soma is sophisticated and not fully understood, but can be simplified as facilitating the propagation of signals from the dendrites to the axon of the neuron. It impacts how this signal is transformed through how it integrates inputs from the dendrites, and which neurotransmitters are sent down the axon to send signals to later neurons. As signals are integrated, they may result in the membrane potential becoming more negative, referred to as hyperpolarisation, or less negative, called depolarisation. For excitatory neurons, becoming sufficiently depolarised can result in an “action potential”. This action potential results in the rapid (in the order of a few milliseconds) rise and fall of the neuron’s membrane potential. For this reason, action potentials are often called spikes.

An action potential then causes a chain reaction of action potentials that travel along the axon towards the axon terminals. At the axon terminals there are vesicles that undergo exocytosis when they receive this electrical signal. These vesicles contain the neurotransmitters produced by the soma and are released into the synapses, where they diffuse across the small gap and stimulate the dendrites of successive neurons. Often, a neuron is stimulated enough that it will generate spikes in quick succession, causing a “spike train”. The frequency of spikes occurring in this spike train are used to generate a metric of how much neurotransmitter is being sent to later neurons and is called the “activity” of the neuron. Since this is the point at which a signal is sent to the next neuron, the axon computationally acts as the output of the neuron.

In summary, a neuron receives inputs through its dendrites and integrates these inputs to adjust its membrane potential. When the membrane potential of a neuron reaches a certain threshold, the neuron generates a spike that is sent down its axon to axon terminals that output neurotransmitters. Often, these spikes occur in rapid succession, generating a spike train. The rate of this spike train is often summarised the activity of the neuron, which acts as a measure of its output.

2.1.2 The synapse

While the synapse is simply a small space that connects neurons, its role is so vital, and its behaviour so subtle, that it merits further description here.

Consider for a moment how interesting it is that neurons function by generating electrical signals that are then converted into chemical ones, via their neurotransmitters, only to be converted back into an electrical signal in the next neuron. At first glance, it seems that this introduces an unnecessary amount of complexity into the manner by which neurons communicate with one another. However, this process endows synapses with computational mechanisms that are critical to communication.

First, it allows them to use different neurotransmitters for different tasks. This allows them to effectively apply different filters to their spike trains, changing the rate that stimulus is diffused across the synapse and the integrity of this information. Second, axon terminals may flexibly adjust how sensitive a synapse is independent of other axon terminals that are all receiving the same spike train. This means that a single electrical signal can be used to send different signals through differences in the concentrations and properties of different neurotransmitters.

To demonstrate this first property, consider two different tasks that one may wish to perform: blocking a fast-moving object with one's arm versus resolving a small image at a distance (such as in an eye exam). The first is time-sensitive, and requires a quick response. Furthermore, precision is not as necessary to perform the task successfully; there are a variety of positions that may prevent the object from hitting one's face. In this instance, neurons in the corresponding region of the motor cortex may opt to use neurotransmitters that diffuse rapidly across the synapse, resulting in the signal being propagated quickly.

However, doing so also causes the resulting signal to more closely resemble the original spike train that it is integrating, so the signal will be more sensitive to noise, and thus less precise.

In contrast, when resolving the small image, precision is more important than the speed at which the image is resolved. For this task, neurons may use neurotransmitters that integrate the spike train over a longer period of time, but more closely resemble the signal that they wish to propagate.

2.1.3 Networks

Now that a description of individual neurons and synapses has been presented, it remains to be discussed how these manifest when combined into larger networks. In particular, this subsection will briefly address how neurons are organised into hierarchical structures and through what method biological networks learn.

When discussing brain architectures, it is important to note that many structural features of a brain are selected for by evolution and are thus species-specific. Since this thesis is not concerned with the physiology of the brain, but rather the emergent computational properties found in brains, only the more universal properties will be discussed. Namely, neural networks in brains are commonly

1. **Hierarchical:** Many structures in the brain are organised into layers of neurons that have connections projecting to other layers. These hierarchies enable efficient computation for complex and non-linear tasks [21].

2. **Recurrent:** Neurons are frequently connected in series that form cycles. This means that the activity of a neuron in one point in time will impact its actvity at a later point in time. This allows the brain to effectively encode dynamical systems and maintain longer term representations.

3. **Parallel:** Networks in the brain often have up to thousands of computations occurring simultaneously. This allows for a variety of benefits, such as being able to quickly compile and process multi-modal stimuli, maintain rich representations within populations of neurons, and allow for a high degree of multi-tasking when performing regulatory and control tasks for various bodily functions at once. This also allows the brain to be extremely efficient when performing computations when compared to traditional computing, as the information pipeline is much wider.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) adapt the learning and signal propogation mechanisms in the brain and use them to perform computational tasks. For decades, their use as a learning model was met with skepticism; One early model, called the perceptron, was proven to be incapable of computing even simple functions, such as the exclusive-OR (XOR) function. Even when more sophisticated, multi-layer models were introduced, it was not until the advent of the error backpropagation algorithm and more powerful hardware that ANNs saw significant use from researchers and software engineers. Now, with improved learning algorithms, more data, and more powerful, parrallelised GPUs, artificial neural networks have become the leading method for learning functions and models.

2.2.1 The Perceptron

First conceived in 1957 by Rosenblatt [20], the perceptron is a computational unit based loosely off the functioning of a single neuron. It can be formalised as a function, $\sigma_w : \mathbb{R}^n \to \{0, 1\}$, where

$$\sigma(x)_w = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

for some n-dimensional input x, n-dimensional weight vector w, and scalar b. Here, the operation $(\cdot)$ is the canonical inner/dot product for vectors.

This model is largely inspired by the function of a single neuron; x may be considered the input current/stimulus, where the dimension n of x corresponds to the number of pre-synaptic neurons that contribute stimulus to the given neuron, w corresponds to the relative strengths of the associated synapses, and b corresponds to amount of current required to induce a spike. This analogy is supported by evidence that perceptrons do in fact capture some of the behaviour seen in biological neurons—they are non-linear, have a threshold, and model the accumulation of stimuli. This model abstracts out many of the more complex mechanisms found in biological neurons, but is still capable of learning functions within a fairly large class of functions called "binary classification functions," so named because they separate inputs into one of two classes (0 or 1). Intuitively, one may think of the function that a perceptron learns as a line or hyperplane that separates a vector space.

Learning in perceptrons can be achieved with very simple algorithms. An example is simply increasing a weight if the corresponding input contributes to a correct classification, and decreasing a weight if an input contributes to a misclassification. This simplicity made perceptrons very appealing until it was shown by Minsky and Papert that perceptrons are incapable of learning a large class of functions [22]. The simplest, and most canonical of these, is the XOR function mentioned above, defined in the two-dimensional case as

$$\mathrm{XOR}(x) = \begin{cases} 1 & \text{if } x_1 \geq 0 \text{ and } x_2 < 0 \text{ or vice versa} \\ 0 & \text{otherwise.} \end{cases}$$

A proof of this result is outside the scope of this thesis, but can be found in [22]. That being said, using a "proof by picture," it is clear that there is no straight line capable of separating the two classes of XOR. When considered in relation to the brain, this is not surprising—it is not a single neuron, but rather a large network of neurons, that give the brain the computational power that it has. For this reason, perceptrons were largely ignored in the field of machine learning, as researchers began to favour other models, such as decision trees and statistical optimisation.

2.2.2 Deeper models

During the 1970's and 80's, many researchers continued to study algorithms inspired by neuroscience. While it was known that a single layer perceptron was incapable of computing even simple functions, it was quickly demonstrated that stacking multiple layers of perceptrons extended the class of functions that could be learned considerably[1] [23]. These networks are referred to as multi-layer perceptrons or feed-forward neural networks. Such networks are composed of an input layer, an output layer, and one or more hidden layers,

[1]In fact, any function can be learned with just a single hidden layer, though the number of neurons needed may be computationally intractable.

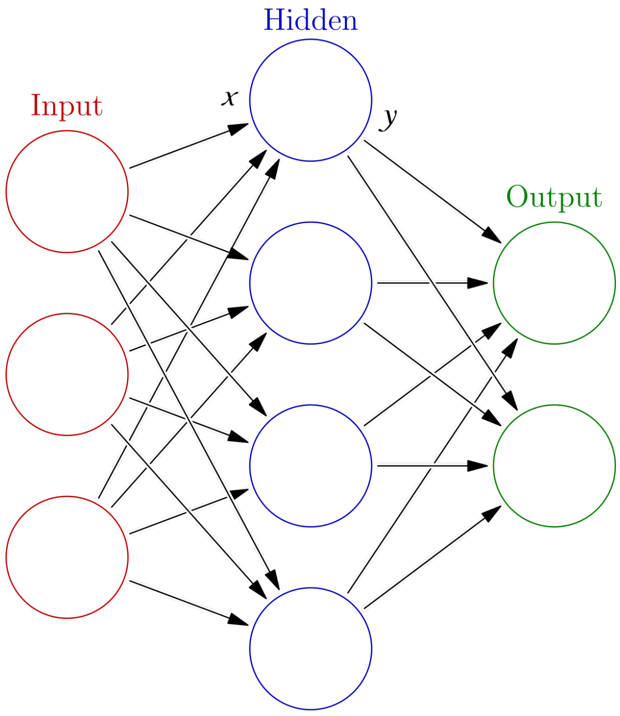


Figure 2.1: **A simple artificial neural network with one hidden layer.** Image courtesy of the Wikipedia commons. Note that x is the value going into the neuron and y is the value that is output from the neuron.

with weights connecting a given layer to the next in a hierarchical manner. An example of a simple neural network can be see in Figure 2.1.

The convention that will be used throughout this thesis is using superscripts to denote a given layer in a network, with 0 denoting the input layer and $k-1$ denoting the output layer in a network that is k layers deep. A subscript will be used to denote a particular neuron within a layer, but this level of detail will largely be unnecessary, so its inclusion will largely be foregone. For a given layer, its inputs will be denoted as x, and its outputs as y. For example, in a network with three layers (i.e. one hidden layer), $x^{(1)}$ will refer to the input into the first hidden layer, while $y_2^{(0)}$ refers to the output of the second neuron in the input layer. To maintain consistency with the notation introduced in the previous section, σ will be the "activation function" of a neuron or layer of neurons.

Since the weights exist *between* layers, they can be enumerated by either treating them as coming out of the previous layer (where $w^{(0)}$ would be the set of weights between the input layer and the first hidden layer), or as going into the next layer (where $w^{(1)}$ would be the set of weights stated above). The convention in this thesis will be to use the latter method of enumerating weights. To simplify notation, rather than discussing the weights from one layer to a particular neuron in the next layer, the connections between all neurons from one layer to the next will be denoted by the $m \times n$ matrix, W, where n is the dimension of the ingoing layer, and m is the dimension of the outgoing layer. Finally, the bias for each layer will be denoted b, with dimension equal to that of the outgoing layer, m.

It is important to note that this notation does not distinguish between singleton values and vectors. This is because it is usually unimportant to make the distinction when referring to neural networks, where the context often makes it clear. It will be stated explicitly when it is unclear whether a particular value or an entire vector is being referenced.

With these conventions introduced, an ANN can now be formalised as:

1. A set of "nodes", organised into layers, with associated inputs and outputs, x and y;

2. A set of connection weights and biases, W and b respectively, and;

3. A propagation function, σ, such that $y = \sigma(W \cdot x + b)$. This function will be applied sequentially, beginning at the first layer to compute the input to the next layer,

until the output of the final layer has been computed. It should be noted that the σ defined in the previous subsection is almost always replaced with a continuous function—often a logistic function or rectified linear function, which will be defined later.

These three components are all that is needed to define a model that is capable of computing any desired function. However, without a means of learning a given function, neural networks would only be useful for modelling predefined functions. For this reason, we also include

4. A learning rule, which specifies a means of modifying W and b to improve the performance.

2.3 Learning in Artificial Neural Networks

Machine learning, or ML for short, is the field of computer science concerned with defining algorithms that learn to perform some computational task, rather than defining the computation involved in the task. Simply put, conventional computation defines a set of known inputs, s, and some known function, f, on those inputs to generate unknown outputs, t. In contrast, ML takes, a set of known inputs, s, and outputs, t, to *learn* an unknown function, f. While much broader than the study of neural networks, this work will focus on learning algorithms that are implemented on neural network architectures. The study of neural networks with more than one hidden layer is typically referred to as "deep learning" and has gained considerable attention in the past few years after demonstrating success in problems considered previously to be intractable, such as learning to play Go [24] or how to drive a car [25].

2.3.1 Back Propagation

Of the various learning algorithms implemented in feed-forward neural networks, the most widely studied and implemented is the error back propagation (BP) algorithm [23]. Simply put, error back propagation works by providing a dataset of input-output pairs, applying the inputs to the neural network, propagating that input forward through the weights and activations of the network, and using this output along with the target output to generate an error signal with some specified loss function. Once this error signal has been computed, the gradient of the error with respect to each weight is calculated using the chain rule. These errors indicate which updates should be made to the weights to reduce the loss of the network on the given task. Thus, back propagation is essentially performing gradient descent on a high-dimensional manifold in the parameter space of the network. A detailed description of the back propagation algorithm can be found in Appendix A. The point of importance is that the gradient of the error, E, can be computed for any given connection weight in the network, w_{ij}, connecting the i-th neuron in the previous layer to the j-th neuron in the current layer. Through applying the chain rule, this gradient is expressed as

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ij}}, \tag{2.1}$$

where y_j is the output of the neuron, and $x_j = \sum_k w_{kj} y_k$ is the input of the neuron, computed by taking the linear sum of the outputs from the previous layer times the associated weights.

Each term in Equation 2.1 is fairly straightforward to compute. $\frac{\partial E}{\partial y_j}$ is typically just $(y - t)$ for the output layer. For internal layers, it is somewhat less clear, but can be computed recursively to yield

$$\frac{\partial E}{\partial y_j} = \sum_h \left(\frac{\partial E}{\partial y_h} \cdot \frac{\partial y_h}{\partial x_h} \cdot w_{jh} \right) \tag{2.2}$$

for all neurons h in the subsequent layer. The term $\frac{\partial y_j}{\partial x_j}$ is simply the slope of the activation function, $y = \sigma(x)$, alternatively denoted as $\sigma'(x)$. The final term, $\frac{\partial x_j}{\partial w_{ij}}$, is less immediate, but remember that $x_j = \sum_k (w_{kj} \cdot y_k)$. So the only term containing w_{ij} is exactly $(w_{ij} \cdot y_i)$ whose gradient is y_i. Biases are computed in a similar manner through recursion. With all the values for $\frac{\partial E}{\partial y_j}$ and $\frac{\partial E}{\partial w_{ij}}$ found for the current layer, the errors for the previous layer can now be readily found in the same manner. This is done recursively for every layer until all the gradients for each weight have been found.

Having obtained all the values of $\frac{\partial E}{\partial w_{ij}}$, the goal is to then use these gradients to improve the performance of the network. Since the objective/loss function provides a concrete measure of performance, and the gradients of that error have been found, a simple update can be made to adjust all parameters of the network such that the loss decreases. This method is called "gradient descent" and gives weight updates of

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial E}{\partial w_{\text{old}}}$$

for some learning rate, η. In general, when discussing any objective function, f, parameterised by a set of weights/values, θ, one may denote their gradients as $\nabla_\theta f(\theta)$.

It is important to note that this method of gradient descent has a number of concerns. For one, it is sensitive to the size of η. If η is too small, then the algorithm may not converge to a minimum for the error in a reasonable amount of time. What is worse, if η

is too large, then it may overshoot the desired minimum and actually cause the error to grow. This could result in increasing values for the gradient of the error, causing divergent behaviour. Another concern is that converging to a solution where the gradient of the error is zero may not mean that the ideal solution has been found. More precisely, if the manifold of the objective function is not convex, then there may be multiple local minima. Since back propagation works by descending along the gradients, the slopes around these minima cannot be climbed out of to find a better, global minimum.

It is also important to note that this process of gradient descent can be heavily influenced by the order in which samples are shown to the network, as each sample has its own optimization manifold, or "landscape." This can result in a phenomenon referred to as "overfitting," where a network's learned state is dictated more by the circumstances of training rather than the nature of the problem itself[2]. For this reason, the full training process of back propagation usually involves randomising the order in which samples are presented, and is called stochastic gradient descent. In modern implementations, much of this work is vectorised, with each tensor being composed of multiple samples for the inputs, called a "batch." However, even with a fixed batch size being passed into the network, the batches themselves will be randomised to prevent the network from overfitting during training.

[2]A much larger contributor to overfitting is actually which samples are chosen for training, and is the reason that networks are commonly validated using a separate test dataset that the network has not previously seen. The details of this topic are largely outside the scope of this research.

2.3.2 Other error propagating algorithms

Back propagation is not the only error propagating algorithm. While not of key importance to this work, two of these algorithms are outlined here. The first algorithm is the error-entropy minimisation (MEE) algorithm [17], and is included due to its use in the work by Li and Li that this research extends. The second is the widely used error propagating algorithm, Adam [13]. Adam is a variant on back propagation that uses per-parameter learning rates, and scales them in a manner that provides some "momentum" to the gradients. We provide an outline of it here, as we use it in place of traditional stochastic gradient descent.

Error-entropy minimisation, or minimisation of error entropy, differs from back prop in that it attempts to minimise the entropy[3] of the error rather than the error (typically mean squared error, or MSE) itself. This has the benefit of taking into consideration higher order moments of the error distribution when adjusting weights, where MSE only considers the first two moments of the error distribution. The precise details of how the entropy of the error is minimised is quite involved, and will not be included here, since it is not used in my work. In short, the objective is to minimise Renyi's quadratic entropy, $H_2(p)$[4], for a random variable, $\mathbf{X}$, with distribution p, given by

$$H_2(p) = -\log \int p^2(\mathbf{X}) d\mathbf{X}.$$

[3]Entropy and, more broadly, information theory are discussed in more detail in Section 2.4 as well as Appendix B. For now, it can be conceptualised as the level of uncertainty in predicting a sample from a distribution, given previously observed samples.

[4]Conventions for notation are taken from Li and Li in [8].

Hence, minimising H_2 is equivalent to maximising

$$V_2(p) = \int p^2(\mathbf{X})d\mathbf{X}.$$

This is equivalent to the expectation of the distribution $p(\mathbf{X})$ under itself, expressed as $\mathbb{E}[p(\mathbf{X})]_{\mathbf{X}\tilde{p}(\mathbf{X})}$. This can be approximated using Gaussian kernel density estimation for a specified bandwidth, and used to compute the final weight update of a weight, w, as

$$\Delta w = \eta \, \frac{\partial V_2(p(E))}{\partial w}$$

for error E and learning rate η.

A key thing to note is that minimising the entropy is invariant to changes in the mean. Hence, while all uncertainty in the error may be eliminated, the value that the MEE algorithm converges to may have non-zero error. This can be addressed by simply adding a biasing term to bring the error to zero.

While this thesis extends the work of Li and Li, who use the MEE algorithm, the synaptic learning rule in this work is the Adam algorithm. As stated above, Adam is essentially back propagation with the two added features of per-parameter learning rates and momentum. Explicitly, to optimise some set of parameters, θ, the algorithm requires a global stepsize, η, decay parameters β_1 and β_2, an initial set of parameters, θ_0, and an objective function with respect to the parameters $f(\theta)$. A time index, t, and first and second order moments, m_0 and v_0 respectively, are all initialised to zero. The main optimisation loop then consists of

1. **Increment the timestep.** t is set to $t + 1$.

2. **Compute the gradients.** The gradients of θ w.r.t. f, $g_t = \nabla_\theta f_t(\theta_{t-1})$, are then obtained (using back propagation, for example).

3. **Update the biased first-order moment.** Compute m_t using m_{t-1} and the gradient, g_t (computed above) as

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t.$$

4. **Update the biased second-order moment.** Do the same thing for v_t, but with g_t^2:

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2.$$

5. **Apply bias correction.** Transform m_t by

$$\hat{m}_t = \frac{m_t}{(1 - \beta_1^t)}$$

and v_t by

$$\hat{v}_t = \frac{v_t}{(1 - \beta_2^t)}.$$

Note that the t in the superscript is exponentiation—not some additional notation.

6. **Update the parameters.** Using the computed values for $\hat{m}_t$ and $\hat{v}_t$, update θ_t by

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

for some extremely small ϵ (10^{-8} is recommended by the original authors).

This loop is performed iteratively until some halting condition is met—usually when θ converges or some specified number of iterations/epochs has been reached.

Adam boasts many benefits. For one, it requires very little extra overhead, as the first and second order moments are the only additional parameters, excluding the added hyperparamters of the stepsize and decay rates. Furthermore, the computational overhead is similarly small, with all of the outlined steps being easily computed in an efficient manner. This makes Adam able to deal with problems that have a large number of parameters or are quite noisy. The use of exponential moving means also means that Adam is also well-suited to both online and offline learning problems. Additional benefits, and both analytic and empirical support for Adam's utility are provided by Kingma and Ba in [13].

2.3.3 The Vanishing Gradient Problem

The vanishing gradient problem is a well known issue in deep learning conerning how error is propagated back through a large network. Consider an ANN with many layers, that uses the logistic function or the tanh function for its activation function. Both of these functions start to "saturate" for large positive and negative values—that is, they are bounded at their extremes and only have large slopes for input values close to zero. During the backward phase of learning, their error will be propagated back through each layer, as presented in Equation 2.1. If the activation of the function is in these saturated regions, then the $\frac{\partial y}{\partial x}$ will be very close to zero, so $\frac{\partial E}{\partial w}$ will be close to zero. Similarly, computing $\frac{\partial E}{\partial y}$ using Equation 2.2 will result in error values close to zero (hence "vanishing") if the subsequent layer is also in the saturated region of its activation function. Hence, regardless of how large an error may be, learning may progress very slowly or not at all. This problem is compounded for each layer of saturated neurons that an error signal is propagated through, so deeper networks are more susceptible to vanishing gradients.

One method of solving the vanishing gradient problem is using rectified linear units, or "ReLUs", as an activation function rather than the more traditional sigmoidal functions. Proposed by Glorot et al. [26], ReLUs are simply the identity function that returns zero for negative inputs. More precisely, a ReLU is a function $f : \mathbb{R} \to \mathbb{R}$ such that

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{otherwise.} \end{cases}$$

Since a ReLU, f, has $f'(x) = 1$ for all positive x, ReLUs avoid saturating in the positive direction, while still maintaining the ability to compute non-linear functions. They also simplify the computation of gradients for back prop, as the vector representing $\frac{\partial y}{\partial x}$ will be just an array of ones and zeros. However, ReLUs are biologically implausible in that real neurons do saturate when exposed to large stimuli.

In contrast with ReLUs, that address the vanishing gradient problem locally by improving gradients, some researchers have suggested structural solutions. The most notable of these is the inclusion of "skip connections"—weights that connect lower layers of a neural network directly to higher layers of the network, skipping over intermediary layers. As stated above, the vanishing gradient problem is compounded when error signals are pushed back through multiple layers with very small gradients for the activation function. Thus, including connections from the bottom layers directly to the input allow the error to more directly inform the updates in those layers. A famous example of this method is ResNET [27], which has become the gold standard architecture for deep learning.

Conversely, ANNs may also face the issue of "exploding" gradients. This phenomenon poses a serious concern to neural networks, as it results in divergent behaviour. In short, when the gradients of the activation function are large *and* the weights of a network are very large, the error signal propagated backwards will grow each layer, resulting in extremely large weight updates for lower layers of the network. However, this issue is outside the scope of this work, and will not be discussed further.

2.4 Intrinsic Mechanisms in Neurons

It is well known by neuroscientists that the brain does not only change by updating the strength of synapses between neurons. These different forms of change are generally referred to as “plasticity”. The altering of synapses through either long-term potentiation or Hebbian mechanisms is broadly referred to as “synaptic plasticity”. In contrast, neuronal mechanisms that regulate the firing rate of a neuron on both short and long timescales can be referred to as “intrinsic plasticity”, or IP, and they often keep the firing of a neuron in a stable regime independent of its stimulus.

The role of intrinsic mechanisms is still an open topic of research, however there are two main benefits that have been proposed. The first is that it controls the energy expenditure of a neuron. A biological neuron with a high firing rate consumes far more calories than a neuron with a low firing rate. In ANNs, the cost of storing different firing rates is constant, and so energy constraints have no bearing. This may be why machine learning researchers have largely focused on synaptic mechanisms rather than intrinsic ones.

The second benefit is computational and founded in information theory, the study of communication first pioneered by Claude Shannon [4]. A comprehensive introduction to information theory is outside the scope of this thesis, however a brief primer is attached for the reader as Appendix B. This research focuses on studying how effective a neuron is as a unit of communication by measuring how a neuron improves its information potential, or information entropy, as it adjusts its activation function to best suit its stimulus. The information entropy, S, for a given neuron is computed as

$$S = -\sum P \log P$$

for a probability distribution P, and is a measure of how much information is observed for said distribution, on average. To illustrate, imagine a neuron that never fires. Clearly it cannot send a signal to other neurons. However, a neuron that fires all the time also carries very little information, as downstream neurons gain no information about such a neuron's input—it would fire equally often for both large and small stimuli. As such, a neuron best propagates signals when it effectively distinguishes between inputs with high and low activities.

Researchers, such as Jochen Triesch, have demonstrated that neurons can effectively learn the activation function that maximises a neuron's information potential for a fixed mean firing rate (or "energy budget") [5]. As previously noted, artificial networks are unconcerned with maintaining a low firing rate, as energy consumption is not a concern. Thus, a rule can be used that strictly maximises information potential. Bell and Sejnowski's Infomax rule does exactly this [7]. The ideal activation function turns out to be one that results in an output distribution as close to uniform as possible, as the uniform distribution possesses the highest entropy. Thus, the Infomax rule shifts and scales the activation function such that it is approximately centered over the input distribution, with a steepness that is proportional to the input distribution.

The exact rules that achieve this vary depending on the choice of non-linearity used as the activation function. For the purposes of this research, the tanh function is chosen, and for an input distribution $\mathbf{x}$, and parameters α_{IM} and β_{IM}, the transformation

$$\mathbf{y} = \tanh(\alpha_{\text{IM}} \cdot \mathbf{x} + \beta_{\text{IM}})$$

is applied with update rules

$$\alpha_{\mathrm{IM}} = \alpha_{\mathrm{IM}} + \eta \left(\frac{1}{\alpha_{\mathrm{IM}}} - 2\,\mathbb{E}[\mathbf{xy}] \right) \tag{2.3}$$

$$\beta_{\mathrm{IM}} = \beta_{\mathrm{IM}} + \eta(-2\,\mathbb{E}[\mathbf{y}]), \tag{2.4}$$

where $\mathbb{E}$ is the expected value, and η is some specified learning rate [7].

While this rule is provably optimal [7], it has a number of issues in practice. One is that it has been shown to be unstable, and will result in divergent behaviour over time [30]. Another issue is that errors in numerical approximation may result in unwanted negative values for α_{IM}, as the $\frac{1}{\alpha_{\mathrm{IM}}}$ term may fail to push α_{IM} away from zero, as would happen in the continous case.

While interesting in its own right, intrinsic plasticity is also fascinating when its effects are studied in tandem with synaptic plasticity mechanisms. Intrinsic plasticity has commonly been implemented in reservoir computing [28], where outputs are sampled from a highly recurrent network of neurons. Here, IP plays a natural role in keeping the activity of the reservoir in a stable regime.

Intrinsic plasticity has also been implemented in feedforward networks, which will be the main focus of this research. Previously, Li and Li implemented neural networks that used the Infomax rule as a local rule for regulating rate of fire, while using an error-based algorithm for updating the weights of a network [8]. Their work showed that intrinsic plasticity improves the efficiency of networks with a single hidden layer, leading to improved performance and higher information potential. In fact, they were able to demonstrate that a network with three neurons and the IP rule could outperform a regular network with as many as fifteen neurons. In this work, they studied the interaction between Infomax, which was used as a local intrinsic rule, and the error-entropy minimisation (MEE) algorithm, which was used to train the weights.

2.5 Batch Normalisation

Batch normalisation is a recent method in machine learning, developed by Ioffe and Szegedy, that normalises the input into each layer of a network such that it has zero mean and unit variance [10]. To expand, consider the input to a given layer in a neural network, $\boldsymbol{x}$. The batch norm algorithm normalises the input using the statistics of the input distribution of $\boldsymbol{x}$ for the batch, then denormalises using error-based parameters, $\boldsymbol{k}$ and $\boldsymbol{h}$, in the following way

$$\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{x}] \tag{2.5}$$

$$\boldsymbol{\tau} = \sqrt{\textsc{Var}(\boldsymbol{x}) + \epsilon} \tag{2.6}$$

$$\boldsymbol{n} = \frac{\boldsymbol{x} - \boldsymbol{\mu}}{\boldsymbol{\tau}} \tag{2.7}$$

$$\boldsymbol{x}' = \boldsymbol{k} \cdot \boldsymbol{x} + \boldsymbol{h}. \tag{2.8}$$

where ϵ is a very small value to prevent variances too close to zero when the batch size is small. This new value, $\boldsymbol{x}'$, is then used as the new input into the layer. Since this algorithm requires no information other the distribution of $\boldsymbol{x}$, it can be implemented as its own layer in a neural network, interleaved between every hidden layer in a deep network. However, it may also be viewed as an affine transformation on the activation function that is applied to $\boldsymbol{x}'$[5]. In this way, batch normalisation is capable of computing the same family

[5]In fact, it is a combination of two affine transformations. While this is equivalent to a single transformation, it is important to note that $\boldsymbol{k}$ and $\boldsymbol{h}$ are error-based parameters, learned through back propagation,

of functions as the Infomax rule, though they converge to different solutions.

Batch normalisation is such a versatile rule that it can be applied to almost any network to improve learning. Where it seems to be particularly useful is in deep ANNs, where the vanishing gradient problem causes error signals to zero out before reaching early layers in the network. This is likely due to batch norm centering inputs in the middle of the activation function, reducing the likelihood of inputs occurring in the saturated regime of the non-linearity. Many researchers have addressed the vanishing gradient problem using ReLUs (rectified linear units) as their non-linearity, however networks that use batch normalisation may use functions such as the logistic function or the hyperbolic tan function in place of ReLUs and still see learning occur where previously it would not have.

That being said, batch normalisation also improves learning in shallow networks, suggesting that the cause of its success is more than simply improving the relative size of the gradients. Initially, Ioffe and Szegedy suggested that this was due to reducing “internal co-variate shift” [10]. This means that a network does not need to learn its weights while also learning how to accommodate a broad variety of inputs. Another possible way to interpret this is to say that batch normalisation separates the tasks of learning representation and computation. Since the representations are all normalised within each layer, the only thing that the network needs to learn is how to distinguish its inputs in a manner relevant to the task.

Since its introduction, other researchers have contested the initial claim that batch norm works by reducing internal co-variate shift. In [11], Santurkar et al. designed experiments that showed batch normalisation improves performance through improving the smoothness

while $\boldsymbol{\mu}$ and $\boldsymbol{\tau}$ are computed using local statistics.

of the optimisation landscape. More precisely, batch normalisation yields smaller changes in loss and the gradients of the loss function are reduced in magnitude and made more Lipschitz.

Kohler et al. suggest that batch normalisation creates a length-direction decoupling effect, and that this improves learning [12]. That is, each parameter is reparameterised as two values, one representing the direction (in the vector sense) of the parameter, and the other representing the length. They used this to distinguish batch normalisation not just as a practical tool, but as a provably converging algorithm. However, their work only applied to learning Halfspace problems and neural networks with Gaussian inputs. For this reason, the source of batch norm's efficacy remains a somewhat open problem.

Chapter 3

A Stable IP Rule

This section will outline the work that I have done to connect and extend the ideas presented in the previous section. I will first present my initial hypotheses when first encountering the topic of intrinsic plasticity, describe how the model was extended and implemented in networks deeper than previously studied, then provide experimental data related to my hypotheses. The work done here studies the first of two rules I implemented. This rule has stronger theoretical support than the rule presented in Chapter 4.

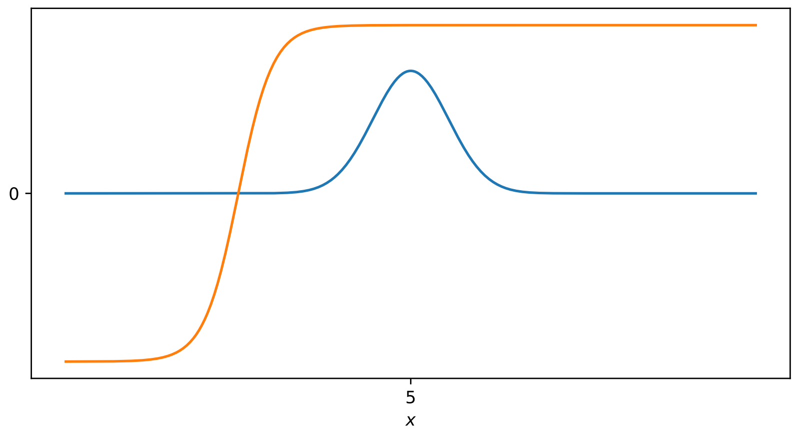
3.1 Hypotheses

Since the work by Li and Li [8] only studied networks with a single hidden layer, the effects of IP on deeper networks remains an open area of research. My first hypothesis was that implementing an IP mechanism in deep ANNs would solve the vanishing gradient problem. This is because centering the activation function of a neuron over its input distribution would naturally result in activities in the steep part of the activation function, rather than out on the wings of the non-linearity. This effect is illustrated in Figure 3.1. Since back propagation works by propagating error through the network, and one of the terms is $\frac{\partial a}{\partial x}$ (i.e. the slope of the activity with respect to its input) larger slopes will result in proportionally larger error signals.

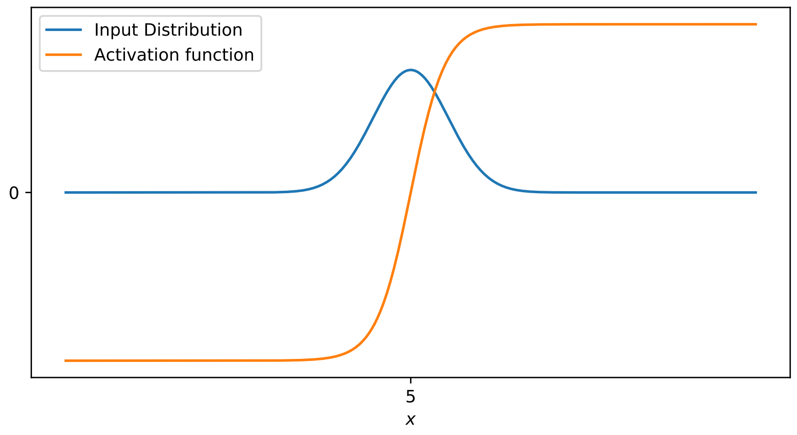
My second hypothesis was that the Infomax rule could be tailored such that an IP mechanism would not be prone to the same instabilities observed in other research [8, 30]. A natural way to achieve this would be to focus on the primary properties that Infomax has and develop a rule that could possess these properties, but be governed by simpler updates. Specifically, Infomax is characterised by centering the activation function and scaling it to match the steepness of the input distribution. As such, I aimed to develop a rule that biased the function such that it would be centered over the median[1] and scaled by $2\,\mathbb{E}[\mathbf{xy}]$—the value in the Infomax rule that keeps the output from always being in the saturated regimes of the activation function. It is important to note that these values are specific to networks where the tanh function is used as the activation function. While I only studied networks using the tanh function in my research, it would be fairly straightforward to apply these rules to other activation functions. For example, rather than biasing by the median and scaling by $2\,\mathbb{E}[\mathbf{xy}]$, a network that uses the logistic function would shift by the mean of the input, $\mathbb{E}[\mathbf{x}]$, and scale by the standard deviation $\sqrt{\mathrm{Var}(\mathbf{x})}$. This would result in an IP rule very similar to batch normalisation.

My third hypothesis was that an altered version of the Infomax rule would demonstrate behaviour similar to that of batch normalisation. Both rules work by applying an affine

[1]A proof that the IP rule I develop is centered over the median is provided in Section 5.1



$$\mathbb{E}[\sigma'(\mathbf{x})] \approx 0$$



$$\mathbb{E}[\sigma'(\mathbf{x})] >> 0$$

Figure 3.1: **Effect of the IP rule on the gradient of the activation function.** When the activation function is centered over its input distribution, the gradients of the activation function are much larger. Since error propagation multiplies by these gradients, centered activation functions will propagate larger error signals than off-center ones.

transformation to the input of the activation function, and are parametrised by two values. As such, they are capable of learning the same family of functions. Furthermore, batch normalisation works by shifting the activation function to be centered over the mean of the *input* distribution, a value that is comparable to the mean of the *activity* for a variety of input distributions and activation functions. The gain is also updated in a similar manner in both rules, with narrower input distributions resulting in steeper activation functions.

Finally, my fourth hypothesis was that a local IP rule would work in networks using error propagating algorithms besides the MEE algorithm, since Li and Li only studied the relationship of Infomax with the MEE algorithm. Specifically, I suspected that using IP would improve a network that trained weights using the Adam algorithm.

3.2 Model Design and Implementation

In this section, I start by describing the design of my main contribution: the stable intrinsic plasticity mechanism. I highlight the changes made to the implementation of the local, Infomax rule used in [7] and provide some intuition for the changes. Then, I outline the full algorithm, when implemented in conjunction with a synaptic learning rule. I then present the features of the rule that make it biologically plausible. Finally, I discuss the specific implementation used in my experiments, as well as the datasets used for testing my model.

Note that IP will henceforth refer specifically to the rule explained below, while the full term "intrinsic plasticity" will be used when discussing other intrinsic mechanisms or when discussing intrinsic mechanisms generally.

3.2.1 IP mechanism

The IP rule is implemented by taking the input into a neuron, x, and applying an affine transformation

$$u = \frac{x - \beta}{\alpha}. \tag{3.1}$$

The non-linearity is then applied in the form of the tanh function[2],

$$y = \tanh(u). \tag{3.2}$$

At the end of each feed-forward pass, α and β are then updated using the following rules:

$$\alpha = \alpha + \eta \cdot \text{Adam}(2\,\mathbb{E}[\mathbf{xy}]) \tag{3.3}$$

$$\beta = \beta + \eta \cdot \text{Adam}(\mathbb{E}[\mathbf{y}]). \tag{3.4}$$

The second term in both updates is the component that contributes to changing the IP parameters. $\mathbb{E}[\mathbf{xy}]$ and $\mathbb{E}[\mathbf{y}]$ are both computed using the input and output statistics for the current batch, and η is the learning rate for the IP rule. Adam is used to smooth the updates. The parameters, α and β, are initialised at 1 and 0, respectively.

Intuitively, α and β respectively scale and shift the activation function so that it is centered over the median of the distribution[3], with the steepness of the sigmoid being adjusted so that it is steeper for narrow distributions and shallower for wider distributions. This is done to adjust the output distribution of the neurons so that it is as close to uniform as possible, since the uniform distribution has the highest information entropy. This is the same effect illustrated above in Figure 3.1.

[2]Again, note that the IP rule can work with non-linearities besides tanh.

[3]Support for this is provided in the Chapter 5.

3.2.2 Complete algorithm for a feedforward neural network

Due to the prevalence of machine learning literature, a complete overview of the algorithm used here will omit details that are universal to neural networks. Additionally, unless it is important to specify a particular layer or neuron, subscripts and superscripts will be dropped to improve readability, and no distinction will be made between vectors of one dimension or more than one dimension.

Step 1: Initialisation

The network is constructed with the specified hyperparameters and its weight matrices initialised randomly. The α and β for each neuron are initialised to 1 and 0, respectively.

Step 2: Feedforward pass

Inputs $x^{(0)}$ are fed into the network's input layer, and the activation function of each neuron is applied

$$u^{(0)} = \frac{x^{(0)} - \beta^{(0)}}{\alpha^{(0)}}, \tag{3.5}$$

$$y^{(0)} = \tanh(u^{(0)}). \tag{3.6}$$

These values are then multiplied by the weight matrix, $W^{(0)}$, with dimensions $m \times n$, where n is the dimension of the input layer and m is the dimension of the subsequent layer, then biased by $b^{(0)}$, yielding

$$x^{(1)} = W^{(0)} \cdot y^{(0)} + b^{(0)}. \tag{3.7}$$

This is then repeated for every layer until the final, output layer, where ReLUs are used in lieu of the tanh function and the IP mechanism is not applied. The output of the network will be referred to as y_{out}.

Step 3: Update IP parameters

After the feedforward pass, the IP parameters for each neuron in the network are then updated using the update rules specified above in Equations 3.3 and 3.4. Note that the IP parameters here are updated every batch, rather than every epoch, as done by Li and Li.

Step 4: Compute error and update weights

The output of the network, y_{out}, and the target output, t, are then used to compute the error for some given loss function. Typically, the L_2 or cross entropy error are used, however this algorithm is agnostic to the choice of loss function. Since the following experiments will be comparing performance for classification tasks, cross entropy will be used for our purposes. The weights of the network are then updated using the Adam algorithm with back propagation.

Step 5: Loop or halt

This process is then repeated for each batch in the data set. Then, if the halting condition is reached (either by having sufficiently low loss or running for enough epochs), the algorithm terminates. Otherwise, the algorithm re-randomises the data set and returns to Step 2.

3.2.3 Biological plausibility

The IP rule, as implemented in the above algorithm, possesses many biologically plausible features. First, it is spatially local. The update rules for α and β only require information about the neuron's input and output, x and y. Second, the rule is temporally local and consistent[4]. The mechanism stores information about the neuron's input and output distributions using persistent parameters, and the statistics used to compute the update rules only observe a relatively small number of samples in the past. It is likely that the brain is capable of computing the described statistics through the regulated, local supply of ambient chemicals.

Furthermore, updates to the IP parameters only occur after a neuron has observed an input. This distinguishes IP from the batch normalisation (BN) method, which applies a transformation to inputs for the current batch, requiring that a neuron update its activation function prior to actually seeing some inputs. This ability to look forward in time improves the effectiveness of BN, but it is unlikely that this is biologically plausible, unless it is later discovered that dendrites are capable of performing normalisation tasks prior to changes in the somatic membrane potential. Batch normalisation also computes its bias using the *input* to a neuron, whereas our IP rule use the *activity* of a neuron to update its activation function, which is consistent with the behaviour of biological neurons. For both these reasons, the IP rule we present appears more biologically plausible than conventional implementations of BN.

While the IP mechanism possesses these biologically plausible features, it has been implemented and tested with back propagation—a learning rule that has been widely crit-

[4]By "consistent," I mean that it does not look forward in time.

icised for its biological implausibility. However, many papers have suggested that the biological implausibility of back prop is overstated [14]. Also, other algorithms that implement local learning rules have been shown to converge to the error gradients computed during back propagation [29]. For these reasons, we feel that the use of back propagation is justified, and may be treated as a simplifying abstraction of other biological mechanisms.

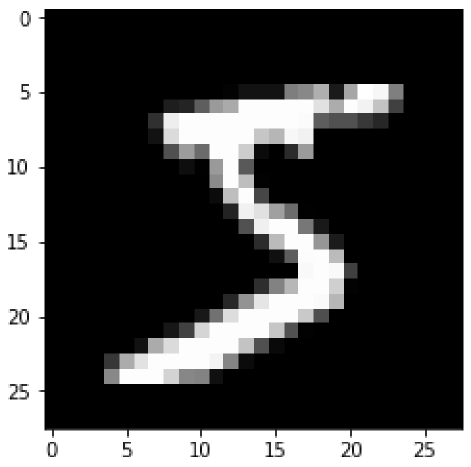
It should be noted that this computational model omits many biological features for the sake of simplicity and more clearly illustrating how the IP rule operates in isolation. I do not claim that the algorithm replicates actual brain function.

3.2.4 Implementation

The networks used in the following sections were built in Python using the pytorch package. A github repository containing the code used in this project can be found at `https://github.com/Shawfest/ip`.

3.2.5 Data sets used

The two data sets used to run the following experiments are the MNIST database of handwritten digits [15], and the CIFAR-10 data set [16], which consists of images from ten classes of objects. Examples images for each of these data sets are shown in Figure 3.2.


(a) A hand-written five in MNIST.

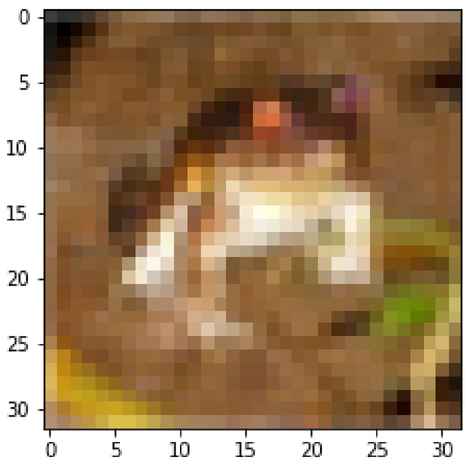

(b) A frog in CIFAR-10.

Figure 3.2: **Example inputs used for experiments.** The above images are two inputs from the MNIST and CIFAR-10 datasets.

3.3 Experimentation

The goal of my work was not to design a competetive algorithm, but rather to test the effect on learning of the IP rule in isolation. For this reason, I chose a basic, fully-connected feedforward architecture for all experiments.

3.3.1 Effects of IP when using back propagation

The easiest hypothesis to test was the fourth hypothesis—that the benefits of intrinsic mechanisms were not restricted to the MEE algorithm. Furthermore, since the IP rule is novel, testing it on a shallow network to mirror the studies done by Li and Li would help establish a foundation that the IP rule can be beneficial. This test was also done to ensure that IP actually works.

A network with the IP mechanism and a standard network without IP were trained on MNIST. Both networks have fully connected layers, with the input layer having 784 neurons to match the size of the MNIST digits, the hidden layer having 50 neurons, and the output layer having 10 neurons—one for each class of digit. For the sake of fair comparison, the weight matrices for both networks were initialised to the same values. The synaptic learning rate for the Adam algorithm was set to 0.03, and the intrinsic learning rate, η, was set to 0.0001. This experiment was run ten times for different weight matrix initialisations, with the results averaged and presented in Figure 3.3.

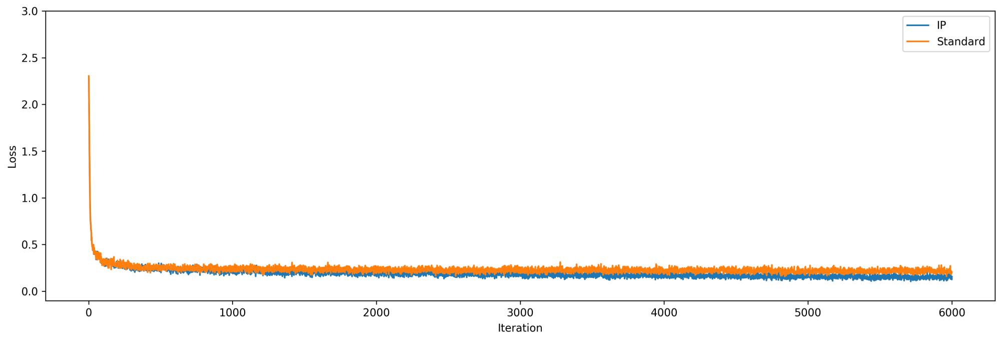


Figure 3.3: **Learning curves for shallow networks.** The averaged learning curves for both IP and standard networks trained on MNIST across 20 epochs. Observe that the IP networks achieve higher performance after training than their standard counterparts.

The same experiment was run for the CIFAR-10 data set with the input layer size changed to 3072 to match the three colour channels of the 32×32 images. To account for the increased difficulty of the problem, both networks had their synaptic learning rates turned down to 0.0001 and were given 150 neurons in their hidden layer. These results can be seen in Figure 3.4.

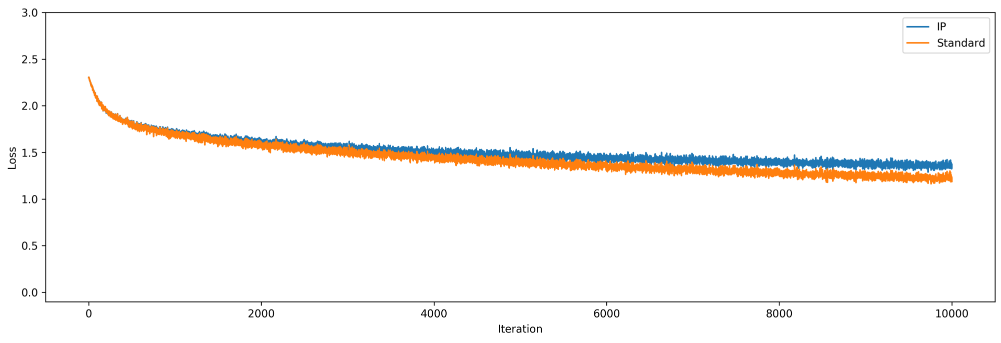


Figure 3.4: **Learning curves for shallow networks on CIFAR-10.** The averaged learning curves for both IP and standard networks trained on CIFAR-10 across 40 epochs. Here, the IP rule fails to improve the performance over a standard network.

The results in these experiments are mixed, with a small improvement in performance on MNIST, and worse performance than a standard network on CIFAR-10. In both cases, a network with IP is still capable of learning, so at least IP is capable of working with back propagation rather than just MEE shown in the Li and Li work. Also, it is worth noting that the improved performance on MNIST is likely not due to improved gradients, since there is only one hidden layer. Rather, it is likely that the improved performance is due to the improved efficiency of the neurons, as observed in [8].

3.3.2 IP improves learning in Deep ANNs

Having shown that IP improves learning in shallow networks, I then tested my first hypothesis: that the IP mechanism improves learning in deep ANNs and is robust to increases in synaptic learning rates.

To test this hypothesis, I designed a series of experiments that would compare IP networks to standard networks for various synaptic learning rates. Inspired by the experiments in [10], I suspected that, while an improvement in learning would be seen for small synaptic learning rates, the benefits of IP would become clearer for large synaptic learning rates. This is because standard networks tend to fail due to divergent behaviour and having activities stuck in the saturated regimes of the activation function.

Like the previous experiments, the networks tested were fully connected but with seven hidden layers rather than one. The results for MNIST are presented in Figure 3.5. As suspected, IP provides a small improvement in learning when the synaptic learning rate is small, but becomes more pronounced as the synaptic learning rate is increased. A standard network will slowly lose performance and eventually diverge as the synaptic learning rate grows, but a network with IP will continue to learn effectively for a longer period.

The results on CIFAR-10 shown in Figure 3.6 are less clear, but maintain a similar pattern. For very small synaptic learning rates, the standard network actually outperforms the network with IP. However, once this learning rate is turned up, a standard network begins to perform comparably worse than the IP rule.

As indicated by Ioffe and Szegedy, further research is required to confirm that robustness to increased learning rates indicates better gradients. To support our hypothesis that

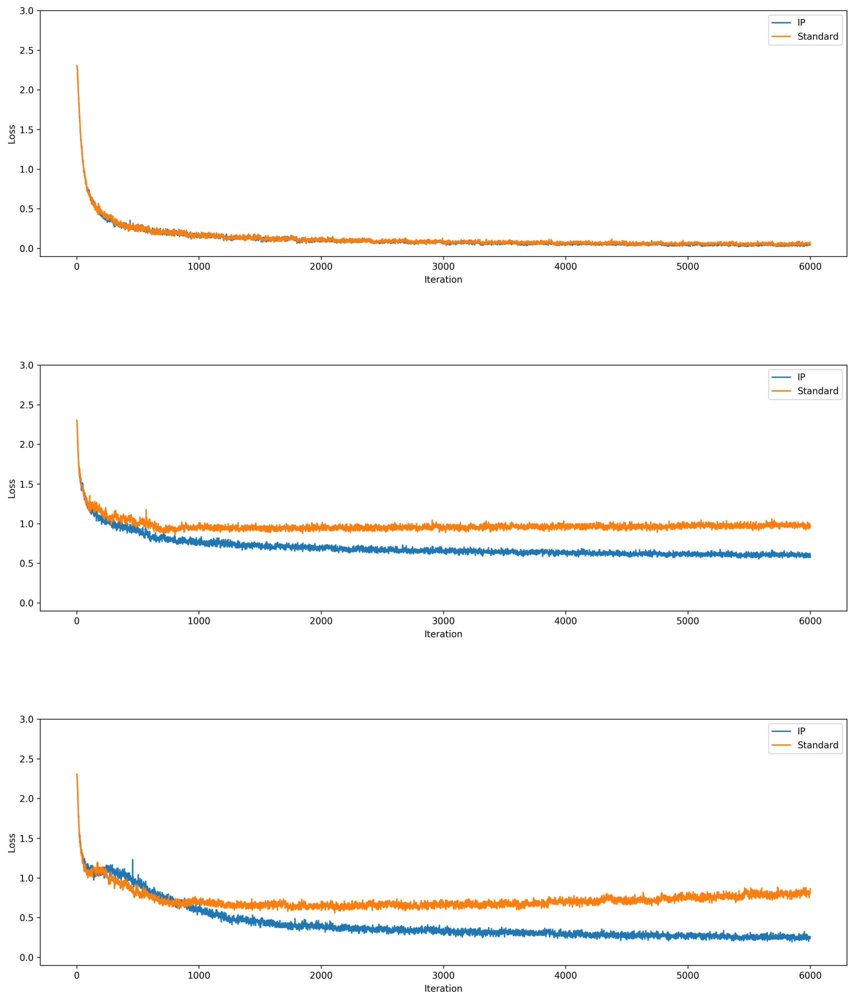


Figure 3.5: **Learning curves for deep networks on MNIST.** The averaged learning curves for both IP and standard networks trained on MNIST across 20 epochs. The synaptic learning rates for each are, in order, 0.003, 0.01, 0.012.

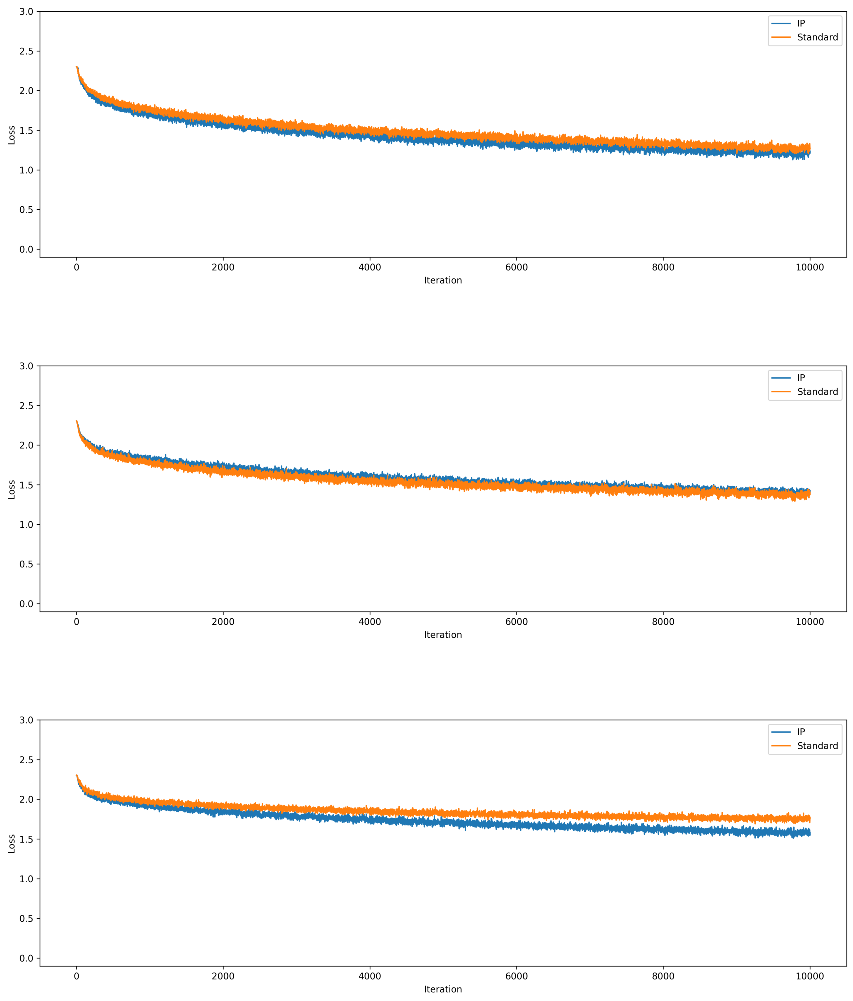


Figure 3.6: **Learning curves for deep networks on CIFAR-10.** The averaged learning curves for both IP and standard networks trained on CIFAR-10 across 40 epochs. The synaptic learning rates for each are, in order, 0.0006, 0.001, 0.0013.

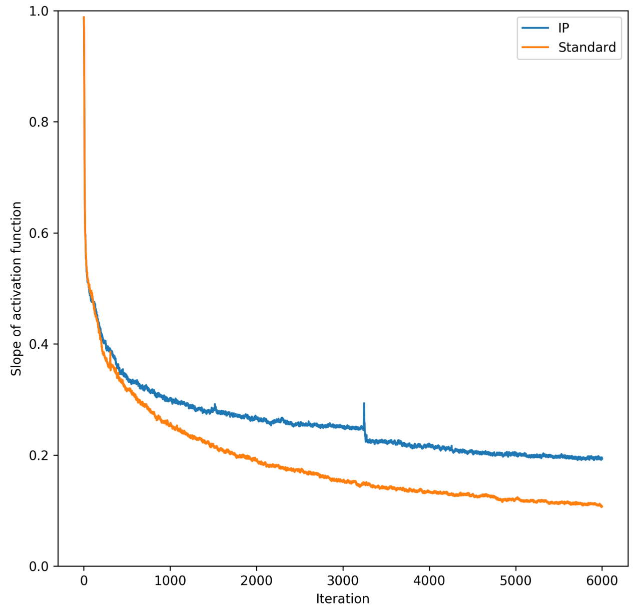


Figure 3.7: **Value of activation gradients.** The graph shows the average value of $\frac{\partial y}{\partial u}$ for a particular layer during training. The fourth layer of the network (i.e. the third hidden layer), was chosen. As you can see, the gradient of y when IP is implemented is much larger than a standard network over the course of learning.

the network is improving error propagation through larger slopes in the activation function, we re-ran the MNIST experiment with the synaptic learning rate set to 0.005, and recorded the values of $\frac{\partial y}{\partial u}$ in an intermediary layer of the network. The results are shown in Figure 3.7. Furthermore, these results indicate that even if the IP rule does not directly solve the vanishing gradient problem, it still greatly improves the stability of a network across learning rates.

3.3.3 Comparing IP to the original Infomax rule and batch normalisation

Having demonstrated the effects of IP on deep ANNs, I then compare the IP rule to the original Infomax rule. This was done to test the second hypothesis that a more stable version of the Infomax rule could be implemented. Additionally, as stated in my third hypothesis, the IP rule bears striking similarity to batch normalisation. Both rules are applied as affine transformations on the input, parameterised by two values. For this reason, both rules are capable of learning the same families of functions. However, unlike IP, batch normalisation works by transforming its input to each hidden layer using the statistics of the current batch, rather than storing this information in persistent parameters. For this reason, batch normalisation has the biologically-implausible advantage of allowing nodes to update their gains and biases perfectly and before observing their input.

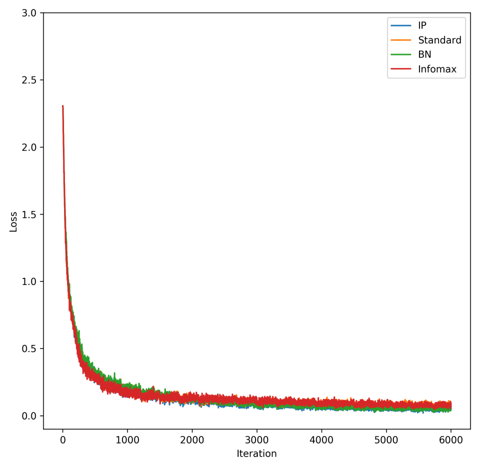
To conduct a fair comparison of IP to batch normalisation, I implement BN in the same incremental manner as IP. The feedforward phase remains the same, but the update rules governing α and β are

$$\alpha_{\mathrm{BN}} = \alpha_{\mathrm{BN}} + \eta \cdot \mathrm{Adam}(\sigma[\mathbf{x}]) \tag{3.8}$$

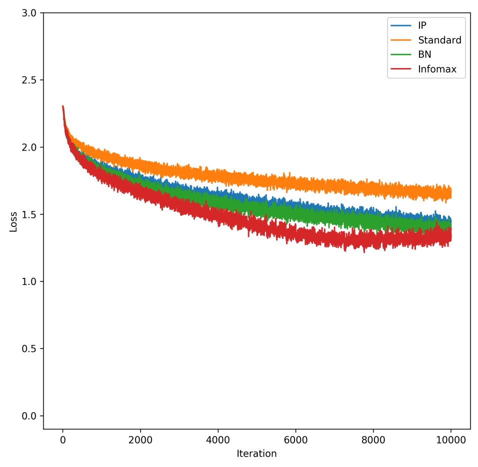
$$\beta_{\mathrm{BN}} = \beta_{\mathrm{BN}} + \eta \cdot \mathrm{Adam}(\mathbb{E}[\mathbf{x}]) \tag{3.9}$$

where $\sigma[\mathbf{x}]$ is the standard deviation of $\mathbf{x}$.

The results of comparing the IP rule to Infomax and BN are shown in Figure 3.8. Overall, IP showed only a slight improvement in performance as compared to the Infomax rule, which fails to support the hypothesis that the IP rule is more stable than Infomax. Also,



(a) MNIST learning curves.



(b) CIFAR-10 learning curves.

Figure 3.8: **Learning curves for deep networks using Infomax, IP, and BN.** For this experiment, all three local rules had the same intrinsic learning rate of 0.0001. Again, 10 experiments were done with the results averaged.

the incremental version of BN outperformed a standard network and showed performance comparable to that of the IP rule.

3.4 Discussion

Overall, the above results are fairly tepid. The experiments for shallow networks showed that the local IP learning rule is compatible with synaptic weight updates, though improved performance was only seen on MNIST. The results for deep networks were more promising, with improved performance being seen for higher synaptic learning rates. Furthermore, Figure 3.7 confirms that larger gradients for the activation function are being observed over the course of training. This supports the first hypothesis of this work—that information maximising rules can address the vanishing gradient problem—but the results are not overwhelmingly convincing.

Additionally, I was unable to show that the IP rule is more stable than Infomax. Further testing is required to see if Infomax eventually diverges, if training continues for a long enough time.

Chapter 4

Weighted Decay Learning Rule

Over the course of searching fo a stable IP rule, I experimented with many different rules. One of the first was simply taking the weighted moving average for each of α and β as follows

$$\alpha = (1 - \eta) \cdot \alpha + \eta \cdot 2\,\mathbb{E}[\mathbf{xy}] \tag{4.1}$$

$$\beta = (1 - \eta) \cdot \beta + \eta \cdot \mathbb{E}[\mathbf{y}]. \tag{4.2}$$

This rule is appealing in that it does not use Adam to smooth the updates of $2\,\mathbb{E}[\mathbf{xy}]$ and $\mathbb{E}[\mathbf{y}]$, however it has a significant theoretical issue of not being able to center itself over distributions whose medians are outside of $[-1, 1]$. To see this, assume a fixed value for η. The tanh function always has an output in the range of $[-1, 1]$, so $\mathbb{E}[\mathbf{y}]$ is bounded between these values. Hence $|\eta \cdot \mathbb{E}[\mathbf{y}]| \leq |\eta|$. If the second term is less than η, then values $|\beta| > 1$ are not possible equilibrium points.

Nevertheless, in the process of experimentation, the above rule demonstrated greatly improved results over all other methods. Not only did it learn far more quickly during training, it also converged to better solutions. The same experiments conducted in Section 3 were conducted for this local rule and the results are presented below. This rule will be referenced as "weighted decay" or "WD", for short. Note that the only change in parameters from the experiments in Chapter 3 was increasing η to 0.1 from 0.0001, since the weighted decay rule is far more stable than the IP rule for large values of η.

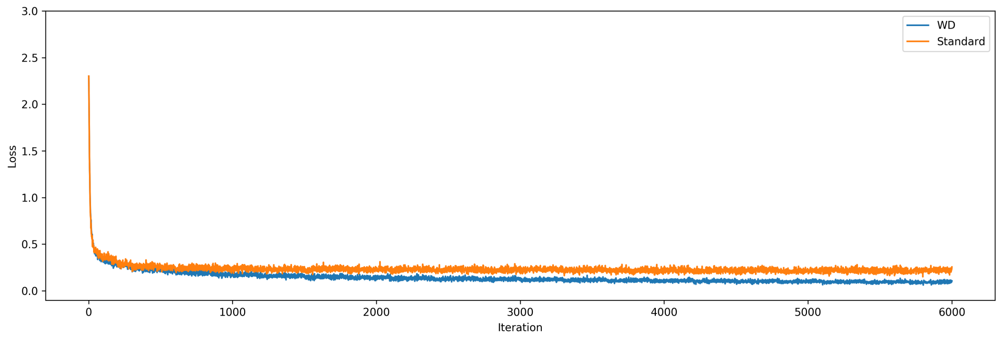
4.1 Performance in shallow nets

The first test done was evaluating the effect of the WD rule in a network with a single hidden layer. The parameters for the network are identical to the ones in Section 3.3.1, aside from the intrinsic learning rate. The results are presented in Figure 4.1.

The results on MNIST are very noticeable, with around a quarter of the average training loss for a network with WD as compared to a standard network. The results on CIFAR-10 are even more compelling. In particular, the networks with the WD rule are able to learn extremely fast at first, as compared to their standard counterparts. As mentioned when reporting the results for the IP rule on shallow networks, it is unlikely that the improved performance is caused by larger gradients rather than improved neuronal efficiency.

4.2 Performance in deep nets

Having demonstrated the improved efficacy of the WD method in shallow networks, I then replicated the test done for deep networks. Figure 4.2 and Figure 4.3 show these results.



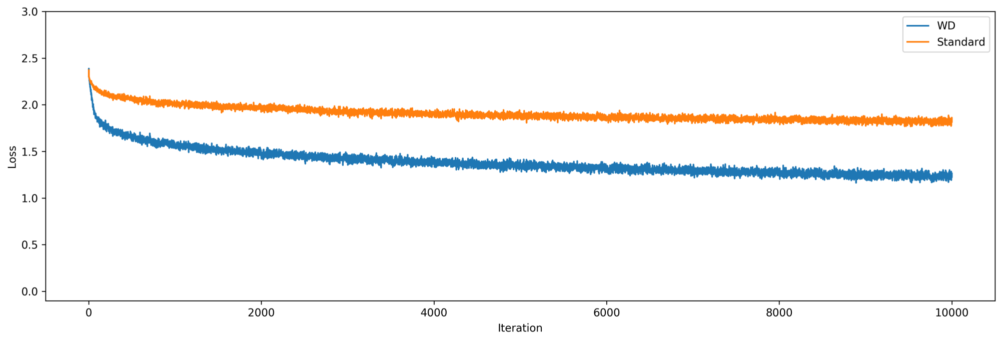


Figure 4.1: **Performance of WD rule on MNIST and CIFAR-10.** The results of training a single-hidden layer network with the local WD rule on both MNIST and CIFAR-10.

Note that in both experiments, a change was made to the final synaptic learning rate, changing it from 0.013 to 0.015 on MNIST, and from 0.0013 to 0.002 on CIFAR-10. This was done to highlight that WD seems to be even more robust to increases in synaptic learning than the IP rule.

The results for this experiment support the hypothesis that intrinsic plasticity can improve synaptic learning through solving the vanishing gradient problem. First, initial learning on both MNIST and CIFAR-10 is far quicker than found in a standard network. Second, networks with WD converged to lower losses than their standard counterparts in almost all the tests (MNIST seems to be simple enough that almost any deep network will converge to low losses). Finally, Figure 4.4 shows that the gradients of the activation functions are much larger than those found in standard networks. This strongly supports the hypothesis that local, intrinsic rules can help solve the vanishing gradient problem.

4.3 Comparison to other methods

Like its IP counterpart, I compared the WD method to both Infomax and batch norm. However, to test batch norm in a manner comparable to the WD method, a weighted decay version of batch norm was implemented with update rules

$$\alpha_{\text{BN}} = (1-\eta)\cdot\alpha_{\text{BN}} + \eta\cdot\sigma[\mathbf{x}] \tag{4.3}$$

$$\beta_{\text{BN}} = (1-\eta)\cdot\beta_{\text{BN}} + \eta\cdot\mathbb{E}[\mathbf{x}] \tag{4.4}$$

in lieu of the incremental version presented in Section 3.3.3.

The results for MNIST and CIFAR-10 in Figure 4.5 tell similar stories. The incremental version of batch norm and our IP rule both outperform standard deep neural networks on

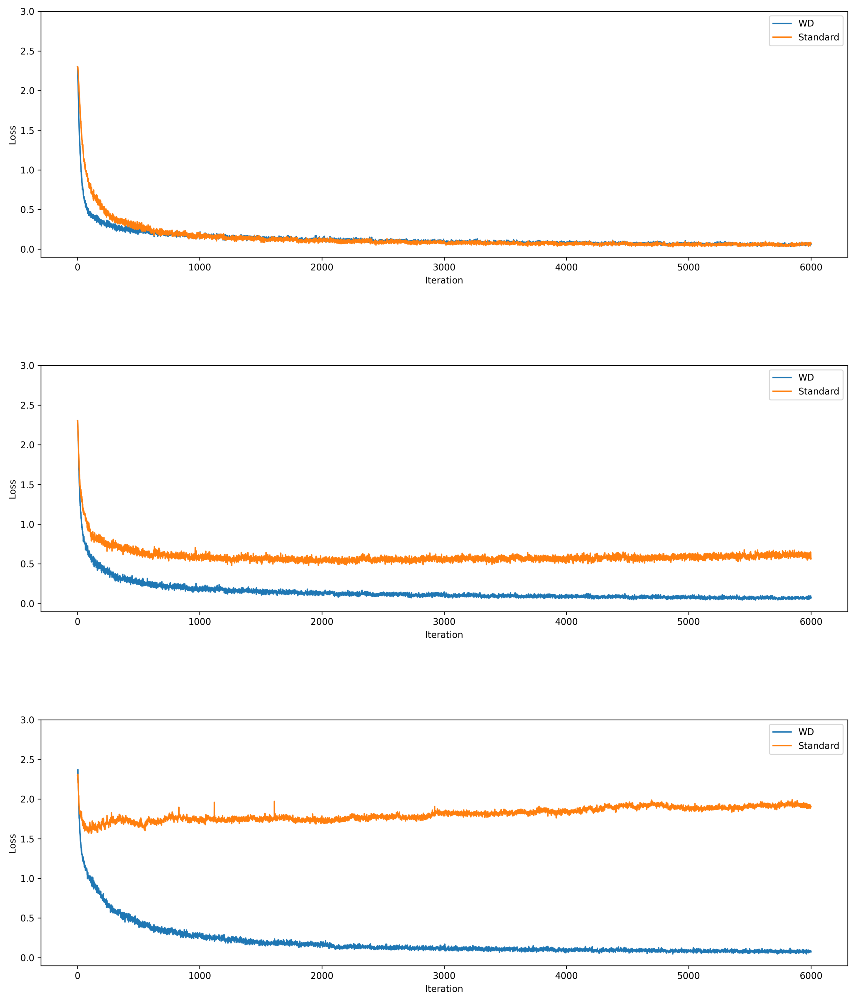


Figure 4.2: **Learning curves for deep networks on MNIST.** The averaged learning curves for both IP and standard networks trained on MNIST across 20 epochs. The synaptic learning rates for each are the same except for the last subfigure, whose learning rate is 0.015. This final learning rate was set higher to highlight the robustness of the WD method

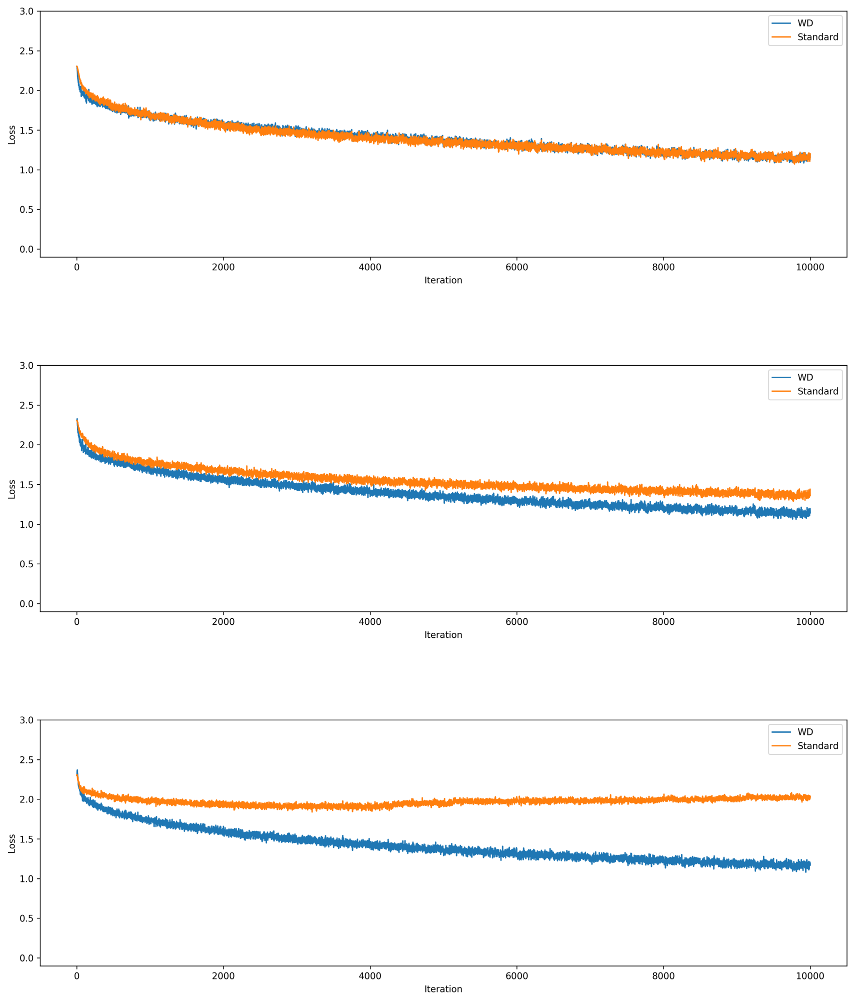


Figure 4.3: **Learning curves for deep networks on CIFAR-10.** The averaged learning curves for both IP and standard networks trained on CIFAR-10 across 40 epochs. The synaptic learning rates for each are the same except for the last subfigure, whose learning rate is 0.002. This final learning rate was set higher to highlight the robustness of the WD method.

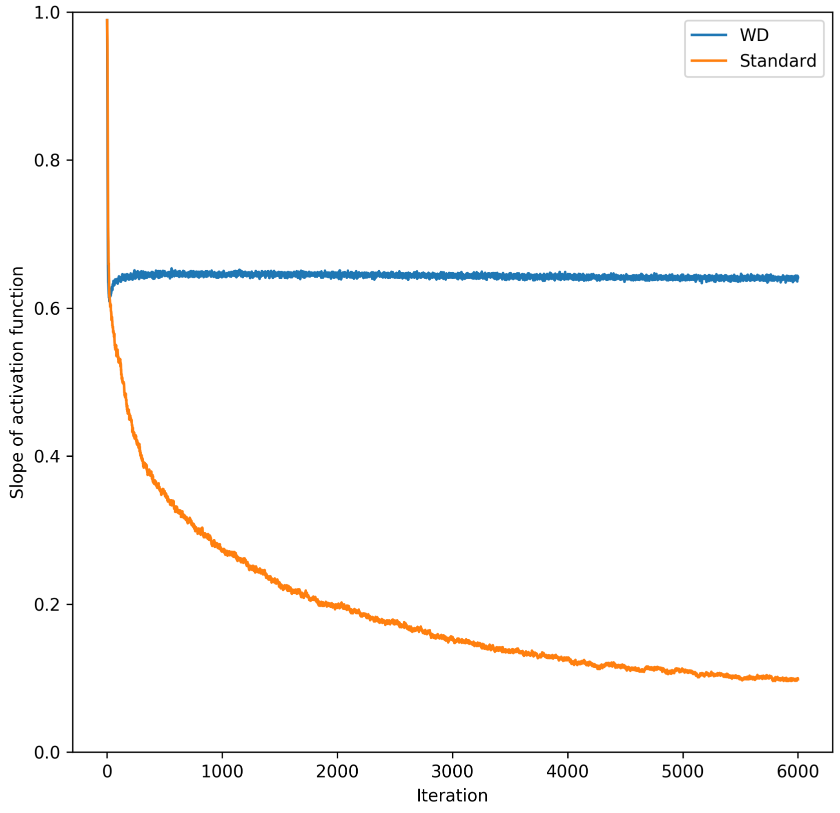
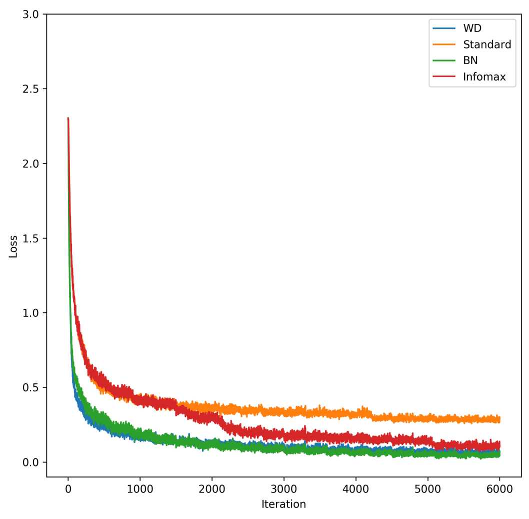
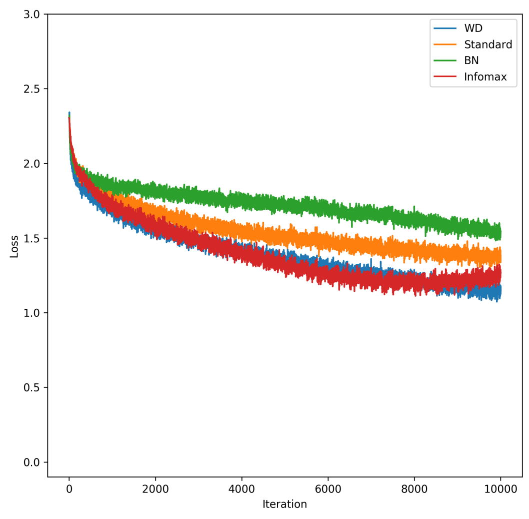


Figure 4.4: **Value of activation gradients.** This graph shows the average value of $\frac{\partial y}{\partial u}$ for a particular layer during training with WD. The fourth layer of the network (i.e. the third hidden layer), was chosen again. The gradient of y when WD is implemented is much larger than both IP networks and standard networks over the course of learning.



(a) MNIST learning curves.



(b) CIFAR-10 learning curves.

Figure 4.5: **Learning curves for deep networks using Infomax, WD, and BN.** Note that the intrinsic learning rate was 0.1 for both WD and BN, however Infomax had an intrinsic learning rate of 0.0001. The weighted moving average rule, WD, outperforms other methods. Again, 10 experiments were done with the results averaged.

MNIST, with WD slightly outpacing batch norm. On CIFAR-10, the incremental batch norm rule was slightly outperformed by a standard network. It should be noted that Infomax seemed to work best when a small exponential decay factor was added to the intrinsic learning rate, η. Also, Infomax tended to diverge for a high η value, which suggests that WD may be more stable than Infomax, supporting the second hypothesis that Infomax may have a more stable implementation.

4.4 Discussion

The empirical results for the WD rule are far more compelling than the results for the IP update rule in Section 3.3. For both MNIST and CIFAR-10, the networks with WD far outperformed networks without. This was true for shallow networks, but the difference was even more apparent in deep networks. In deep networks, it was shown that the gradients of the activation function remain quite large throughout training. For this reason, the WD method seems to solve the vanishing gradient problem even more than its IP counterpart.

It is worth noting that while performing the experiments in Section 3, values for β outside $[-1, 1]$ were not observed. I am uncertain as to why β remained in this range, even when it was not strictly bounded to converge to values in this range. However, it does suggest that the WD rule above could be used to compute values of β that correctly center the distribution when implemented in synergy with weight updates.

Chapter 5

Additional Analysis

5.1 Using neuronal activity to compute the median of the input

The first theoretical observation is that my IP rule biases the activation functions of its neurons such that they are centered over the median of their input distributions, when tanh is used.

Lemma 1. *β converges to $\tilde{\mathbf{x}}$, the median of input distribution $\mathbf{x}$, for activation $y = \tanh(u)$, $u = \frac{x-\beta}{\alpha}$, and update rule $\beta = \beta + \eta \cdot \mathbb{E}[\mathbf{y}]$.*

Proof. Since tanh is an odd function ranging between -1 and 1, it can be approximated by the indicator function, $\mathbb{1}[-1, 1]$, that returns -1 if its input is less than 0 and 1 if its input is greater than or equal to 0. The error of this approximation is only large when u

is close to zero, which occurs when the function is already close to centered over its input. Hence, for a distribution, $\mathbf{u}$,

$$\begin{aligned}
\mathbb{E}[\mathbf{y}] &= \tanh(\mathbf{u}) && (5.1)\\
&\approx \mathbb{1}[-1,1](\mathbf{u}) && (5.2)\\
&= -1 \cdot p + 1 \cdot (1-p), \qquad \text{where } p \text{ is the proportion of inputs less than 0,} && (5.3)
\end{aligned}$$

which equals zero when $p = 0.5$. Thus, β continues updating until $\mathbf{u}$ is less than zero half the time and more than zero the other half, i.e. when $\beta = \tilde{\mathbf{x}}$. □

A corollary of this observation is that $\beta = \bar{x}$ (the mean of $\mathbf{x}$) for symmetric distributions. Some testing indicates that the distributions within networks tends to be approximately Gaussian, indicating that computing the mean, as done in batch normalisation, causes β to converge to similar values as the IP rule. Furthermore, this supports the idea that a benefit of batch normalisation is improving the information potential of neurons.

5.2 Information potential

Since the IP rule used in this work is derived using simplifying assumptions for the equilibrium solutions of the original Infomax rule, there should at least be empirical results to demonstrate that it is improving the information potential/entropy of the neurons in the network.

To this end, Figure 5.1 shows the results of an experiment performed by my collaborator, Tyler Jackson. This experiment demonstrates that for two different input distributions, uniform of width 4 centered on 1, and Gaussian with $\sigma = 2$ and $\mu = 1$, the IP rule

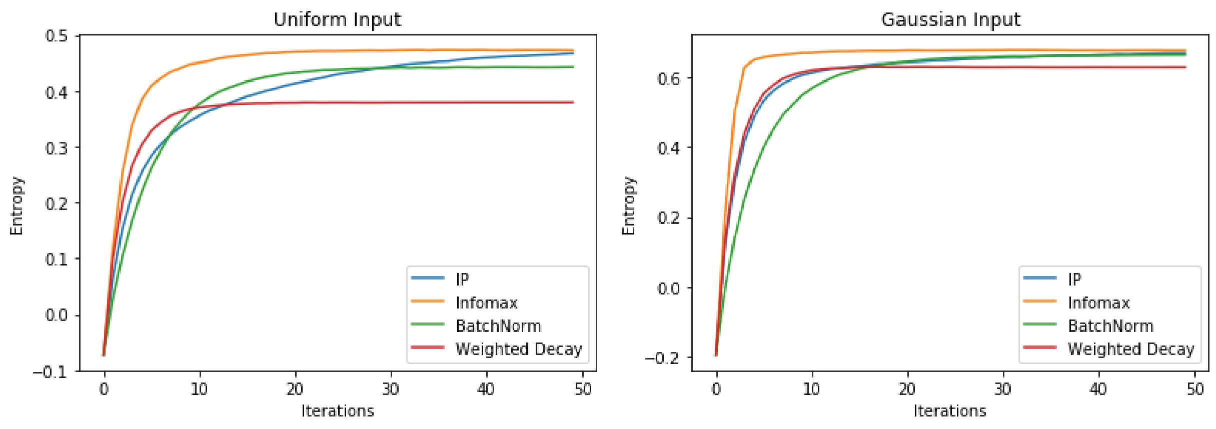


Figure 5.1: **Neuronal information potential.** These figures were generated by taking the entropy of the distribution, as estimated using the density histograms of the values of y as a Riemann approximation for the integral of the differential entropy. The update rules for each mechanism were applied for multiple iterations on the same collection of 10000 samples.

does in fact increase the entropy of the output distribution. For comparison, the effect of the incremental batch normalization rule is also included which also shows an increase in entropy as well as the original Infomax rule. Note that the IP rule converges to entropy levels similar to that of the Infomax rule, and outperforms the incremental implementation of BN. This supports part of my second hypothesis that the IP rule achieves the same purpose as Infomax, though the experiments in Section 3.3.3 failed to show that it is more stable.

This test demonstrates that the IP rule does, in fact, converge to the same levels of entropy as the Infomax rule (or at least very nearly). That being said, Infomax converges to these levels far faster. It is worth noting just how well the incremental batch norm rule improves the information potential of a neuron. This indicates that increased information entropy may be a cause of batch norm's success.

It is also important to note that the WD method has fairly poor levels of entropy, depsite it showing stronger results than IP in almost all other respects. This is unsurprising, as it was pointed out early that the WD method does not allow values for β outside of $[-1, 1]$. However, these results are taken from fixed input distributions, rather than from within networks during training. The interplay between synaptic learning, and its impact on the input distribution of downstream layers, may mean that the WD rule does converge to similarly high values of information potential in that setting.

To provide further validiation that the IP rule is working as intended, I conducted a similar test where the output of an activation function with the IP rule is measured for a fixed input distribution. In this case, the input was a Gaussian distribution with fixed mean of $\mu = 5$ and standard deviation of $\sigma = 2$. Figure 5.2 shows that a neuron with the IP rule converges to an approximately uniform output distribution. Since the uniform distribution has the highest information entropy, this supports the hypothesis that this IP rule maximises information potential. The same graph for the WD rule is not included, because it fails to shift the activation function correctly when the center of the distribution is outside $[-1, 1]$.

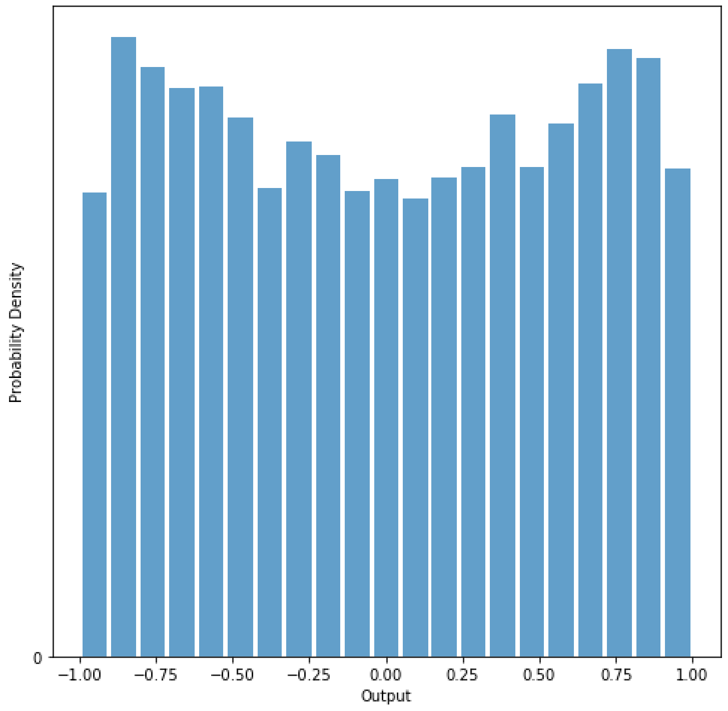


Figure 5.2: **Output distribution of a neuron with IP.** The above graph is a histogram that shows the ouput distribution for a neuron with the IP rule for a fixed Gaussian distribution after 10000 iterations.

5.3 Choosing the update rule for α

The inspiration for the choice of α comes from observing that the Infomax update rule for α is

$$\alpha = \alpha + \eta \left(\frac{1}{\alpha} - 2\,\mathbb{E}[\mathbf{xy}] \right) \tag{5.4}$$

i.e.

$$\Delta\alpha \propto \frac{1}{\alpha} - 2\,\mathbb{E}[\mathbf{xy}]. \tag{5.5}$$

Treating this as a dynamic system and rearranging for $\Delta\alpha = 0$ yields

$$\begin{aligned} 0 &= \frac{1}{\alpha} - 2\,\mathbb{E}[\mathbf{xy}] && (5.6)\\ \Rightarrow \frac{1}{\alpha} &= 2\,\mathbb{E}[\mathbf{xy}] && (5.7)\\ \Rightarrow \alpha &= \frac{1}{2\,\mathbb{E}[\mathbf{xy}]} && (5.8) \end{aligned}$$

Note that this ignores the non-linearities from $y = \tanh\left(\frac{(x-\beta)}{\alpha}\right)$ in the $\mathbb{E}[\mathbf{xy}]$ term. However, the incremental nature of the IP rule may lessen the impact of these non-linearities on the value that α converges to. Also, α being expressed as the inverse of $2\,\mathbb{E}[\mathbf{xy}]$ is why the IP rule was chosen to be applied as $u = \frac{(x-\beta)}{\alpha}$ rather than $u = \alpha \cdot x + \beta$, as is done with Infomax.

Chapter 6

Conclusion

6.1 Summary

In this work, I studied the relationship between a local, intrinsic learning mechanism and a synaptic, error-based learning mechansim in ANNs. I developed a local intrinsic rule, dubbed IP (after "intrinsic plasticity", that was inspired by the Infomax rule. The biological plausibility of this rule was discussed, and it was shown to be more biologically plausible than the functionally similar batch normalisation method.

This work demonstrates that local information maximisation can work in conjunction with synaptic learning rules other than the MEE algorithm, which aims to minimise the entropy. In shallow networks, the IP rule improves learning on MNIST. It was shown that this IP rule makes deep networks more robust to increases in synaptic learning rates, and that it increases the average value for the slope of the activation functions. When compared to batch normalisation and Infomax, whose family of solutions were shown to be the same, the IP rule demonstrates a negligible improvement in learning on MNIST and CIFAR-10.

In addition, an alternative rule was developed that had many of the same properties as IP, but used a weighted moving average to compute the desired values for the neuronal gain and bias rather than an Adamised update rule. This rule, dubbed WD, exhibits universally superior performance over IP. It improves performance in shallow networks on both MNIST and CIFAR-10. In deep networks, it shows faster learning and a greatly increased robustness to increases in synaptic learning. The gradients of the activation function were again compared, and the WD method shows far larger gradients on average, suggesting that this intrinsic, information-theoretic rule solves the vanishing gradient problem. The WD method also outperforms Infomax and a weighted moving average version of batch normalisation.

Supplementary analysis was done that reinforces the relationship between intrinsic plasticity and batch normalisation. Specifically, it was shown that the IP method centers its activation over the median of its input distribution, which is equivalent to centering it over the mean of the input distribution for symmetric distributions.

Analysis was also provided that demonstrates the IP rule converging to similar levels of neuronal entropy as the Infomax rule, when tested on a fixed input distribution. This same analysis shows that the WD version of intrinsic plasticity also improved information potential, but fails to reach the same levels as IP and Infomax. Interestingly, it was observed that batch normalisation also improves information potential, suggesting that this may be a cause for the efficacy of batch normalisation—an open problem as of the writing of this thesis.

6.2 Future Work

It was observed in some preliminary tests that the use of $\text{Cov}(\mathbf{x}, \mathbf{y})$ for updating α yields more stable behaviour than $2\,\mathbb{E}[\mathbf{xy}]$, at the expense of a small decrease in performance. I suspect that this may be due to approximation errors in $\mathbf{y}$ as the α and β parameters are learned. Since $\text{Cov}(\mathbf{x}, \mathbf{y}) = \mathbb{E}[\mathbf{xy}] - \mathbb{E}[\mathbf{x}]\,\mathbb{E}[\mathbf{y}]$, these approximation errors are partly offset by the $\mathbb{E}[\mathbf{x}]\,\mathbb{E}[\mathbf{y}]$ term when $\text{Cov}(\mathbf{x}, \mathbf{y})$ is used. Further reasearch is required to verify that this is the case.

In addition, more work is required to explain the discrepancy in performance between the more theoretically grounded IP rule presented in Section 3, and the better performing WD rule presented in Section 4. As mentioned in Section 4.4, sampled values for β within a learning network tended to be in the range of $[-1, 1]$. Since the test for entropy involved evaluating these rules for fixed distributions, it would be interesting to see if the WD method has improved levels of entropy when tested in an actual network, where input distributions are constantly shifting. If this was the case, then that would improve the theoretical justification of why the WD method works so well, meaning that the WD method would have the best of both worlds—strong theory as well as greatly improved performance.

Another concern with the work done is that occasionally α would become negative during updates. This is a problem because negative values of α result in the activation function being reflected in the y axis. With Infomax, this is not possible in the continuous case, as the $\frac{1}{\alpha}$ term in the update rule pushes α away from zero when it becomes too small. In the IP rule, $2\,\mathbb{E}[\mathbf{xy}]$ tends to be positive, since the activation function is initially centered over zero, and $\mathbb{E}[\mathbf{xy}]$ tends to be slightly positive when β reaches its equilibrium point. So both rules should never have negative values of α. However, in both cases, α would occasionally cross into negative values due to numerical approximation errors from the discrete updates. Strangely, this did not seem to adversely affect performance. Some preliminary tests suggest that this is because tanh returns similar values when a reflection occurs in cases where $\mathbb{E}[\mathbf{xy}]$ is negative.

Furthermore, more involved experiments are required to test the hypothesis that IP is more stable than Infomax, when used as a local rule. One possibility would be to increase the intrinsic learning rates to test which rule fails first. Another possible test would be to have a network continue learning for far longer than was done in these tests, to see if one rule eventually diverges. In contrast, the WD rule worked for much larger values of η than Infomax. This leads me to suspect that IP may be as well.

Finally, to further test the hypothesis that batch normalisation works because it improves information entropy, experimentation should be done that compares the performance of a "hard", or instantaneous, version of IP to conventional batch norm, as we have only compared the IP rule to an incremental rule that converges to the parameters used in batch normalisation. Doing so would detract from the biological plausibility of the IP rule, but may yield benefits in machine learning.

6.3 Closing Remarks

The concrete focus of this work has been the interplay between intrinsic and synaptic plasticity, and attempting to show that intrinsic plasticity can solve the vanishing gradient problem. More abstractly, the goal of my work is to close the gap between two views:

that phenomenon in neurobiology can inform the theory of machine learning, and that mathematical principles underlie the function of brains.

As a researcher in AI or ML, it is important not to forget that many advances in these fields have happened after scientists and mathematicians directed their attention back to the biological brain, and that biological plausibility is not just a vanity that needlessly restricts the development of models. At the same time, work in neuroscience and neurobiology could be bolstered by asking questions that lead to thinking of neurological functions in terms of mathematical ideals. It is, after all, quite beautiful that information-theoretic notions seem to be found in the brain just as much as a telecommunication channel or a data compression algorithm, especially given that these phenomena are emergent in biology rather than designed, as is the case in these latter examples.

Ultimately, while the majority of this work has focused on how intrinsic plasticity can improve learning by using performance as a metric, I hope that the reader can appreciate that the real goal is not to ask the question "How does this compete against other models?" but rather to approach the study of neural networks in a principled manner, always keeping theory within arm's reach and keeping an eye out for how the insights gained from studying neural networks may relate to other areas of research, and vice versa.

References

1. Hebb DO. “The organization of behavior: A neuropsychological theory.” Psychology Press; (2005) Apr 11.

2. Oja E. “Simplified neuron model as a principal component analyzer.” Journal of mathematical biology. (1982) Nov 1; 15(3):267-73.

3. Werbos PJ. “Applications of advances in nonlinear sensitivity analysis.” In System modeling and optimization (1982) (pp. 762-770). Springer, Berlin, Heidelberg.

4. Shannon CE, Weaver W. “The mathematical theory of communication.” University of Illinois press; (1998) Sep 1.

5. Triesch J. “Synergies between intrinsic and synaptic plasticity in individual model neurons.” Advances in neural information processing systems (2005) (pp. 1417-1424).

6. Triesch J. “A gradient rule for the plasticity of a neurons intrinsic excitability.” In International Conference on Artificial Neural Networks 2005 Sep 11 (pp. 65-70). Springer, Berlin, Heidelberg.

7. Bell AJ, Sejnowski TJ. “An information-maximization approach to blind separation and blind deconvolution.” Neural computation. 1995 Nov; 7(6):1129-59.

8. Li Y, Li C. Synergies between intrinsic and synaptic plasticity based on information theoretic learning. PloS one. 2013 May 9; 8(5):e62894.

9. LeCun Y, Bengio Y, Hinton G. “Deep learning.” Nature. 2015 May; 521(7553):436.

10. Ioffe S, Szegedy C. “Batch normalization: Accelerating deep network training by reducing internal covariate shift.” arXiv preprint arXiv:1502.03167. 2015 Feb 11.

11. Santurkar S, Tsipras D, Ilyas A, Madry A. “How does batch normalization help optimization?” Advances in neural information processing systems 2018 (pp. 2483-2493).

12. Kohler J, Daneshmand H, Lucchi A, Zhou M, Neymeyr K, Hofmann T. “Towards a theoretical understanding of batch normalization.” stat. 2018 May 29; 1050:27.

13. Kingma DP, Ba J. “Adam: A method for stochastic optimization.” arXiv preprint arXiv:1412.6980. 2014 Dec 22.

14. Bengio Y, Lee DH, Bornschein J, Mesnard T, Lin Z. “Towards biologically plausible deep learning.” arXiv preprint arXiv:1502.04156. 2015 Feb 14.

15. Deng L. The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Processing Magazine. 2012 Oct 18;29(6): 141-2.

16. Krizhevsky A, Nair V, Hinton G. The CIFAR-10 dataset. online: http://www. cs. toronto. edu/kriz/cifar. html. 2014;55.

17. Erdogmus D, Principe JC. “An error-entropy minimization algorithm for supervised training of nonlinear adaptive systems.” IEEE Transactions on Signal Processing. 2002 Aug 7; 50(7):1780-6.

18. Sumbre, German, et al. “Control of octopus arm extension by a peripheral motor program.” Science 293.5536 (2001): 1845-1848.

19. Hodgkin, Alan L., and Andrew F. Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve.” The Journal of physiology 117.4 (1952): 500-544.

20. Rosenblatt, Frank. “The perceptron: a probabilistic model for information storage and organization in the brain.” Psychological review 65.6 (1958): 386.

21. Riesenhuber, Maximilian, and Tomaso Poggio. “Hierarchical models of object recognition in cortex.” Nature neuroscience 2.11 (1999): 1019.

22. Minsky, Marvin, and Seymour Papert. “An introduction to computational geometry.” Cambridge tiass., HIT (1969).

23. Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. “Learning internal representations by error propagation”. No. ICS-8506. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

24. Silver, David, et al. “Mastering the game of Go with deep neural networks and tree search.” nature 529.7587 (2016): 484.

25. Falcone, Paolo, et al. “Predictive active steering control for autonomous vehicle systems.” IEEE Transactions on control systems technology 15.3 (2007): 566-580.

26. Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." Proceedings of the fourteenth international conference on artificial intelligence and statistics. 2011.

27. He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

28. Schrauwen, Benjamin, et al. "Improving reservoirs using intrinsic plasticity." Neurocomputing 71.7-9 (2008): 1159-1171.

29. Whittington, James CR, and Rafal Bogacz. "An approximation of the error backpropagation algorithm in a predictive coding network with local Hebbian synaptic plasticity." Neural computation 29.5 (2017): 1229-1262.

30. Steil, JJ. "Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning." Neural Networks 20 (2007): 353364.

APPENDICES

Appendix A

The Back Propagation Algorithm

This appendix is provided as a reference for readers who are unfamiliar with the error back propagation algorithm. While the details necessary for understanding this work are provided in the main body of the text, the back prop algorithm is so fundamental to neural networks that a full description may help readers who wish to develop more intuition for how signals propagate in ANNs—both forward, as activities, and backwards, as errors.

A.1 Overview

A complete algorithm for updating the weights of an ANN consists of

1. **The feedforward pass.** This is when sample inputs are fed into the input layer of the network, then propagated forward through each layer until an output is returned.
2. **Computing the error.** Once the output, y_{out}, is returned, it is compared to a

target value, t, and a loss is computed to determine how "far" the output was from the target. The L_2 norm (a.k.a. the euclidean distance), or the categorical cross entropy are typically used, for regression and classification problems respectively.

3. **The backward pass.** Now that an error signal has been computed, it is then propagated back through the network, while the gradient of the error w.r.t. both activities and weights are determined. This is done recursively, with the gradients of the error w.r.t. the activities for later layers being used to compute the same gradients in early layers.

4. **Updating the weights.** When the gradient of the error has been found with respect to every weight, the weights are then updating simultaneously through the process of gradient descent.

In practice, this is done many times. Learning to perform a task often requires observing upwards of thousands or even millions of different samples, and the network may need to observe each of these samples multiple times, since the improvement in performance is typically incremental with each sample.

Note that the conventions for notation used in the main body of this work will continue here. As a reminder, the superscript refers to the layer of the network, indexed from zero, while the subscript will be used to refer to particular neurons, indexed similarly. For weights, which sit between layers, their superscript will be indexed such that they match the output layer, while their subscripts will refer first to the input neuron, then the output neuron. For example, $w_{ij}^{(1)}$ will refer to the connection that joins the i-th neuron in the input layer to the j-th neuron in the first hidden layer.

A.2 The Feedforward Pass

A sample from a dataset is composed of an input, x, and a target output, t. The first step is to take x and pass it into the input layer of the network (so $x = x^{(0)}$). Then, $y^{(0)} = \sigma(x^{(0)})$ is computed for a given activation/transfer function, σ.

This value is transformed by the weight matrix, $w^{(1)}$, and biased by $b^{(1)}$ to yield the input to the next layer

$$x^{(1)} = y^{(0)} \cdot w^{(1)} + b^{(1)}.$$

This is then done recursively for each of the k layers of the network, where the general steps are

$$x^{(i)} = y^{(i-1)} \cdot w^{(i)} + b^{(i)} \tag{A.1}$$

$$y^{(i)} = \sigma(x^{(i)}) \tag{A.2}$$

until a final output for the network, $y_{\text{out}} = y^{(k)}$ is determined.

A.3 Computing the Error

With the output, y_{out}, computed, it is now necessary to determine how well the network performed and generate an error signal. The function used to compute this error signal is typically called the loss function or objective function. In a regression problem, where you are typically attempting to fit a hyperplane to some ground truth hyperplane, the L_2 norm is typically used—to measure how far away the output is from the target. This is given as

$$E = |y_{\text{out}} - t|^2$$

whose derivative is (conveniently)

$$\frac{\partial E}{\partial y_{\text{out}}} = y_{\text{out}} - t.$$

For classification problems, the categorical cross entropy can be used instead. This is

$$E = -\sum_i [t \cdot \log(y_{\text{out}}) + (1-t) \cdot \log(1 - y_{\text{out}})]$$

for each class, i. The gradient for this loss function is (again, very conveniently)

$$\frac{\partial E}{\partial y_{\text{out}}} = \sum_i y_{\text{out}} - t.$$

There are many other possible loss functions. What is important is that $\frac{\partial E}{\partial y_{\text{out}}}$ can be computed. This means that it must be possible to express the error in terms of the output of the network. For training with multiple samples, it is also important that the error can be averaged across samples.

A.4 The Backward Pass

Now that an error signal, E, has been generated, this error needs to be propagated back through the network with respect to the parameters of the network. Consider $\frac{\partial E}{\partial w^{(\ell)}}$ for an arbitrary layer ℓ. We know by the chain rule that this can be rewritten as

$$\frac{\partial E}{\partial w^{(\ell)}} = \frac{\partial E}{\partial y^{(\ell)}} \cdot \frac{\partial y^{(\ell)}}{\partial x^{(\ell)}} \cdot \frac{\partial x^{(\ell)}}{\partial w^{(\ell)}}. \tag{A.3}$$

Thus, provided that these three values on the right can be computed, it is possible to compute the contribution to error for any weight in the network. $\frac{\partial y^{(\ell)}}{\partial x^{(\ell)}}$ is simply the slope of the activation function, which can be easily found for any differentiable function. Furthermore, since $x^{(\ell)} = w^{(\ell)} \cdot y^{(\ell-1)}$, we have $\frac{\partial x^{(\ell)}}{\partial w^{(\ell)}} = y^{(\ell-1)}$.

This breaks down the problem of assigning the contribution of error into three main parts. Determining the gradient of the error w.r.t.

1. The activity of a given layer
2. A given weight
3. A given bias

First, consider the gradient of the error w.r.t. an activity, $\frac{\partial E}{\partial y^{(\ell)}}$, when ℓ is the output layer of the network. As shown in the previous section, this is often fairly straightforward, and is simply the gradient of the loss function (which is why it was important to assume that the loss was in terms of the output of the network).

Now consider when ℓ is not the output layer of the network. In this case, we can recursively compute the error for this layer in terms of the error of the layer above. We know that this can be done since we've already considered the base case of the output layer. So, provided we have $\frac{\partial E}{\partial y^{(\ell+1)}}$, we can express $\frac{\partial E}{\partial y^{(\ell)}}$ as

$$\frac{\partial E}{\partial y^{(\ell)}} = w^{(\ell+1)\mathrm{T}} \cdot \frac{\partial E}{\partial y^{(\ell+1)}},$$

where w^{T} is the transpose of w. The intuition for this operation is that we are essentially passing the error back through w by reversing the operation that it applies.

Having found $\frac{\partial E}{\partial y^{(\ell)}}$, The lefthand side of Equation A.3 can now be computed by simply multiplying the three values together elementwise on the righthand side. This gives a final expression for the error w.r.t. a given weight as

$$\frac{\partial E}{\partial w^{(\ell)}} = \left(w^{(\ell+1)\mathrm{T}} \cdot \frac{\partial E}{\partial y^{(\ell+1)}} \right) \cdot \sigma'(x^{(\ell)}) \cdot y^{(\ell-1)} \tag{A.4}$$

with each value on the righthand side having been previously determined.

Finally, we have to do the same thing for each bias in the layer. This is even simpler, as

$$\frac{\partial E}{\partial b^{(\ell)}} = \frac{\partial E}{\partial y^{(\ell)}}.$$

Since $\frac{\partial E}{\partial y^{(\ell)}}$ has already been computed, $\frac{\partial E}{\partial b^{(\ell)}}$ is obtained immediately.21

A.5 Updating the Weights

Finally, once all the gradients w.r.t. the weights and biases have been computed, they can now be simultaneously updated. For some learning rate η, we wish to *decrease* loss and so we have

$$w_{\mathrm{old}}^{(\ell)} = w_{\mathrm{old}}^{(\ell)} - \eta \cdot \frac{\partial E}{\partial w_{\mathrm{old}}^{(\ell)}}$$

and

$$b_{\mathrm{old}}^{(\ell)} = b_{\mathrm{old}}^{(\ell)} - \eta \cdot \frac{\partial E}{\partial b_{\mathrm{old}}^{(\ell)}}$$

for every layer ℓ.

A.6 Practical Limitations and Implementation

The above algorithm presents back propagation in its simplest form, and only for one interation. In practice, back prop is repeated for multiple iterations, as it incrementally converges to a minima for the error. While the algorithm should theoretically converge to a minima for any continuous loss function, there are a large number of practical concerns that limit its implementation. For one, the steps taken in gradient descent, dictated by η, are discrete, and are thus prone to instability if the steps are too large, and taking excessive time to converge if they are too small. Also, an immediate problem is that the optimisation manifold, or "landscape", may not be convex, and thus a discovered minima may not be optimal.

Another issue is that optimisation manifolds for the loss function may not be well-shaped, even if they are convex. For multi-dimensional loss functions, the surface may be stretched such that the surface is very wide when projected along one axis, and very narrow when projected along another. This can cause back propagation to converge very slowly as it attemps to reach the bottom of its nearby valley rather than trying to go more directly to the minima.

Many of these issues are currently being studied and are open topics of research. A review of all the proposed solutions to these issues would be too involved for this work, but I will briefly mention that augmenting back propagation with "momentum" mechanisms is a common and fruitful means of addressing both the problems stated above, as learning can gain enough momentum to "roll out of" local minima, and that it will improve the rate at which longer valleys in the optimisation surface are traversed. The algorithm featured in the main algorithm of this work, Adam, does exactly these things.

Finally, it is important to note that the cost function is actually conditional to the input of the network. For this reason, training heavily on a certain sample can negatively condition the network when it attempts to generalise to new samples. This limits a network's ability to generalise across a dataset of multiple samples and classes. For this reason, large datasets often are fragmented, and the order that samples are presented to the network is randomised. This randomisation is so key to learning that learning through gradient descent of error is almost universally referred to as stochastic gradient descent, or "SGD" for short.

Appendix B

A Primer on Information Theory

The field of information theory is relatively young, born primarily out of the work of Claude Shannon in 1948[1] [4]. Built upon probability and statistics, its central concern is the study of communication and noise in a signal. Though grounded very naturally in the practical world of communication (for instance, in telecommunication and networking), information theory is important in the abstract as well. At its heart, it formalises the connection between probability distributions and sampling, quantifies the effects of functions on random variables, and provides a principled grounds for relating statistics and probabilities to signals. Its generality in abstracting communication to probability and statistical theory make the applications of information theory varied and many. These include (but are not limited to) coding theory, cryptography, computer science, bioinformatics, linguistics, and—most importantly for this work—neurobiology and computational neuroscience. Furthermore,

[1]Though it is important to note that many of the ideas that Shannon built upon had been developed previously, though poorly formalised and not expressed with as much generality.

its importance in the abstract is reinforced by its close similarity to many principles found in thermodynamics, as the equation for entropy in statistical mechanics is identical to the one given for information.

While too large a field to adequately outline here, I will make an effort to describe two fundamental notions in information theory. The first is the measure of information contained in a single random variable, which can be more concretely interpreted as a "source" of information. This is measured in terms of "entropy," and shares its name with the same notion in thermodynamics due to the statistical connections between the two. The second is measuring what information is shared by two random variables, and can be thought of as the amount of information common to the sending and receiving ends of a signal. This value is measured in terms of "mutual information." Since this thesis is primarily concerned with measuring the entropy of a neuron's activity, less attention will be dedicated to this second notion.

B.1 Information Entropy

Consider a random variable, p, taken from a distribution, P. If the distribution of P is known, then each time you see a sample from P you can measure the new information, or "surprise", of the sample as $p \cdot \log(\frac{1}{p})$. This can be interpreted as the first term stating the likelihood of seeing p, multiplied by the amount of new information, $\log(\frac{1}{p})$ when p is seen. This second part of term makes intuitive sense, as samples p with lower likelihoods will lead to larger values of $\frac{1}{p}$, but thi should have diminishing returns, as captured by $\log(\frac{1}{p})$. If you wish to ask what the average information/surprise is of this distribution then this is just the sum of the information gained for each sample, multiplied by the likelihood of seeing that sample, formalised as

$$H(P) = \sum \left(p \cdot \log(\frac{1}{p}) \right) \tag{B.1}$$

$$= -\sum (p \cdot \log(p)) \tag{B.2}$$

for each p that is a possible outcome/sample of P.

The value H defined above is the entropy of a distribution P. The unit that H is measured in depends on the chosen base for log. If $\log_2$ is chosen, as is traditionally done, then the entropy of a distribution is measured in "bits" (which is precisely where the terminology comes from in computer science), or "shannons" in honour of Claude Shannon. If taken in base e[2], then the entropy is measured in terms of "nats". Decimal digits, or "hartleys", are for $\log_{10}$, and other units such as bytes can be used. Just like changing bases in counting systems, the effects of using different bases are largely unimportant, but the convention used in this work will always be that log expressed without a base will be treated as $\log_e$.

One property that stands out is that H should always be positive. Since P is a distribution, each p is bounded between 0 and 1. So the first part of the term p is positive, and $\log(p)$ is always less than or equal to zero. Since every term is negative, H being the negation of negative terms will always be positive. It should also be noted that in the case where p is zero, $p \cdot \log(p)$ will be evaluated as zero, since

$$\lim_{p \to 0^+} p \cdot \log(p) = 0$$

[2]This will never be written as ln because we are real mathematicians.

by L'Hopital's rule. Also, if $p = 1$ for a particular p, say p', then the entropy is zero, since

$$\begin{aligned} H(P) &= -\sum(p \cdot \log(p)) \\ &= (p' \cdot \log(p') && \text{since no other } p \text{ can have non-zero probability,} \\ &= 1 \cdot \log(1) \\ &= 0 && \text{since } \log(1) = 0. \end{aligned}$$

To develop more intuition of entropy in practice, let's consider a simple, illustrative example. Imagine a coin flip, with the two possible outcomes of head or tails. It is known that the distribution of this probabilistic event is the binomial distribution. If the coin is weighted such that it always returns heads, then heads will be seen every time, but there's no new information gained each time a heads is seen—it isn't surprising to see yet another instance of heads. So across all outcomes, there will be no new information. This is true of tails as well. Now imagine that the coin is heavily weighted to one outcome. When the unlikely outcome does happen it will be very rare, and so be very surprising. However, the majority of outcomes will be unsurprising, and carry very little information. For this reason, the average information across all outcomes will still be quite small, and so this event will have very little entropy.

In fact, it quickly becomes clear that H is maximised for this event when the weight of the coin is 0.5. This is because $(p \cdot \log(p)) + ((1-p) \cdot \log(1-p))$ is convex in p. Thus, a fair coin yields the highest entropy[3]. This notion actually generalises for distributions with more than two outcomes. This leads us to the central idea that much of this work is founded upon.

[3]Perhaps gamblers simply wish to maximise entropy?

Theorem 1. *For a random event, P, with k outcomes, the entropy of P is maximised when P is uniformly distributed. That is to say, when each of the k outcomes is equally likely.*

B.2 Mutual Information

Having discussed how information can be measured and maximised, we now focus briefly on defining how much information two sources can share. The mutual information between two sources of information is the amount of information gained about one distribution when observing the other. Consider two random variables, A and B, whose joint probability distribution is $P(A, B)$. The mutual information I is given as

$$I(A;B) = \sum_{a,b} P(a,b) \cdot \log\left(\frac{P(a,b)}{P(a)P(b)}\right),$$

or

$$I(A;B) = P(A,B) \cdot \log\left(\frac{P(A,B)}{P(A)P(B)}\right)$$

for short. This can be stated as either "the information gained about A while observing B" or vice versa, as I is a symmetric property.

This property bears a lot of similarity to the notion of entropy. One such property is that, like entropy, it is always non-negative. A key application of this concept is in the measure of channel capacity. In communication, the channel capacity is how much information can be carried from the input of the channel to the output, and it is exactly the maximum mutual information between the input and output of the channel.