An Analysis of a Biologically Plausible Mechanism for Error Backpropagation

William Greedy

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2019

Abstract

The backpropagation algorithm is the most commonly used method to train Artificial Neural Networks. In combination with other developments in Deep Learning, artificial networks that utilise this learning mechanism are becoming more powerful and are increasingly surpassing the performance of more conventional Machine Learning approaches. Due to its impressive performance and the biological inspiration for artificial neural networks, it is interesting to consider if the brain's learning mechanism is similar to backpropagation. This work focuses on a specific model using a new learning mechanism that is similar to backpropagation and works with more biologically realistic neurons. The model is inspired by neural circuitry observed in the brain and addresses a number of the issues that arise trying to create a biologically plausible implementation of backpropagation. The contributions of this project include a `Python` implementation of this model in addition to a flexible experiment framework that simplifies the model specification and its application to new tasks. This implementation was used to carry out further analysis in which the behaviour of the network dynamics under a set of different conditions was explored.

Acknowledgements

I would like to thank my supervisor, Dr. Matthias Hennig, for his valuable suggestions and guidance throughout this project.

Table of Contents

Chapter 1

Introduction

Recent developments in the field of Deep Learning have resulted in dramatic improvements in the performance of Artificial Neural Networks (ANNs) for an ever-increasing variety of tasks. Specific ANN architectures, developed for vision tasks and sequential decision making, have been able to achieve performance comparable, and sometimes exceeding, that of humans [1, 2]. Much of this success can be attributed to architectural developments such as Convolutional Neural Networks (CNNs) [3] and Long-Short Term Memory networks (LSTMs) [4]. However, the foundational building blocks of these networks — the perceptron and the backpropagation learning algorithm [5] — remain the same.

The human brain contains approximately 86 billion neurons [6] where each neuron can have thousands of synaptic connections, making it many orders of magnitude larger than the largest artificial networks [7]. Currently, the learning mechanisms of the brain and the ways in which they interact are not fully understood. Despite their biological inspiration, ANNs make use of a much simpler model of neurons and their connectivity in order to gain computational efficiency. These specific simplifications are a explained further in Section 2.1. For a number of reasons outlined in Section 2.3, the simplified model allows for the backpropagation algorithm to operate in a way that would not be possible for any mechanism in the brain. Nevertheless, backpropagation is still worth considering due to its performance and the possibility for similar, more biologically plausible mechanisms to mimic its effect. It has also been shown in recent work by Yamins et al. [8] that the internal responses within the brain when performing visual tasks are similar to those observed within artificial networks on the same task.

This work focuses on a single model developed by Sacramento et al. [9] that aims to provide a more plausible learning mechanism by taking inspiration from dendritic cortical microcircuits in the brain. Learning within this model uses a similar idea to standard backpropagation but satisfies significantly more of the biological constraints. This is achieved by adding complexity to the neuron model to give each one more realistic behaviour and the capability to represent multiple quantities simultaneously. Additional background to understand the model is given in Chapter 2 and a full description is given in Chapter 3.

The main contributions of this project are a well-structured implementation of the model dynamics, an experiment framework for this model in addition to an analysis of the model behaviour under different training conditions. Implementing this model was important as it is reasonably complex and has no publicly available implementation. This removes a significant barrier to entry for new research to be done and makes it easier for alterations or extensions to the model to be investigated. The implementation details can be found in Section 3.4 and a number of experiments to validate its correctness and reproduce the original results can be found in Section 4.1. In the final analysis, three specific research questions were identified relating the effect of noise on the network dynamics, the effect of synaptic weight plasticity on a set of feedback weights and an empirical comparison with updates from backpropagation. A number of experiments were designed to address these questions and their results are presented and interpreted in Section 4.2.

Chapter 2

Background

This chapter provides the necessary background information to understand the explanation for the dendritic error model in Chapter 3. Section 2.1 begins by describing the important differences between Artificial Neural Network models and the brain. Section 2.2 then describes the credit assignment problem in both contexts and Section 2.3 outlines the main reasons why the backpropagation algorithm cannot be applied to solve it within the brain. Finally, Sections 2.4 and 2.5 introduce two necessary concepts in use by the dendritic that address a single one of these problems.

2.1 Artificial Neural Network Models and the Brain

ANNs are heavily inspired by the brain and are made up of a collection of nodes, referred to as neurons, with weighted connections between them. Each neuron in the network can be thought of as representing an individual computational unit which takes a weighted sum of the activity from connected input neurons and applies a non-linear, monotonic *activation function* to produce an output activity. One of the most common and simplest class of ANNs is the Multi-Layer Perceptron (MLP) in which the neurons within the network are arranged in a sequence of distinct layers such that each neuron takes the activity from all neurons in the previous layer as input and propagates its computed output to all neurons in the following layer. The first layer is referred to as the *input layer*, the last layer as the *output layer* and all other layers as *hidden layers*. The entire network thus defines a computational graph for a potentially complex non-linear function of the inputs that is parameterised by the network weights. Interestingly, MLPs with just a single hidden layer are capable of approximating any non-linear function by adjusting the connection weights provided there are a sufficient

number of hidden layer neurons [10].

The simplistic approach to modelling neurons and their connections used in MLPs is not sufficient to capture the complexities of the behaviour of neurons found in the brain. A typical neuron within the brain will have a soma, an axon and a number of dendrites. The soma is the cell body and connects to the axon and dendrites. The role of the dendrites is to receive signals from the axons of many other neurons and relay this back towards the soma through electrical activity. Once the potential within the soma reaches some threshold, an event called an action potential or *spike* is triggered resulting in a rapid increase followed by a rapid decrease in the somatic potential. These action potentials are transmitted down the axon and the signal is communicated to the dendrites of connected neurons through a structure called a synapse. There are many notable between this and the MLP model. Within an MLP, signals are propagated as a continuous, real-valued output instead of discrete spike events. It is possible to interpret these real values as rates of spike events but in doing so any information conveyed by their precise timings is lost. Another key difference is that the neuron's potential is a physical quantity that evolves in continuous time and can differ based on location within the same cell. This contrasts with the homogeneous activity within MLP neurons that is fully determined by the previous layer activity and does not change until a new training example is presented.

2.2 Credit Assignment

The problem of *credit assignment* [11] in the context of both biological and artificial neural networks refers to how the contributions to the overall behaviour or output of the network can be determined for each neuron and all of their connections. The ability to solve this problem is vitally important in order for the network to be modified in such a way to improve its future performance. For artificial networks, the most effective and commonly used solution is called the backpropagation algorithm [5] in which the contribution of each weight is measured with respect to a loss function defined over the observed activity of the output neurons' and their desired output. This algorithm is motivated by the fact that both the linear weighted sum and non-linear activation function operations used by the network are necessarily differentiable allowing for the gradient of the loss function with respect to each weight to be computed. Furthermore, it exploits the observation that the gradients for any layer can be computed given only

the gradients of the next layer. Backpropagation can thus compute all gradients in an efficient way by starting with the final layer weights, for which the gradients can be computed directly, before propagating the result back through the non-linearity to the previous layer weights for use in the same computation. This process is repeated for all layers in the network until the input layer is reached. In order to reduce the overall loss and improve network performance, simple gradient descent strategies are typically used to take small steps in the weight space.

As well being much more computationally efficient than the naive approach to computing gradients, backpropagation provides an intuitive basis for viewing error feedback as a propagation of signals through layers which mirrors the process of the forward propagation through the network. It is therefore relevant when investigating the potential ways in which the brain solves the credit assignment problem as it has been shown that feedback channels exist within the brain that project signals from downstream brain areas back to earlier ones [12]. How the brain specifically solves this problem is still an open question and while the brain cannot implement backpropagation exactly for a number of reasons (Section 2.3), it is still possible that the biological mechanisms are similar or approximate the same behaviour.

2.3 Biological Plausibility of Backpropagation

There are a number of problems associated with the backpropagation algorithm that disqualify it from consideration as a complete explanation for how the brain learns. A summary of the six main issues presented by Bengio et al. [13] is as follows:

(1) **Linear Operations:** Every step in the backpropagation computation consists only of linear operations. This is a problem from the perspective of biology because real neurons necessarily involve the use of non-linear operations.

(2) **Knowledge of Feedforward Non-Linearity:** Feedback pathways would need to have knowledge of the non-linear operations in use in the feedforward pass to correctly propagate their gradients backwards. This is not realistic because the behaviour of different neurons can deviate which makes it hard to even estimate.

(3) **Weight Symmetry:** Also known as the *weight transport* problem [14], this refers to fact that the synaptic weights in the feedback pathways must exactly

match those in the forward direction for the correct values to be computed. Backpropagation can make this assumption for artificial networks since that information is globally available but in the brain the synaptic weight strengths are very much local to the specific connection.

(4) **Spiking Communication:** In the brain, neurons communicate through the use of discrete spike events that are the result of a rapid increase then decrease in the cell's membrane potential. Backpropagation makes the assumption that the relationship between the total input to a neuron and its output can be described as a differentiable function which holds for simple rate-based models with well-defined activation functions. However, in the case of spiking networks the assumption is broken since the gradients can either be considered to be undefined or sparse and non-stationary.

(5) **Separate Phases:** The backpropagation algorithm has two distinct phases of operation: the forward and backwards phase. Artificial networks typically perform both phases sequentially for a single training example before moving onto the next training example. Since the neurons in the brain operate using continuous and time-varying inputs, the concept of a single training example is not meaningful in the same way and so the cut-off point for the different phases is no longer clear. For the brain to use separate phases in learning it would also require a significant amount of coordination so that neurons are in the same phase at the same time and this has not been observed.

(6) **No Clear Targets:** There are no explicitly defined output targets in the brain therefore there is no clear definition of an error or loss function. This poses a problem since the objective of backpropagation is to assign credit to neurons and their connections with respect to some loss function. However, this does not prevent the use of implicit losses such as reconstruction error or internally generated losses.

These issues can be understood to arise from two main sources: the broken assumptions when moving from a simplistic model to a more biologically accurate one ((1), (4) and (6)) and the physical constraints that are present in the brain relating to the spatial and temporal locality of information ((2), (3) and (5)). These physical constraints are not usually enforced for artificial networks as they provide unnecessary inflexibility for the model design in a simulated environment where any and all relevant informa-

tion is stored indefinitely and can be accessed globally.

2.4 Feedback Alignment

The idea of *feedback alignment* was initially presented by Lillicrap et al. [14] and is an alternative but very similar mechanism to backpropagation that specifically aims to address the biological implausibility of weight symmetry (problem (3) from Section 2.3). They address this problem by proposing that the feedback weights used in the backwards phase of backpropagation can instead be fixed to randomly initialised values in a way that does not depend on the corresponding forward weights. The use of these asymmetric and decoupled weights yielded two key results. Firstly, this approach is able to achieve a similar performance to backpropagation trained networks on a number of different tasks and architectures. Secondly, the updates to the forwards weights during training implicitly led them to become more aligned with the fixed feedback weights, stabilising at an angle of 45°. This is important because as this alignment increases, the updates prescribed by the learning rule become closer to those prescribed by backpropagation which is known to produce useful updates. In independently developed work, Liao et al. [15] show that weight asymmetries due to differences in magnitude were of much less importance than differences in the direction or individual signs of the weights.

The effectiveness of feedback alignment is an initially surprising and highly unintuitive finding because the credit assignments no longer have a direct correspondence to the gradients. It can be better understood by observing that the initial angle between the two weight matrices drawn from independent and symmetric uniform distributions is 90° on expectation. At this angle, updates using the feedback weights have no expected improvement to the loss function. However, the updates do result in this angle decreasing over time and it can be shown that sufficiently small updates that are more closely aligned than 90° necessarily cause the weights to move closer to where they would be under an update in the direction of steepest descent. For non-trivial tasks there are typically a large number of local minima across the error landscape that are close in performance to the global minimum so it is not unreasonable to expect a single one of these minima to be found when moving in a direction similar to backpropagation.

2.5 Target Propagation

Target propagation [16] is another alternative to backpropagation for performing credit assignment that, like feedback alignment, also aims to address the biological implausibility of weight symmetry. The central idea behind the method is to compute and propagate *target* values backwards instead of the gradients. These target values are computed using a set of feedback weights that are decoupled from the feedforward weights and trained separately to reduce the reconstruction error between the target values and previous layer's activity. This means that every pair of consecutive layers essentially forms an auto-encoder made from the feedforward and feedback weights. This approach has a few benefits such as allowing for the network to learn using much stronger non-linearities and can also work well with stochastic and spiking neurons.

The intuition behind this mechanism can be easily gained from a simple example. Assume a completely linear network with just two layers which have n_1 and n_2 neurons respectively. The feedforward and feedback weights are denoted by $\overrightarrow{\mathrm{W}}(n_2 \,\mathrm{x}\, n_1)$ and $\overleftarrow{\mathrm{W}}(n_1 \,\mathrm{x}\, n_2)$ respectively and layer activity is given by x_1 and x_2. The activity in the second layer is a function of the first layer's activity using the feedforward weights, $x_2 = \overrightarrow{\mathrm{W}} x_1$. Using these values, an approximation of the first layer activity, $\hat{x}_1$, can be obtained using the feedback weights, $\hat{x}_1 = \overleftarrow{\mathrm{W}} x_2$. The objective for the feedback weights is to minimise the reconstruction error, $||x_1 - \hat{x}_1||_2^2 = ||x_1 - \overleftarrow{\mathrm{W}} x_2||_2^2$, which can be solved analytically in the linear case. The optimal solution for the feedback weights is the Moore-Penrose pseudoinverse [17] of the feedforward weight matrix, $\overleftarrow{\mathrm{W}} = \overrightarrow{\mathrm{W}}^{+}$. Assuming a good approximation is possible (i.e. the rank of the feedforward weights is not significantly higher than n_2) the pseudoinverse will have the property $\overrightarrow{\mathrm{W}}^{+} \cdot \overrightarrow{\mathrm{W}} \approx \mathbf{I}$ where $\mathbf{I}$ is the identity matrix. This means is that the corresponding columns and rows in the feedforward and feedback weight matrices will tend to have a positive dot product because of the positive 1's along the diagonal and will therefore be aligned with one another below an angle of 90°. As mentioned in Section 2.4, the alignment of the two weight matrices is far more important for training than their magnitudes being similar [15] so this same intuition can be applied here.

Chapter 3

Methods

3.1 Model Overview

This work focuses on a specific model of dendritic cortical microcircuits that was introduced by Sacramento et al. [9] and aims to address the problem of credit assignment with a more accurate neuron model and more biologically plausible synaptic plasticity rules than are typically used. This model is applicable to any kind of supervised learning problem and is capable of performing well on some simple classification and regression tasks. The main idea behind the model is related to the idea of predictive coding [18] in which the activity of later brain areas is used to predict sensory input and mismatches in this prediction are used as a residual error signal. In this case, the network has a layered structure where predictions of the activity of later layers are made and errors in these predictions are what drive the synaptic plasticity rules.

The model contains two neuron types, pyramidal cells and interneurons, that are connected in a specific way to produce the layered structure. This structure can be seen in Figure 3.1 for a simple two layered network. Each neuron is comprised of multiple compartments that are each modelled with their own membrane potential. This allows for single neurons to represent multiple different quantities simultaneously and is congruent with observed neuron anatomy. Each pyramidal cell is split into three compartments: the basal, somatic and apical compartments. Interneurons contain only a basal and somatic compartment. The basal compartment of each pyramidal cell in a given layer has weighted synaptic connections from the somatic compartment of every pyramidal cell in the previous layer. The activity that is propagated through these connections and represented in this compartment corresponds to the standard feedfor-

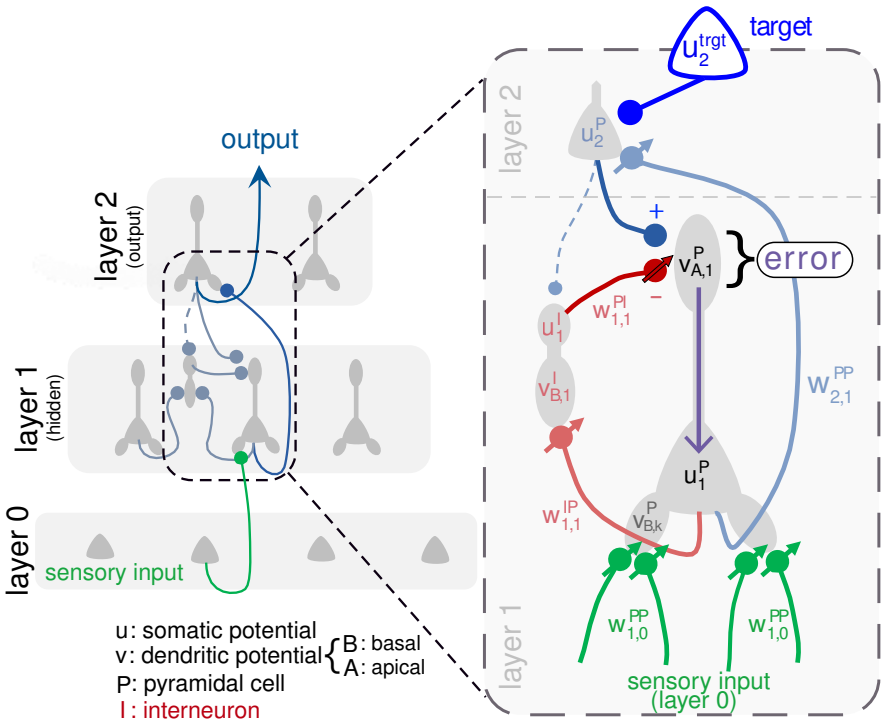

Figure 3.1: Network structure for a 2-layer network including both pyramidal cells and interneurons. Figure copied from [9].

ward activity that is seen in simpler networks and the synaptic weights associated with these connections are referred to as the *feedforward* weights. The apical compartment of each pyramidal cell is used to represent an explicit error signal derived from the activity of the next layer and a prediction of this activity encoded in the interneurons. It has synaptic connections from the somatic compartment of all pyramidal cells in the following layer in addition to the connections from the somatic compartment of all interneurons in the same layer. The weights on these connections are referred to as the *feedback* and *interneuron* weights respectively. The somatic compartment of the pyramidal cells is used to integrate the signal from both of these compartments. The number of interneurons in each layer is equal to the number of pyramidal cells in the next layer and there is a one-to-one correspondence between these cells. The goal of these interneurons is to predict the next layer activity which is done through the use of synaptic connections from the somatic compartment of all pyramidal cells in the same layer. The weights of these connections are referred to as the *prediction* weights and the goal of the network is for these to match exactly with the feedforward weights from the same layer. To make this possible, pyramidal cells in the next layer provide a weak teaching signal to their corresponding interneuron's soma.

Training this network on a given set of input-target pairs involves the sequential presentation of each pair for a fixed amount of time. The input values are presented to the first layer through the initial feedforward connections and the target values are used to create a teaching current that is applied to the somatic compartment of the output layer pyramidal neurons and nudges their potentials in the direction of those target values. Under ideal conditions and assuming an initially random network state, the evolution of the network weights throughout training should result in a network state where next layer activity is accurately predicted by each interneuron, apical compartments of the pyramidal cells are silenced (meaning there is zero prediction error) and output layer pyramidal cells produce activity that closely matches the target values. Any deviation from the target values will induce changes to all previous weights in the network.

This model is particularly interesting as it addresses a number of the problems outlined in Section 2.3 relating to the biological plausibility of the backpropagation algorithm. It can also be shown analytically to use updates that approximate those given by backpropagation under certain conditions. Specifically, issues (1) and (2) are addressed naturally by the neuron model as the same non-linearity is used for feedforward and feedback connections within the network. Issue (3) of weight symmetry is addressed through the use of separate feedback connections that can have fixed weights in a similar way to feedback alignment (Section 2.4) or can additionally undergo plasticity and be updated using a similar mechanism to target propagation (Section 2.5). The network is trained in continuous time with plasticity always on and without the need for separate phases for errors to be computed thus addressing issue (5). Neurons within the model communicate through the use of spike rates instead of individual spike events so it does not address issue (4) but there is nothing preventing the addition of spiking communication as an extension. Finally, since the network uses explicit target values, issue (6) is still present. This does not invalidate the usefulness of this model but more investigation would have to be done to justify where these target values could come from.

3.2 Model Dynamics and Plasticity Rules

This section describes and provides the justification for the specific equations that govern the network's behaviour. Equations 3.1 and 3.2 show the membrane equations for the somatic compartments of the pyramidal cells and interneurons respectively:

$$\frac{d}{dt}\mathbf{u}_k^{\mathrm{P}}(t) = -g_{\mathrm{lk}}\mathbf{u}_k^{\mathrm{P}}(t) + g_{\mathrm{B}}\left(\mathbf{v}_{\mathrm{B},k}^{\mathrm{P}}(t) - \mathbf{u}_k^{\mathrm{P}}(t)\right) + g_{\mathrm{A}}\left(\mathbf{v}_{\mathrm{A},k}^{\mathrm{P}}(t) - \mathbf{u}_k^{\mathrm{P}}(t)\right) + \sigma\boldsymbol{\xi}(t) \tag{3.1}$$

$$\frac{d}{dt}\mathbf{u}_k^{\mathrm{I}}(t) = -g_{\mathrm{lk}}\mathbf{u}_k^{\mathrm{I}}(t) + g_{\mathrm{D}}\left(\mathbf{v}_k^{\mathrm{I}}(t) - \mathbf{u}_k^{\mathrm{I}}(t)\right) + \mathbf{i}_k^{\mathrm{I}}(t) + \sigma\boldsymbol{\xi}(t) \tag{3.2}$$

Here $\mathbf{u}$ refers to the somatic potentials with the superscripts P and I indicating the pyramidal and interneuron cell types. The other dendritic compartments are denoted by $\mathbf{v}$ with the basal and apical compartments of the pyramidal cells differentiated using the subscripts B and A. For the interneurons there is only a single dendritic compartment so this subscript is omitted. The time is represented by t and the specific layer being referred to is indexed by k. Parameters g_{lk}, g_{B}, g_{A} and g_{D} are the cell membrane's leak conductance, the pyramidal cell's basal and apical conductances, and the interneuron's basal (dendritic) conductance. In the case of output layer pyramidal neurons, the apical compartment is not necessary and can be removed by simply setting the value of g_{A} to 0. Both somas take an additional Gaussian noise input, $\boldsymbol{\xi}(t)$, that represents that the independent background noise present in brain and its magnitude is controlled by parameter, σ. The final term, $\mathbf{i}_k^{\mathrm{I}}(t)$, is the somatic teaching current that nudges the potential in the direction of the next layer's pyramidal cell activity where $\mathbf{i}_k^{\mathrm{I}}(t) = g_{\mathrm{som}}(\mathbf{u}_{k+1}^{\mathrm{P}}(t) - \mathbf{u}_k^{\mathrm{I}}(t))$ and g_{som} is the somatic nudging conductance. Ignoring the noise, the terms in these equations all have the effect of moving the somatic potentials towards some target potential. The leak terms move the potentials to a resting potential (set to zero for convenience) and the others move them towards the potentials of connecting compartments. In a network state where the adjacent compartments have a constant potential both of these equations have steady state solutions where the effects of all terms cancel one another out.

The three dendritic compartment potentials evolve over time with no time lag and can be computed directly at each time step without knowledge of their previous values. Equations 3.3 and 3.4 show this for the basal compartments of each cell type.

$$\mathbf{v}_{\mathrm{B},k}^{\mathrm{P}}(t) = \mathbf{W}_{k,k-1}^{\mathrm{PP}}\phi\left(\mathbf{u}_{k-1}^{\mathrm{P}}(t)\right) \tag{3.3}$$

$$\mathbf{v}_k^{\mathrm{I}}(t) = \mathbf{W}_{k,k}^{\mathrm{IP}}\phi\left(\mathbf{u}_k^{\mathrm{P}}(t)\right) \tag{3.4}$$

Here $\mathbf{W}^{\text{PP}}_{k,k-1}$ denotes the feedforward weights from pyramidal cells in layer $k-1$ to pyramidal cells in layer k. Similarly, $\mathbf{W}^{\text{IP}}_{k,k}$ denotes the prediction weights from pyramidal cells in layer k to the interneurons in the same layer. The function ϕ is a positive and monotonically increasing rate function that is applied elementwise to each individual potential. Due to the matrix-vector product, the resulting basal potentials are just a weighted sum of rates from the soma all connecting pyramidal cells. The potentials for apical compartment of the pyramidal cells is shown in Equations 3.5 which is slightly different as they have two sets of connections.

$$\mathbf{v}^{\text{P}}_{\text{A},k}(t) = \mathbf{W}^{\text{PP}}_{k,k+1}\phi\left(\mathbf{u}^{\text{P}}_{k+1}(t)\right) + \mathbf{W}^{\text{PI}}_{k,k}\phi\left(\mathbf{u}^{\text{I}}_{k}(t)\right) \tag{3.5}$$

In this case $\mathbf{W}^{\text{PP}}_{k,k+1}$ and $\mathbf{W}^{\text{PI}}_{k,k}$ represent the feedback weights for the connections from the following layer and the interneuron weights for the same layer. Note here that because the rates are always non-negative, the weights must be allowed to take on negative values so that the apical compartment can eventually become completely silenced.

The synaptic plasticity rules dictate how the weights of the network change over time in response to the network activity. Under normal training conditions weight plasticity is switched on for the feedforward weights, the prediction weights and the interneuron weights. Weight plasticity for the feedback weights is optional and can be off to achieve feedback alignment (see Section 2.4) behaviour or switched on to get similar behaviour to target propagation (see Section 2.5). Equation 3.6 gives the plasticity rule for the feedforward weights.

$$\frac{d}{dt}\mathbf{W}^{\text{PP}}_{k,k-1} = \eta^{\text{PP}}_{k,k-1}\left(\phi\left(\mathbf{u}^{\text{P}}_{k}\right) - \phi\left(\hat{\mathbf{v}}^{\text{P}}_{\text{B},k}\right)\right)\left(\mathbf{r}^{\text{P}}_{k-1}\right)^{T} \tag{3.6}$$

Here $\eta^{\text{PP}}_{k,k-1}$ is just a scalar learning rate parameter that controls the speed of these weight changes for the connections from layer $k-1$ to k and $\mathbf{r}$ simply denotes the rates associated with the corresponding somatic potentials with $\mathbf{r} = \phi(\mathbf{u})$. Instead of using the basal potential directly, it is first rescaled such that $\hat{\mathbf{v}}^{\text{P}}_{\text{B},k} = \frac{g_{\text{B}}}{g_{\text{B}}+g_{\text{A}}+g_{\text{lk}}}\mathbf{v}^{\text{P}}_{\text{B},k}$. To understand why this rescaling is done it is instructive to consider the ideal equilibrium state of Equation 3.1 in the absence of noise when the apical potential is being completely silenced. By setting $\frac{d}{dt}\mathbf{u}^{\text{P}}_{k}(t) = \mathbf{0}$ the equation can be rearranged to obtain $\hat{\mathbf{v}}^{\text{P}}_{\text{B},k} = \mathbf{u}^{\text{P}}_{k}(t)$. Substituting this into the plasticity rule gives $\frac{d}{dt}\mathbf{W}^{\text{PP}}_{k,k-1} = \mathbf{0}$ because the difference in their rates must exactly cancel one another out. This property is important as it maintains the stability of this state. In the situation where there is an error represented by a

positive or negative potential in the apical compartment there will be a difference in the somatic potential from the scaled basal value in the same direction. This means that the difference operation of the plasticity rule will cause changes to the feedforward weights going into the basal compartment that correct for this and move them closer together.

The plasticity rules for the prediction weights and interneuron rates can be seen in Equations 3.7 and 3.8.

$$\frac{d}{dt}\mathbf{W}^{\mathrm{IP}}_{k,k} = \eta^{\mathrm{IP}}_{k,k}\left(\phi\left(\mathbf{u}^{\mathrm{I}}_{k}\right) - \phi\left(\hat{\mathbf{v}}^{\mathrm{I}}_{k}\right)\right)\left(\mathbf{r}^{\mathrm{P}}_{k}\right)^{T} \tag{3.7}$$

$$\frac{d}{dt}\mathbf{W}^{\mathrm{PI}}_{k,k} = \eta^{\mathrm{PI}}_{k,k}\left(\mathbf{v}_{\mathrm{rest}} - \mathbf{v}^{\mathrm{P}}_{\mathrm{A},k}\right)\left(\mathbf{r}^{\mathrm{I}}_{k}\right)^{T} \tag{3.8}$$

Note here that each plasticity rule has a learning rate parameter that can be set differently. The prediction weights are updated in much the same way as before where $\hat{\mathbf{v}}^{\mathrm{I}}_{k} = \frac{g_{\mathrm{D}}}{g_{\mathrm{D}}+g_{\mathrm{lk}}}\mathbf{v}^{\mathrm{I}}_{k}$. The motivation is also very similar — the error signal within the interneuron is represented by the teaching current from the next layer instead of an apical compartment. In the equilibrium state where the prediction weights are exactly equal the feedforward weights, $\mathbf{W}^{\mathrm{IP}}_{k,k} = \mathbf{W}^{\mathrm{PP}}_{k+1,k}$, the interneuron activity will exactly match the next layer activity and the teaching input will be zero. Any mismatches in the activity will again induce changes to the weights that attempt to correct this. The interneuron weights are updated by comparing the apical potential with some resting potential, $\mathbf{v}_{\mathrm{rest}}$, that is typically just set to 0. Weight changes are only induced when the apical compartment is not completely silenced. The use of potentials instead of their rates allows for the use of rate functions that are saturated at the resting potential.

Feedback weight plasticity is an optional addition to the model that gives rise to different network behaviour. In both the on and off states the network is typically still capable of learning a target function. The plasticity rule that govern these changes is given in Equation 3.9.

$$\frac{d}{dt}\mathbf{W}^{\mathrm{PP}}_{k,k+1} = \eta^{\mathrm{PP}}_{k,k+1}\left(\phi\left(\mathbf{u}^{\mathrm{P}}_{k}\right) - \phi\left(\hat{\mathbf{v}}^{\mathrm{P}}_{\mathrm{TD},k}\right)\right)\left(\mathbf{r}^{\mathrm{P}}_{k+1}\right)^{T} \tag{3.9}$$

Here $\hat{\mathbf{v}}^{\mathrm{P}}_{\mathrm{TD},k}$ represents the first term of Equation 3.5, $\hat{\mathbf{v}}^{\mathrm{P}}_{\mathrm{TD},k} = \mathbf{W}^{\mathrm{PP}}_{k,k+1}\phi(\mathbf{u}^{\mathrm{P}}_{k+1}(t))$, which is the weighted top-down feedback activity from the next layer's pyramidal cells. The plasticity rule compares this top-down activity with the somatic activity and changes the weights to minimise the difference. This relates strongly to the idea of target propagation (Section 2.5) where the feedforward and feedback weights can be viewed as

an auto-encoder minimising a reconstruction loss. Due to the application of two non-linearities and the potential loss of information when layer $k+1$ has fewer neurons that layer k, this reconstruction is unlikely to ever be perfect or provide full weight symmetry. The process of minimising this reconstruction error has the benefit of allowing for a greater alignment of the feedforward and feedback weights than would be possible when using feedback weights that are fixed.

3.3 Approximation to the Backpropagation Algorithm

Sacramento et al. [9] were able to show analytically that under certain conditions the updates prescribed by the synaptic plasticity rules given in Section 3.2 approximate those computed using the backpropagation algorithm. This section outlines some of the important conditions that are required, how they compare to the standard training conditions and the intuition for why this result holds true. The first condition they impose for simplicity is that the network operates in the absence of any noise. The next condition is that the prediction weights, interneurons weights and feedback weights are all tied to the feedforward weights such that the interneurons perfectly predict next layer activity, all apical potentials are zero and all feedback weights are symmetric. This is referred to as the *self-predicting* state and under normal and stable training conditions the network will try to move towards this state but will likely never achieve it exactly. The final condition for the updates to be a close approximation is that the strength of the nudging conductance, g_{som}, must be very small since the proof only holds in the limit $g_{\text{som}} \to 0$. In practice this parameter cannot be set too low without nullifying the effect of the teaching input and hindering ability of the network to learn at a reasonable speed. This introduces a trade-off between accurately matching the direction of backpropagation and training speed.

3.4 Implementation

A major goal for this project was to create a correct implementation for the dendritic error network described in Sections 3.1 and 3.2 that can be made publicly accessible with clear and well-structured code. This was carried out using the `Python` programming language because it provides many flexible and simple to use libraries for efficient numerical operations, plotting and standard neural network training. The implementation can be accessed through its GitHub repository [19] and it is split into two distinct com-

ponents. The first is the model and a simulator for its dynamics which is outlined in Section 3.4.1 and the second is an experiment framework that is used to instantiate a simulator, preprocess the dataset and monitor relevant quantities throughout training.

3.4.1 Dynamics Simulation

The dynamics simulator operates on a data structure containing the potentials and weights that define the state of the model according to the equations defined in Section 3.2. This state is stored for individual layers using arrays from the `NumPy` library and each update rule has been implemented using library operations that are optimised for execution on the CPU. Potential leveraging of the GPU was not explored here for the sake of code clarity and the prioritisation of correctness over performance. Many of the equations that define the updates are given as a ordinary differential equation which cannot be simulated perfectly using discrete timesteps. To approximate the continuous dynamics, the Euler method is used with a very small time steps to compute the updates for all potentials and weights in the network. Every update is computed separately using only the state at previous timestep which means that it necessarily takes $2n$ timesteps for an input to a network with n layers to have any effect on the output due to the different compartments. An alternative but valid approach could make use of the layered structure to define an ordering for the compartment updates and make this effect instantaneous but this was avoided in this case due to complexity and the potential for instability. For small enough timesteps this distinction is less important as both methods should approach the same behaviour.

3.4.2 Experiment Builder and Monitoring

There are no widely used libraries that are flexible enough to properly manage the running of the simulation and monitoring of the model's internal state due to the novelty of the model structure and the continuous nature of the training procedure. For this reason, a framework to build, run and monitor experiments was created from the ground up. This framework handles the creation of the model and simulator using parameters that are specified in a given `JSON` configuration file. It also handles the preprocessing of a given dataset with a discrete number of training examples into a repeating stream of continuous data. The simulator is continuously stepped through using this data and the network state is regularly saved to disk to allow for the experiment to be resumed in the event of a crash.

At regular intervals throughout training, a set of monitoring functions are evaluated to measure quantities of interest and these results are stored for the entire training period. This monitoring system is necessary because storing the entire network state for each iteration requires too much memory in most cases. It has been created to be extremely flexible and provides the implementations for a number of useful monitoring functions. In particular there are implementations that measure individual quantities within the network state such as the potentials or weights, layer statistics of these values as well the alignment angles for the different weight matrices in different layers. There are also implementations for more complex monitoring functions that involve copying the feedforward weights from the model state to a feedforward network, computing the full backpropagation update and comparing it with the updates from the model's plasticity rules. To support future extensions, there is also the ability for monitoring functions to be based on arbitrary lambda functions applied to the entire model state.

Chapter 4

Results

A number of experiments were carried out using the implementation of the dendritic error network that has been created and described in Section 3.4. In this chapter, Section 4.1 focuses on initial experiments that were designed to recreate key results of Sacramento et al. [9] and verify that the model has been implemented correctly. Section 4.2 then shows the results of a number of additional experiments that were each carried out to answer specific questions about the behaviour of this model under different conditions and how it compares to a network trained with backpropagation.

4.1 Model Validation

The purpose of this section is to show that the model implementation exhibits the key behavioural properties that would be expected under normal training conditions. Each result demonstrated here has a direct correspondence to what was shown when the model was initially presented. Section 4.1.1 shows the most basic properties of the model for a small network trained on a simple toy problem with just one training example. Some of the more complex properties are then demonstrated in Section 4.1.2 for a larger network trained on non-linear regression task.

4.1.1 Storing a Single Input-Output Pattern

In this experiment, a small network with just two inputs, one hidden layer containing two neurons and a single output neuron was trained using feedback alignment to store a single input-output pair. The purpose of this was to verify that the model dynamics and synaptic plasticity rules led to the network moving to a state in which the output

neuron potential correctly matches its intended target value. Using a task this simple allowed this property to be measured in a way that was isolated from situations where backpropagation would be necessary for effective learning. The neuron conductance parameters used here were: $g_{\mathrm{lk}} = 0.1$, $g_{\mathrm{A}} = g_{\mathrm{som}} = 0.8$, $g_{\mathrm{B}} = g_{\mathrm{D}} = 1.0$ with the standard deviation of the injected noise, σ, set to 0.2. The logistic function was used for the rate function, $\phi(u) = \frac{1}{1+e^{-u}}$, and the network was initialised in a random state with all weights drawn from a uniform distribution, $U(-1.0, 1.0)$. The four learning rate parameters $\eta^{\mathrm{PP}}_{1,0}$, $\eta^{\mathrm{PP}}_{2,1}$, $\eta^{\mathrm{PI}}_{1,1}$ and $\eta^{\mathrm{IP}}_{1,1}$ were all set to 0.005 for simplicity. The network is very robust to changes in these learning rates for this task so there was no need for their values to be tuned differently.

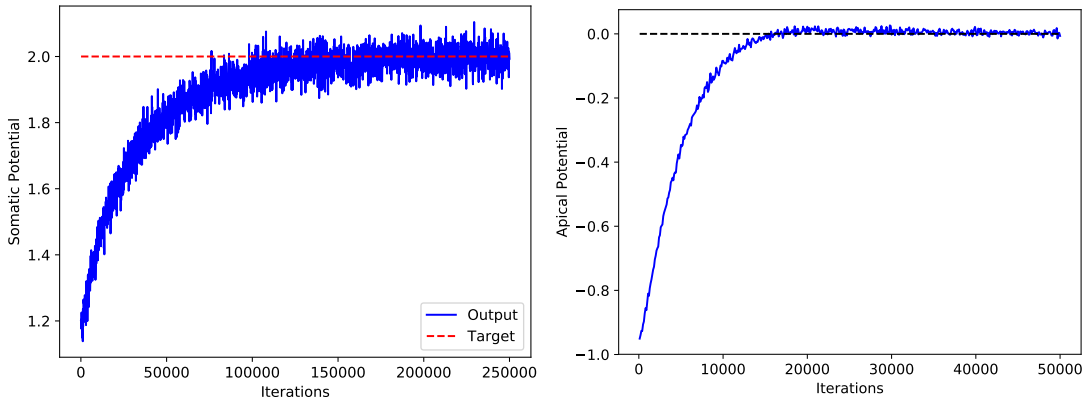

Figure 4.1: Compartment potentials throughout training. **Left:** The somatic potential of the output neuron. The red dashed line shows the target value for the output potential. **Right:** The apical potential for one of the two hidden layer neurons. The black dashed line shows where the potential is zero.

Figure 4.1 shows the evolution of the potentials within the somatic compartment of the output neuron and the apical compartment for one of the hidden layer neurons throughout training. Here it can clearly be seen that even in the presence of some noise the synaptic plasticity rules adjust the network weights such that the output somatic potential eventually matches the target value and it remains stable in this state. It can also be seen that interneuron weights are correctly altered so that the apical compartment potential is silenced. The tendency to move towards these states is an important property for the network to exhibit but is much harder to quantify for harder tasks where the inputs and output target are regularly changing.

4.1.2 Learning a Non-Linear Function with Feedback Alignment

For this experiment, a network was used that had 30 inputs, 50 hidden layer neurons and 10 output neurons. The network was trained to perform a non-linear regression problem that involves matching its outputs to those produced by a fixed target network applied to the same input. This target network is a simple deterministic feedforward network that uses the same function for its activation function as the rate function of dendritic error network, does not include the complexities of the multiple compartments or the additional interneurons and in this case used the same structure except for using only 20 hidden layer neurons. The main benefits of using a target network like this is that it provides the guarantee that there exists some set of synaptic weights that can produce the correct outputs perfectly and it allows for the difficulty of the task to be changed arbitrarily. Performing this more complex task allowed for the internal alignments of the different weight types to be looked at specifically which is important for verifying that the training process is resulting in the self-predicting network state that was described in Section 3.3.

The rate function that was used in both networks was a shifted and rescaled soft rectified linear function, $\phi(u) = 0.1 \cdot \log(1 + \exp(u - 3.0))$. The first and second layer weights of the target network were drawn from the uniform distribution $U(-2.0, 2.0)$ and $U(-10.0, 10.0)$ respectively which was done to match the original experimental setup and produce sparse outputs. The network used the same initialisation and neuron parameters as before in Section 4.1.1. Since the learning rates here are more important, they were set to proportionally match the learning rates that produced the initial results as follows: $\eta^{\text{PP}}_{1,0} = 0.011875$, $\eta^{\text{PP}}_{2,1} = 0.005$, $\eta^{\text{PI}}_{1,1} = 0.011875$ and $\eta^{\text{IP}}_{1,1} = 0.059375$. The input training data used here included a total of 500 examples drawn from a uniform distribution, $U(-1.0, 1.0)$, where each training example was presented for a total of 100ms (1ms = 10 iterations) before moving on to the next.

Figure 4.2 shows a plot of the sum of squares error over the training period, computed as $||\phi(\mathbf{u}^{\text{P}}(t)) - \phi(\mathbf{u}^{\text{trgt}}(t))||_2^2$, that has been exponentially smoothed with a time constant of 35000 iterations. The reduction of this error over the course of training indicates that the network is capable of remembering previously seen training examples and that the weight changes made during the presentation of each example are useful and are not lost when later examples are shown. In Figure 4.3 the alignments be-

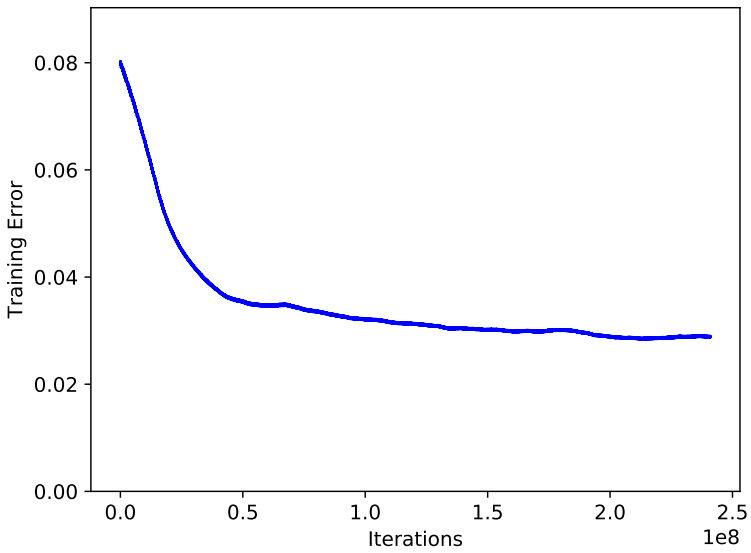

Figure 4.2: Sum of squares error throughout training for the network outputs compared with the outputs of the target network.

tween the internal weights within the network can be seen as training progresses. Here alignment refers to the angle between two flattened weight matrices. For two arbitrary matrices, A and B, this is computed as $\alpha = \cos^{-1}\left(\frac{\mathbf{a}\cdot\mathbf{b}}{||\mathbf{a}||\,||\mathbf{b}||}\right)$ where $\mathbf{a}$ and $\mathbf{b}$ are the flattened vectors containing the elements of A and B respectively. In a self-predicting state the expected behaviour is for the prediction weights to be closely aligned with the feedforward weights so that they produce accurate predictions of the next layer activity and for the interneuron weights to line up with the negative of the feedback weights so that the apical potentials are silenced. This is exactly the behaviour observed here as the angle between the prediction weights and feedforward weights reduces to approximately 4.3° from an initial value of 90°. The interneuron weights with the negative of the feedback weights can also been seen to align very quickly to an angle of around 20° indicated by the alignment with the positive weights at 160°. Since the feedback weights are fixed, the changes in this alignment are caused solely by the interneuron weight plasticity rule and the fluctuations are likely due to the comparatively higher learning rate and the fact that the apical compartment has connections from two separate noisy somatic compartments. It is not expected for either of these alignments to become perfect when the error is non-zero due to the presence of the teaching signal that directly causes mismatches in both predictions.

The final important property for the network is its ability to perform feedback align-

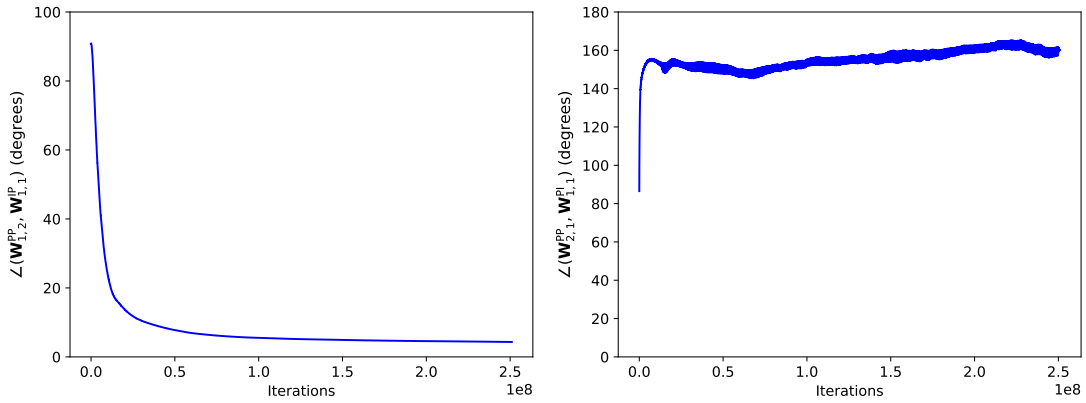

Figure 4.3: Alignment of internal weights throughout training. **Left:** Angle between the feedforward weights and the prediction weights. **Right:** Angle between the interneuron weights and the feedback weights.

ment. Unlike the previous two properties, this is an indirect product of all plasticity rules instead of the direct goal for a single one of the rules. Figure 4.4 shows the angle between the feedforward and feedback weight matrices over the training period. This clearly shows an increasing alignment until an angle of approximately 59.7° where it plateaus. This is in line with the original results that were able to achieve maximum alignment at an angle of approximately 54°. The presence of this property is extremely important as it is what allows the feedback connections in the network to provide signals to the previous layers that result in useful updates.

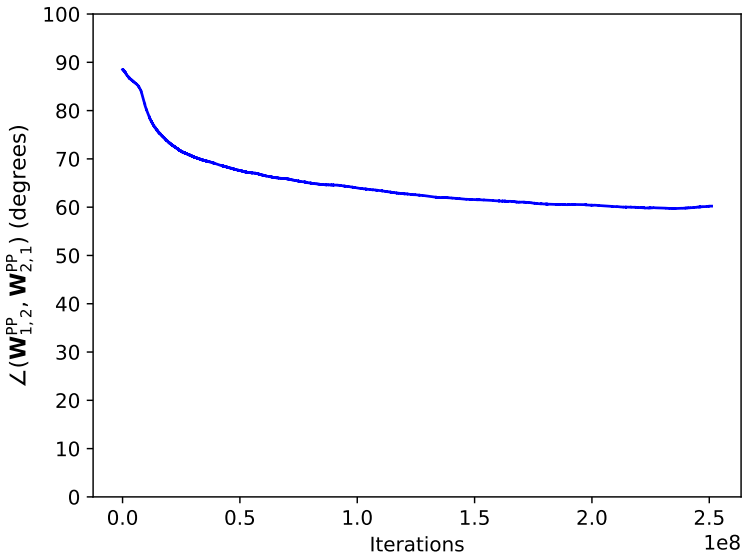

Figure 4.4: Alignment angle of the feedforward and feedback weights during training.

4.2 Further Analysis

As was mentioned in Section 3.3, the training procedure of this model approximates backpropagation under very specific conditions. The analysis in this section focuses on exploring the network behaviour outside of these conditions and looking at how modifying them affects the network performance. The secondary goal was to relate these insights back to the behaviour of standard backpropagation algorithm and highlight notable differences and similarities between the two approaches.

The experiments that were carried out explore three main directions. Firstly, Section 4.2.1 looks at the effect of the injected noise on the network dynamics. This specifically focused on the network robustness under differing levels of noise and its effect on the network performance in terms of training speed, training performance and generalisation performance. The goal here was to determine if the existence of noise provides measurable benefits to the network. Secondly, in Section 4.2.2 the effect of using feedback plasticity was compared with the use of fixed feedback weights. Here the goal was to determine whether or not training speed would increase as a result of the additional weight changes to the feedback weights or if these changes would cause instability and hinder the learning process. A secondary goal was to look at how the alignments of the feedforward and feedback weights would differ througout training using the two regimes. The final experiment in Section 4.2.3 was created to try obtain empirical results that quantify the difference in the updates prescribed by the model and by backpropagation. This allows for stronger claims to be made about their similarities when the conditions of the approximation are not met.

4.2.1 Training in the Presence of Noise

The original motivation for the presence of noise within this model was to demonstrate the robustness of the model dynamics under conditions that mimic the background noise in the brain. This experiment focused on investigating any secondary effects that the noise has on the network behaviour and how this changes when the level of noise is altered. The experimental setup used here was slightly modified from the setup from the non-linear regression task described in Section 4.1.2. A total of seven networks were trained on the exact same task to produce outputs that matched those of a fully feedforward target network with fixed weights. Each of the seven networks used the same layer structure and model parameters that were used produce the results in Sec-

tion 4.1.2. The single difference between the training conditions of the networks was the value of σ which represents the standard deviation of the noisy current injected into the somatic compartments of every neuron. The seven values of σ that were used were: 0.00, 0.15, 0.30, 0.45, 0.60, 0.9 and 1.20.

Starting from the same random weight initialisation, the evolution of the training error and the feedback weight alignment angle for all of the networks is shown in Figure 4.5. Both plots highlight an interesting effect of the noise on the training; the noise causes the speed of training and speed of alignment of weights to increase. In the case of the training error there is a limit to this effect as the largest noise level of $\sigma = 1.20$ results in an increase in error over the previous level at $\sigma = 0.9$. The cause of this effect can be understood by recalling that the rate function in use by the network promotes sparsity in the network activity. This means that for many of inputs there are numerous neurons that are saturated at near 0, which by inspection of the plasticity rules results in very small changes to their weights. In the presence of noise, the somatic potential will be much more variable and these neurons have the ability to occasionally produce larger activities where larger changes to the weights can be made. This essentially gives these neurons a mechanism by which to escape from the state of being overly saturated.

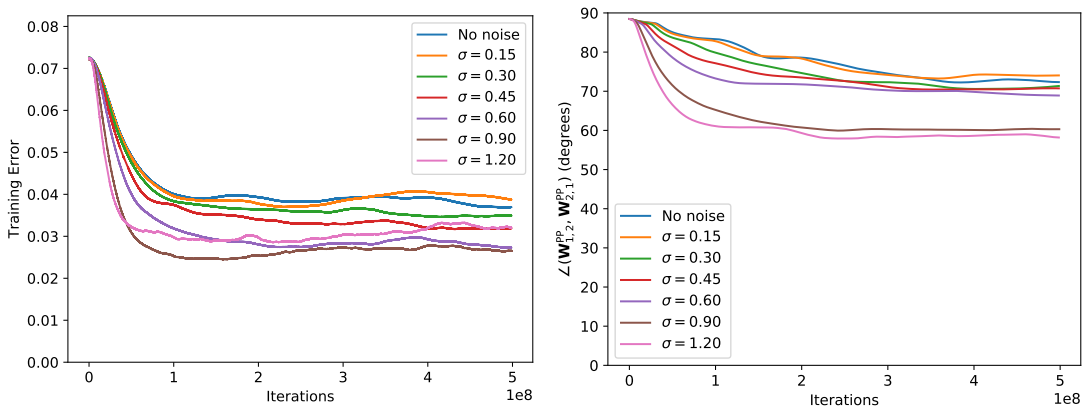

Figure 4.5: **Left:** Training error for each of the seven networks over the training period. **Right:** Alignment angle of Feedforward and Feedback weight matrices over the training period for each of the seven networks.

When considering the effect of noise in any type of neural network it makes sense to explore its regularisation effect on the weights. This is because the idea of adding noise as a means of regularisation is the central idea behind the commonly used method

for standard neural networks called Dropout [20]. This method introduces noise to the network by sampling from a Bernoulli distribution at each iteration and using these samples to mask the neuron outputs. It produces a regularisation effect by reducing the network's reliance on a small number of neurons since their outputs will not always be present and this forces the network to learn redundancies. A number of other methods exist for standard neural networks that use noise injection for regularisation, some of which are more similar to this dendritic error model and can involve the addition of Gaussian noise [21]. This connection between noise and regularisation was the motivation for first measuring the test error for each the different noise levels as shown in Table 4.1. Test error was measured at regular intervals throughout training by computing the average error over a total of 500 unseen examples in the absence of noise and with weight plasticity disabled. For each noise level, the results contain the value of training error at its lowest point and the corresponding test error. Interestingly, these results show that increasing the noise does actually provide an initial benefit to generalisation performance but as the noise becomes much larger it starts to have a negative effect.

Noise (σ)	0.00	0.15	0.30	0.45	0.60	0.90	1.20
Training Error	0.0382	0.0399	0.0371	0.0340	0.0284	0.0251	0.0277
Test Error	0.0810	0.0811	0.0807	0.0798	0.0794	0.117	0.118

Table 4.1: Lowest training error achieved and the corresponding test error for the seven networks trained under different noise conditions.

In addition to generalisation performance, another effect of regularisation is the suppression or penalisation of larger weight values. Figure 4.6 shows the mean of the feedforward weight magnitudes in the two layers for all seven networks during training. The self prediction state is shown by the convergence of the alignment angle between the feedforward and prediction weights and is achieved for all networks at iteration number $\approx 10^8$. Before a self-predicting state is reached, the effect of increasing the noise within the first layer appears to cause some instability that results in larger weights changes to be made. Once the network reaches a self-predicting state, the change in magnitude of these first layer weights begins to slow down significantly. The initial instability appears to overpower any penalty for the weights taking on larger values. In the second layer the weight magnitudes are much larger in general. This makes the regularisation effect of the noise play a larger role because any noise

will be magnified by these weights into the output neurons. For the largest noise values of $\sigma = 0.9$ and 1.2 the plot shows the average weight magnitude beginning to plateau at a much lower value than those using less noise.

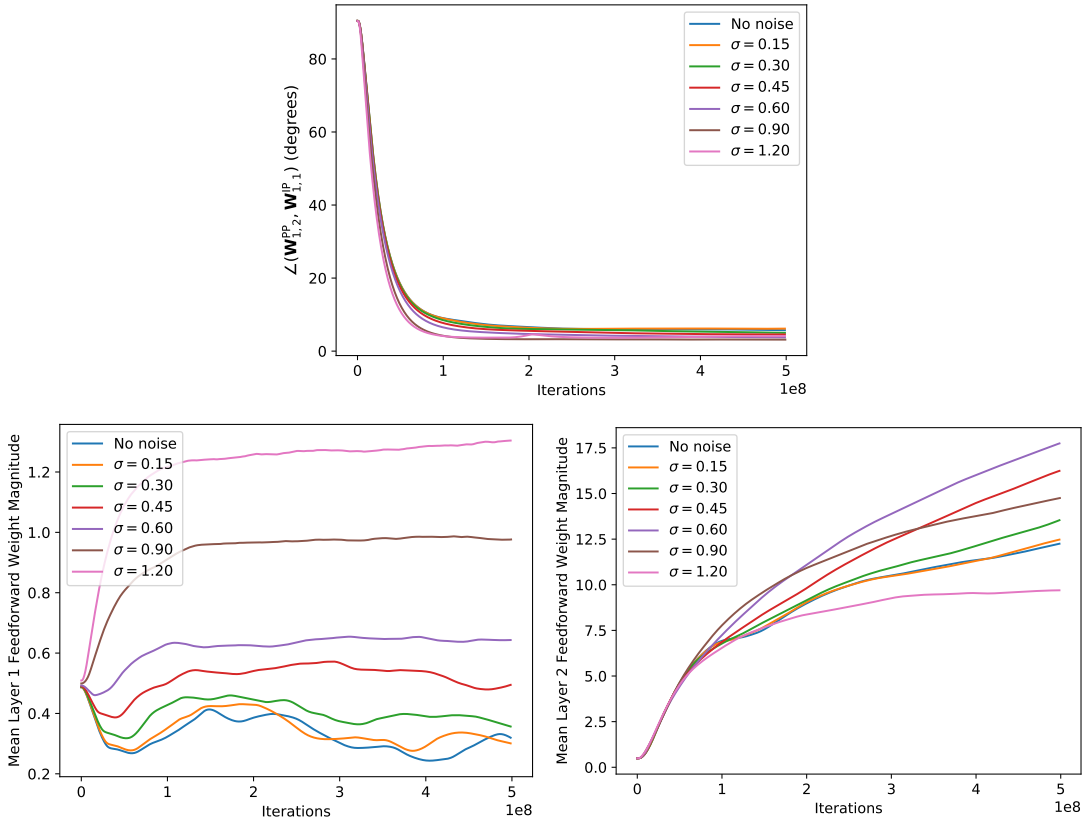

Figure 4.6: **Top:** The alignment angle between the feedforward and prediction weights. **Bottom:** The average magnitudes of the feedforward weights throughout training for layer 1 (**Left**) and layer 2 (**Right**).

The results shown here are important because they reinforce existing ideas about the significance of noise for learning within the brain and further demonstrate the robustness of the network dynamics of this model. The ability for completely uninformed, random noise to improve training speed and generalisation performance is not initially intuitive. However, it is valuable to consider when moving to more biological plausible models because many other existing mechanisms that aim to solve these problems, such as weight penalisation, cannot be applied in this context.

4.2.2 Comparison of Training With and Without Feedback Weight Plasticity

This experiment was designed with the goal of showing how the behaviour of the network dynamics using fixed feedback weights compares to the behaviour when these feedback weights are learned. These two approaches relate to the ideas of feedback alignment and target propagation and their specific differences within this model are explained in more detail in Section 3.2. Awareness of the benefits and trade-offs associated with each approach is important for gaining an understanding of how they work and how they can be used for solving different problems.

For this experiment, a simple network structure was trained to perform a classification task involving just the 4 training examples that define the `XOR` logic gate. The network contained three inputs, a single hidden layer with 20 neurons and 1 output neuron. The inputs of the training examples were encoded with the addition of a constant, $\left\{ [x_1, x_2, 1]^{\mathrm{T}} \,\middle|\, (x_1, x_2) \in \{-1, 1\}^2 \right\}$, to effectively give the network the ability to learn a bias term and the output was simply encoded as binary 1 or 0. To produce reasonable classification outputs, the rate function in use here was the logistic sigmoid function, $\phi(u) = \frac{1}{1+e^{-u}}$. The same model parameters and weight initialisation distributions were used as before in Section 4.1.1. Finally, the noise was set to $\sigma = 0.2$ and the learning rates were set to the values: $\eta^{\mathrm{PP}}_{1,0} = 0.0011875$, $\eta^{\mathrm{PP}}_{2,1} = 0.0005$, $\eta^{\mathrm{PI}}_{1,1} = 0.0011875$ and $\eta^{\mathrm{IP}}_{1,1} = 0.0059375$.

A total of six networks were trained on this task with the same weight initialisation, one using feedback alignment and the others using plastic feedback weights with different feedback learning rates. The specific values of learning rates that were used were: $\eta^{\mathrm{PP}}_{1,2} = 1.0\text{x}10^{-3}$, $5.0\text{x}10^{-4}$, $2.5\text{x}10^{-4}$, $1.25\text{x}10^{-4}$ and $6.25\text{x}10^{-5}$ where each one differs by a factor of 2. For one of the networks, Figure 4.7 shows the somatic potential of the single output neuron close to start of training and at the end of training to demonstrate that the model is capable of performing this task despite it not being linearly separable. Note here that each input and target value are presented for only 1000 iterations but the target value changes only every 2000 iterations due to the order of the training examples.

For the six models, a comparison for the evolution of the training errors and the

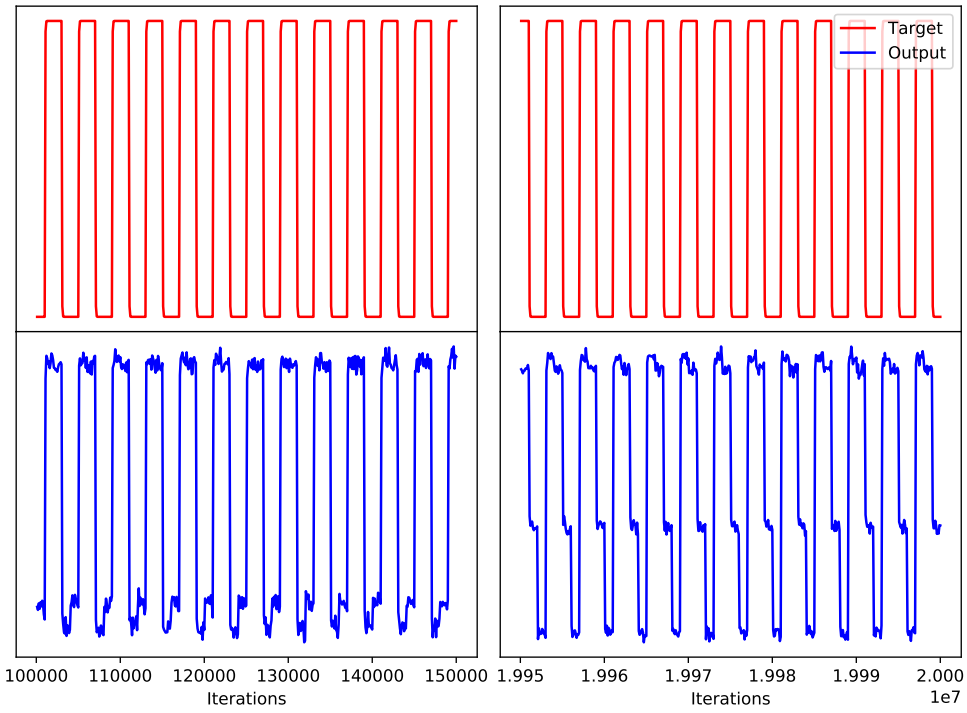

Figure 4.7: **Top:** The output neuron's somatic target potential. **Bottom:** The potential in the somatic compartment of the output neuron before (**Left**) and after (**Right**) training.

alignment angle between the feedforward and feedback weights can be seen in Figures 4.8 and 4.9 respectively. These plots show that the addition of the feedback weight plasticity adds to the instability of the network in its initial random state and this temporarily prevents the network from learning. Once the network dynamics stabilise into a self-predicting state, the presence of feedback weight plasticity results in a much faster alignment of the feedforward and feedback weights. This effect is more visible with the larger learning rates for which the speed of training also increases significantly. In all cases, the feedback plasticity leads to a larger alignment than what is achieved by feedback alignment. Interestingly this does not translate into these networks achieving the best final performance. To understand how this is possible it is worth remembering that feedback alignment is the only training scheme where the distribution of the feedback weights is preserved throughout training. This is important because the feedforward weights will always try to align with feedback weights regardless of their plasticity and movement away from their initial uniform distribution can make some useful feedforward weight configurations less likely.

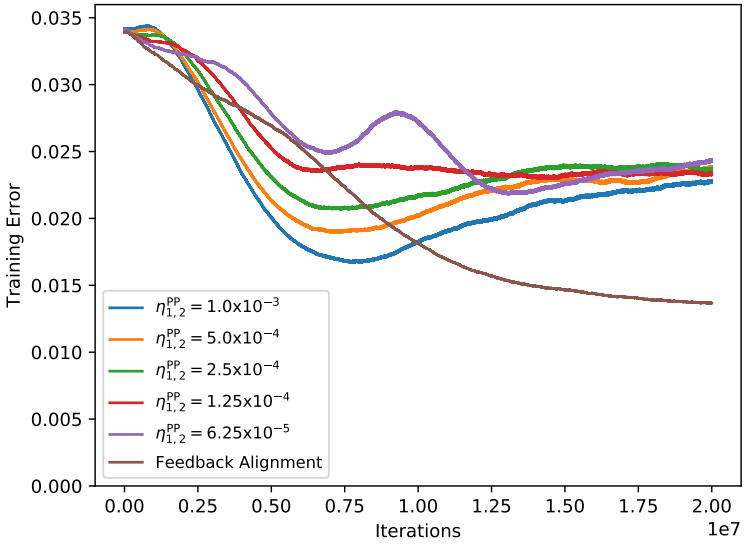

Figure 4.8: Training error for the model using feedback alignment and the five models with plastic feedback weights.

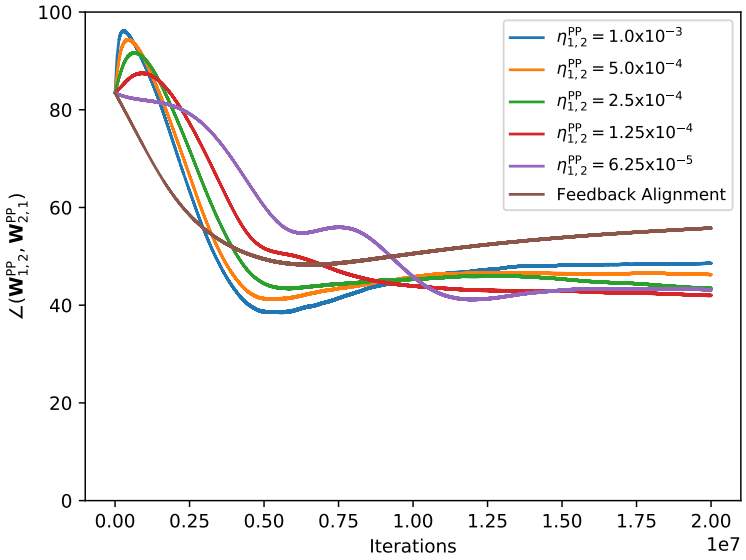

Figure 4.9: Alignment angle between the feedforward and feedback weights for the model using feedback alignment and the five models with plastic feedback weights.

4.2.3 Comparison of Network Updates with Backpropagation

In the final experiment, the primary goal was to determine whether the updates prescribed by the network dynamics were empirically similar to those prescribed by the backpropagation algorithm. Section 3.3 provides an explanation of the necessary conditions for network to produce approximately the same feedforward weight updates

to backpropagation. These conditions include the absence of any noise, a perfect self-predicting state and holds only in the limit as the nudging conductance tends to 0. None of these three conditions are fully met under normal training conditions. Removing the injected noise or forcing weight symmetry within the circuit decreases biological plausibility of the model significantly and reducing the nudging conductance results in much weaker training signals that greatly increase training time. Under normal conditions, the connection to backpropagation has not been quantified.

This experiment utilised the same network structure, parameters and non-linear regression task as before in Section 4.1.2. To provide clearer results and reduce fluctuations in the updates, the standard deviation of the noise was decreased by a factor of 10 to $\sigma = 0.03$. The network was trained using feedback alignment and with a slightly modified rate function that produces less sparsity within the network activity, $\phi(u) = 0.1 \cdot \log(1 + \exp(u - 1.0))$. At regular intervals throughout training, 500 iterations from the start of showing each training example to the network, the simulation was stopped and the network's feedforward weights were saved. These weights were then copied into a standard neural network model with the same number of layers and neurons, the same rate function and no bias terms. The backpropagation algorithm was then performed with respect to a sum of squares loss function and the computed weight updates for each layer, δ_{BP}, were saved. Before resuming the simulation, the dendritic error network's feedforward weight changes, δ_{DEN}, were also computed for the current iteration by exponentially smoothing all previous weight changes with time constant of 300 iterations.

Figures 4.10 and 4.11 show the angle between the vectors δ_{BP} and δ_{DEN} throughout training for the first and second layer respectively. Both plots show significant fluctuations in this angle which is not unexpected due to the constantly changing state of the network. Looking at the exponential moving averages for the angle of these updates in both layers one can see that this average is consistently below 90°. This is important because small updates made below an angle of 90° will necessarily move the network weights closer to where a backpropagation update would take them. Since feedback alignment was used here, seeing full alignment of the two updates would not be expected. However, the updates to the second layer weights are noticeably more aligned than those in the first layer which points to differences in how the earlier layer weights are learned.

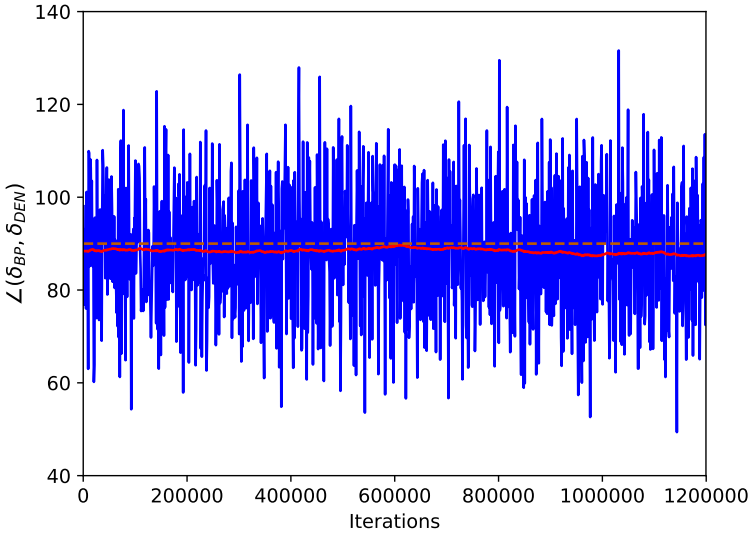

Figure 4.10: Alignment angle between the updates induced by the dendritic error network and the updates computed using standard backpropagation in the first layer.

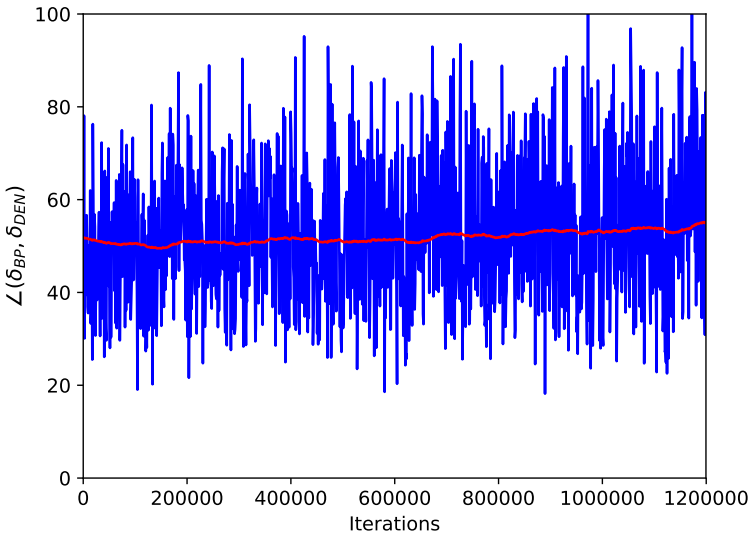

Figure 4.11: Alignment angle between the updates induced by the dendritic error network and the updates computed using standard backpropagation in the second layer.

Chapter 5

Conclusions and Future Directions

In this work, an implementation for a dendritic error model has been created and its correctness has been successfully validated through the comparison of several properties with the originally presented results. Additional analysis has also been conducted to gain new insights into how the dynamics change under different training conditions.

From the first experiment in which the strength of the injected noise was modified, a relationship was found between the noise level and the network's training speed where larger noise values led to faster training. It was also shown that it provided a regularisation effect for the weight changes that resulted in improved generalisation performance for the networks subjected to a moderate noise levels when compared with the noiseless model. From a biological perspective the noise simply represents unexplained background activity. This suggests that the presence of this activity is actually useful and may indicate the existence of a mechanism in the brain to modulate this variability. This could allow for the control of the training speeds and the regularisation trade-off.

In the second experiment it was shown that feedback weight plasticity results in faster alignment with the feedforward weights and that this initially leads to faster training but could also lead to worse final performance. This can be understood by observing that changes to the feedback weights may allow the initial uniform feedback weight distribution to be unstable under the new network dynamics. This can lead to situations where the only stable feedback weight configurations are states where performance is degraded. A more thorough analysis comparing the two approaches for different network structures and tasks is left as future work.

In the final experimental result it was show that, on average, the network updates were positively aligned to those prescribed by backpropagation and that this alignment was stronger in the final layer than in the hidden layer. This is an important result to show empirically as the link to the backpropagation algorithm had only previously been shown analytically. A future direction to expand on this idea could be to train deeper and deeper networks to see if later layers are always more aligned with the updates backpropagation. It might also be informative to explore the effect of reducing the somatic nudging potential as this should provide updates closer to backpropagation in a self-predicting state.

For the implementation there are several changes that could be made to improve its operating efficiency. The most notable downside of the implementation is that it runs entirely on the CPU. Rewriting the dynamics simulator to use a library such as PyTorch or TensorFlow would allow for GPU to be utilised and open up new possibilities for training larger networks for longer training periods. There are number of potential extensions to the model that would interesting to explore. One example would be to replace the rate functions with samples from Poisson processes to obtain spiking network dynamics. This would incredibly significant as this would address the major remaining source of biological implausibly.

Bibliography

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017. ISSN 00010782. doi: 10.1145/3065386. URL http://dl.acm.org/citation.cfm?doid=3098997.3065386.

[4] Felix A Gers, Jrgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. 1999.

[5] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL https://www.nature.com/articles/323533a0.

[6] Suzana Herculano-Houzel. The Human Brain in Numbers: A Linearly Scaled-up Primate Brain. *Frontiers in Human Neuroscience*, 3, November 2009. ISSN 1662-5161. doi: 10.3389/neuro.09.031.2009. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2776484/.

[7] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the Limits of Weakly Supervised Pretraining. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision ECCV 2018*,

pages 185–201, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01216-8.

[8] D. L. K. Yamins, H. Hong, C. F. Cadieu, E. A. Solomon, D. Seibert, and J. J. DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, June 2014. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1403112111. URL http://www.pnas.org/cgi/doi/10.1073/pnas.1403112111.

[9] Joo Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. *arXiv:1810.11393 [cs, q-bio]*, October 2018. URL http://arxiv.org/abs/1810.11393. arXiv: 1810.11393.

[10] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL http://www.sciencedirect.com/science/article/pii/0893608089900208.

[11] Pieter R. Roelfsema and Anthony Holtmaat. Control of synaptic plasticity in deep cortical networks. *Nature Reviews Neuroscience*, 19(3):166–180, February 2018. ISSN 1471-003X, 1471-0048. doi: 10.1038/nrn.2018.6. URL http://www.nature.com/doifinder/10.1038/nrn.2018.6.

[12] Amy M. LeMessurier and Daniel E. Feldman. Plasticity of population coding in primary sensory cortex. *Current Opinion in Neurobiology*, 53:50 – 56, 2018. ISSN 0959-4388. doi: https://doi.org/10.1016/j.conb.2018.04.029. URL http://www.sciencedirect.com/science/article/pii/S0959438818300199.

[13] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards Biologically Plausible Deep Learning. *arXiv:1502.04156 [cs]*, February 2015. URL http://arxiv.org/abs/1502.04156. arXiv: 1502.04156.

[14] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7:13276, Novem-

ber 2016. ISSN 2041-1723. doi: 10.1038/ncomms13276. URL https://www.nature.com/articles/ncomms13276.

[15] Qianli Liao, Joel Z. Leibo, and Tomaso Poggio. How Important is Weight Symmetry in Backpropagation? *arXiv:1510.05067 [cs]*, October 2015. URL http://arxiv.org/abs/1510.05067. arXiv: 1510.05067.

[16] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference Target Propagation. *arXiv:1412.7525 [cs]*, December 2014. URL http://arxiv.org/abs/1412.7525. arXiv: 1412.7525.

[17] Adi Ben-israel and Thomas N. E. Greville. *Generalized Inverses: Theory and Applications*. 2001.

[18] Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79, 1999.

[19] Will Greedy. MSc Project, 2019. URL https://github.com/willgreedy/msc-project.

[20] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs]*, July 2012. URL http://arxiv.org/abs/1207.0580. arXiv: 1207.0580.

[21] Hyeonwoo Noh, Tackgeun You, Jonghwan Mun, and Bohyung Han. Regularizing Deep Neural Networks by Noise: Its Interpretation and Optimization. *arXiv:1710.05179 [cs]*, October 2017. URL http://arxiv.org/abs/1710.05179. arXiv: 1710.05179.