# Predictive Coding and Biologically Plausible Neural Networks

Bachelor Project

Anders Bredgaard Thuesen
s183926@student.dtu.dk

January 2022

## Abstract

Write abstract here.

## Preface

Thank you to a lot of people here.

# Contents

# 1 Introduction

In the recent years, deep learning has shown impressive results due to the availability of massive parallel compute and huge amounts of data. From the initial biological inspiration of the brain's neurons to the perceptron where data inputs are weighted, summed together and thresholded, several new modern architectures, like recurrent [14], residual [11] and transformer neural networks [31] have pushed the limits and achieved state of the art results in speech recognition, computer vision and natural language understanding. Although deep learning has shown impressive results in the last decade, training state-of-the-art models requires an increasing amount of compute and energy resources often only available to large corporations and research institutions as well as innumerable training examples that are often expensive to obtain. Is now is the time to evaluate on what we have learned in the last decade of deep learning and begins to look for alternative, more efficient methods? One source of inspiration is to again look at the brain, still considered state-of-the-art in generalization, data- and energy-efficiency. Today, most success of deep learning has been in supervised learning problems. It is not, however, currently known how the brain would have access to labels on what is a cat and what is a dog in the same way. Perhaps a more plausible scenario is that the brain uses self-supervised learning to predict the future from past experiences. Optimizing deep learning for self-supervised learning might offer an appealing replacement for the vast amount of labeled data needed today. However, self-supervised learning is just one part to this. Equally important is how the learning is actually done. By studying the biological constraints of the brain and the interconnection between neurons it might be possible to discover more efficient learning mechanism and implementations for improving deep learning. Researchers have proposed alternative methods that aim to make different parts of deep learning more biological plausible, but so far little work has been put into combining these methods and showing the results.

## 1.1 Related Work

Several approaches have been made to make deep learning more aligned with how the brain learns. These can generally be divided into two types of categories. The first category consists of methods that aim to make the learning phase biologically plausible by only relying on local weight updates. This is in alignment with the Hebbian learning theory [12] from the neuroscience literature which states that the synaptic plasticity is only dependent on the pre- and post-synaptic activity, possibly modulated through some global signaling mechanism (ex. dopamine). Recently, methods such as (Direct) Feedback Alignment (DFA/FA) [25] [21], has been proposed as a biological plausible alternative to back-propagation, capable of scaling to state-of-the-art deep learning tasks and architectures. DFA and FA aim to mitigate the symmetric weight problem of back-propagation problematized by Bengio et al [1], by back-propagating prediction errors through random weights. This happens either directly from the output layer to each hidden layer in the case of DFA or sequentially back through the network in standard FA. Other biological alternatives to back-propagation include Equilibrium Propagation (EP) [5] for training energy-based models, which claims to be well suited for neuromorphic hardware implementations. Training using EP occur in two phases. In the first phase, the network is presented with the input of a training pair. After the network comes to an equilibrium, the prediction of the network is then nudged towards the correct label. When the network relaxes in the second phase, the weights are then updates according to the local activity differences in each neuron that implicitly encode

the gradient of the loss function. EP, however, requires connection weights to be symmetric and access to previous neuron activity after the network has come to an equilibrium in the second phase, which is not local in time. Continual Equilibrium Propagation (C-EP) [6] fixes this issue by simultaneously updating the weights while the network reaches equilibrium in the second phase, but still requires symmetric weights. To address this issue, the same authors adapt C-EP to systems with asymmetric connection weights and call this method the Continual Vector Field (C-VF) approach. A third kind of biological alternative to back-propagation is Predictive Coding Networks (PCN) and Inference Learning (IL) [32] inspired by the seminal neurocomputational theory of perception by Rao and Ballard [27]. PCNs have in the past couple of years shown to approximate the backprop algorithm and even under certain conditions implement it exactly using Zero-divergence Inference Learning (Z-IL) [30] and work on arbitrary computation graphs [29] [24]. Work has also been done on higher level models inspired by predictive coding, such as PredNet [22], whose architectural design resembles that of predictive coding networks, but is trained using back-propagation.

The second category consists of methods that aim to make the communication between neurons more biologically plausible and possible more efficient on low-powered neuromorphic hardware such as the Intel Loihi or IBM TrueNorth chips by relying on binary spikes for communication. SpikeProp [3] is one of the early proposed methods for training spiking neural networks. The method uses an algorithm akin to back-propagation directly on the generated spikes using surrogate gradient functions. Another approach is to use a recurrent neural network type architecture to model the dynamics of neurons and trained with back-propagation using snnTorch framework [7]. A third approach is the method of shadow training, where standard ANNs are training using a neural response function of the spiking frequency as function of the input stimuli and later run in simulation or hardware using the exact neuronal dynamics [16].

This project will primarily focus on PCN and shadow training methods presented in the two papers *Spiking Deep Networks with LIF Neurons* by Eric Hunsberger et al. [16] and *Can the Brain Do Backpropagation? - Exact Implementation of Backpropagation in Predictive Coding Networks* by Yuhang Song et al. [30]. The endgoal of this project is to compare these methods with a unification of the two on the supervised task of predicting the label of MNIST digits. Hopefully, the combination of these methods will shine some light on the intersection between artificial and biological neural networks and provide a foundation for further research.

## 1.2 Research Questions

The project will address the following three research questions:

- **Does the combination of predictive coding networks and spiking neural networks provide a biologically plausible model of learning**

  We examine the biological plausibility of the novel combination of predictive coding networks, inference learning and spiking network shadow training by evaluating it against biological constraints defined by Bengio et al. [1] and later expanded upon by Hunsberger [17] together with constraints of locality, spiking and unidirectionality derived from biological neurons.

- **How well does the proposed biologically plausible model perform compared to deep learning methods?**

  The performance of the model and learning algorithm is then compared against feed-forward neural networks trained with back-propagation on the MNIST benchmark dataset.

- **In what ways can deep learning methods benefit from biological plausible learning?**

  We discuss different possibilities for improving the training of deep neural networks with biological plausible methods and how this work contributes to this.

## 1.3 Project Outline

The thesis is structured in three parts. In the first chapter, deep learning methods are introduced and reviewed to provide a foundation for the second part of the thesis. The second part will begin by defining what biological plausibility entails and what constraints must be satisfied for models and methods to be biological plausible. The chapter will then go on to introducing predictive coding networks, inference learning and spiking neural network shadow training, claimed to be more biological plausible than current deep learning methods. Finally, in the results and discussion section of the thesis, a novel combination of the previous mentioned, self-acclaimed biological plausible methods will be investigated according to the research questions above. Finally, the thesis will conclude.

All code from this project is available at https://github.com/andersthuesen/bachelor-project.

# 2 Deep Learning

To give an understanding of the current deep learning landscape and de facto standard methods, this first part of the thesis will go into the different aspects of training deep neural networks. It is by no means meant to be an exhaustive review, and will primarily be focused on the tasks of supervised learning and simple feed-forward models. Hopefully, this will provide a firm foundation for the second part of the thesis when alternative biological plausible methods will be introduced and compared against. We begin by generically defining our dataset, central to supervised learning, from which we would like to train our neural network to learn a mapping from input to output variables. As we will later be training our networks to predict the written digits in the MNIST dataset, our focus will primarily be on discriminative models and classification tasks. The method of maximum likelihood estimation as a means of optimally estimating parameters of statistical models will be introduced and shown to output the particular loss functions used in deep learning. With connections to error units in predictive coding networks, introduced in the latter part of this thesis, we show how derivatives of loss functions in both regression and classification settings can be defined in terms of prediction residuals. Finally, back-propagation, crucial in training deep neural networks, will be derived for feed-forward models along with batch normalization for efficiently doing so.

## 2.1 Mathematical Notation

Throughout this thesis, we will be using **boldface** and CAPITALIZATION when dealing with **MATRICES**, eg. $\mathbf{A}$ and **vectors**, eg. $\mathbf{x}$. To avoid irrelevant interruptions in derivations, vectors of lowercase letters will implicitly represent columns in their corresponding matrices unless otherwise stated, eg. $\mathbf{x}_i$ will be the $i$'th column in the matrix $\mathbf{X}$. The same is the case for non-boldface scalers and their corresponding vectors, eg. $x_j$ will be the $j$'th index in vector $\mathbf{x}_i$. The hat symbol will be used to indicate the prediction of a variable, eg. $\hat{y}$ will be the prediction of $y$.

## 2.2 Dataset

We define our dataset, $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ for $i = 1 \dots N$ consisting of $N$ pairs of datapoints, $\mathbf{x}_i \in \mathbb{R}^k$ and $\mathbf{y}_i \in \mathbb{R}^l$ where $N$ is the size of the dataset. We notice, that both $\mathbf{x}_i$ and $\mathbf{y}_i$ can be vectors of possibly differently dimensions $k$ and $l$ respectively. $\mathbf{x}_i$ might represent eg. an image as it is the case with the MNIST dataset used later in this project that consists of 60.000 training examples and 10.000 test examples of 28x28 images depicting handwritten digits and their corresponding labels [20]. As the images in the MNIST dataset are two-dimensional, the image has to be flattened into a one-dimensional vector. In the case of supervised learning, the objective is from $\mathbf{x}_i$ to predict the corresponding label, $\mathbf{y}_i$ which would amount to a single scalar number from 0 to 9 in the MNIST dataset.

## 2.3 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is method of finding the optimal parameters, $\theta$, of some assumed probability distribution or model, $P$, in order to best describe the observed data, $\mathbf{X}$. Statistically this can be formulated as $P(\mathbf{X}; \theta)$ or more concisely $P_\theta(\mathbf{X})$. Assuming that the observed data is independent and identically distributed, the likelihood estimator

function can be defined as:

$$\mathcal{L}_\theta(\mathbf{X}) = \prod_{i=1}^{N} P_\theta(\mathbf{x}_i). \tag{1}$$

MLE can then be formulated as the optimization problem:

$$\theta^\star = \underset{\theta}{\operatorname{argmax}} \;\; \mathcal{L}_\theta(\mathbf{X}). \tag{2}$$

where the goal is to find the optimal set of model parameters (maximum likelihood estimate), $\theta^\star$, that maximizes the probability of the observed data (the likelihood). In the case of supervised learning it is often more interesting to model the conditional probability function $P_\theta(\mathbf{Y}|\mathbf{X})$ where $\mathbf{Y}$ might be some label of $\mathbf{X}$. Fortunately, MLE can be generalized to work with conditional probabilities [8] using the conditional likelihood estimator:

$$\mathcal{L}_\theta(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{N} P_\theta(\mathbf{y}_i \mid \mathbf{x}_i). \tag{3}$$

The MLE task is then to find $\theta^\star = \operatorname{argmax}_\theta \; \mathcal{L}_\theta(\mathbf{Y}|\mathbf{X})$. Often, it is easier to instead minimize the negative log-probability as taking the logarithm of a product yields a sum of logs:

$$-\log \mathcal{L}_\theta(\mathbf{Y} \mid \mathbf{X}) = -\sum_{i=1}^{N} \log P_\theta\left(\mathbf{y}_i \mid \mathbf{x}_i\right). \tag{4}$$

As the logarithm is a monotonic increasing function, minimizing the log-probability is essentially the same as maximizing the probability itself. By assuming that $\mathbf{Y}$ follows a Gaussian distribution $P_\theta(\mathbf{Y} \mid \mathbf{X}) = \mathcal{N}(\hat{\mathbf{Y}}(\mathbf{X}), \sigma^2)$ with a mean of $\hat{\mathbf{Y}}$ and a fixed variance of $\sigma^2$ the log likelihood function can be written as:

$$\begin{aligned} -\log \mathcal{L}_\theta(\mathbf{Y} \mid \mathbf{X}) &= -\sum_{i=1}^{N} \log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{2\sigma^2}} \\ &= \sum_{i=1}^{N} \frac{1}{2}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \frac{1}{\sigma^2} + \log\left(\sigma\sqrt{2\pi}\right). \end{aligned} \tag{5}$$

If all constant terms are dropped it becomes apparent that minimizing the mean squared error, $\text{MSE} = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}(y_i - \hat{y}_i)^2$, performs maximum likelihood estimation under the assumption that the data is normally distributed. To optimize this quantity with gradient based method it is necessary to compute the partial derivative:

$$\frac{\partial}{\partial \hat{\mathbf{Y}}} \text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \hat{\mathbf{y}}_i - \mathbf{y}_i \tag{6}$$

It is often of interest in computer vision to classify images into several categories. Examples hereof could be monitoring for explicit content on social media platforms, making images accessible for visually impaired by detecting and reading aloud the image contents or detecting pedestrians in self-driving cars. Although the last two examples are more advanced use cases of computer vision, namely scene recognition and image segmentation, they all rely on the ability to classify images or objects therein.

It is possible to use artificial neural networks in classification tasks by transforming the output logits, $\mathbf{z} = (z_1, \dots z_K) \in \mathbb{R}^K$, into a discrete probability distribution using the softmax vector function:

$$\text{softmax}(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } k = 1, \dots, K \tag{7}$$

The softmax function scales every $z_k$ to lie in the range from 0 to 1 and makes the sum of all output probabilities equal to 1. It thereby satisfies the formal definition of probability mass functions. An important property of the softmax function is that the ordering of logits and their respective output probabilities stays the same. Eg. the largest value element of $\mathbf{z}$ also has the largest output probability.
If the label is one-hot encoded, meaning that the element in the vector with index corresponding to the correct class set to one and the rest to zero, $\mathbf{Y}$ would have distribution:

$$p(\mathbf{y}_i \mid \mathbf{x}_i)_k = \begin{cases} 1 & \text{if } c_i = k \\ 0 & \text{otherwise} \end{cases} \quad \text{for } k = 1, \dots, K \tag{8}$$

where $c_i$ is the correct class label for $\mathbf{x}_i$. Neural networks with softmax activation in the last layer can be trained by minimizing the cross-entropy between the one-hot encoded label, $p$, and the output distribution of the network, $q$, given by:

$$H(p, q) = -\sum_{i=1}^{N} \sum_{k=1}^{K} p(y_i \mid x_i)_k \log q(y_i \mid x_i)_k. \tag{9}$$

Since $p(y_i \mid x_i)_k$ is only 1 when $k = c_i$, the inner summation over $k$ can be compressed to:

$$H(p, q) = -\sum_{i=1}^{N} \log q(y_{i,c_i} \mid x_i). \tag{10}$$

By comparison with equation 4, minimizing the cross-entropy can be seen as performing maximum likelihood estimation. To actually minimize the cross-entropy using gradient based methods such as back-propagation, both functions have to be differentiable. The Jacobian of the softmax function is a $K$ by $K$ matrix:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial s_1}{\partial z_1} & \cdots & \frac{\partial s_1}{\partial z_K} \\ \vdots & \ddots & \vdots \\ \frac{\partial s_K}{\partial z_1} & \cdots & \frac{\partial s_K}{\partial z_K} \end{pmatrix} \tag{11}$$

where each $s_i$ is the $i$'th entry of the softmax output. As each entry is positive, the partial derivative of the logarithm of $s_i$ wrt. $z_i$ can be written as:

$$\frac{\partial}{\partial z_j} \log s_i = \frac{1}{s_i} \frac{\partial s_i}{\partial z_j} \Rightarrow \frac{\partial s_i}{\partial z_j} = s_i \cdot \frac{\partial}{\partial z_j} \log s_i. \tag{12}$$

From the softmax function definition in 7 it follows that:

$$\log s_i = \log \left( \frac{e^{z_i}}{\sum_{l=1}^{K} e^{z_l}} \right) = z_i - \log \sum_{l=1}^{K} e^{z_l} \tag{13}$$

such that the partial derivative of the above becomes:

$$\frac{\partial}{\partial z_j} \log s_i = \frac{\partial z_i}{\partial z_j} - \frac{1}{\sum_{l=1}^{K} e^{z_l}} \cdot \frac{\partial}{\partial z_j} \sum_{l=1}^{K} e^{z_l}. \tag{14}$$

Here, $\frac{\partial z_i}{\partial z_j}$ will be equal to 1 if $i = j$ and otherwise 0, which can be denoted by $1\{i = j\}$. Because $e^{z_l}$ in the sum only depends on $z_l$, the partial derivative of terms where $l \neq j$ is 0. The above can now be rewritten as:

$$\frac{\partial}{\partial z_j} \log s_i = 1\{i = j\} - \frac{e^{z_j}}{\sum_{l=1}^{K} e^{z_l}} = 1\{i = j\} - s_j \tag{15}$$

in order to finally reveal:

$$\frac{\partial s_i}{\partial z_j} = s_i \cdot (1\{i = j\} - s_j). \tag{16}$$

We will now consider the partial derivative of the cross-entropy function:

$$\frac{\partial}{\partial z_j} H(p, q) = -\sum_{i=1}^{K} y_i \cdot \frac{\partial}{\partial z_j} \log s_i \tag{17}$$

and substitute in $\frac{\partial}{\partial z_j} \log s_i$ from equation 15 to get:

$$= -\sum_{i=1}^{K} y_i \cdot (1\{i = j\} - s_j) \tag{18}$$

which can be split into the two sums:

$$= \sum_{i=1}^{K} y_i \cdot s_j - \sum_{i=1}^{K} y_i \cdot 1\{i = j\}. \tag{19}$$

As the indicator function $1\{i = j\}$ is only 1 when $i = j$ the above can be simplified as:

$$= \sum_{i=1}^{K} (y_i \cdot s_j) - y_j \tag{20}$$

and as $s_j$ does not depend on $i$, it can be pulled out of the sum:

$$= s_j \sum_{i=1}^{K} y_i - y_j. \tag{21}$$

Finally, since $y_i$ sums to 1, the partial derivative of the cross-entropy wrt. $z_j$ becomes:

$$\frac{\partial}{\partial z_j} H(p, q) = s_j - y_j. \tag{22}$$

Interestingly, in both regression and classification settings it turns out that the derivative of the loss functions (equation 6 and 22) can be formulated as the residual of the prediction and the label. This may be intuitive in the case of regression where continuous values are predicted, but not so much in the case of classification. In order to grasp why this is still the case, it might be useful to imagine a binary classification problem. In the case of a false positive ($s_j = 1$, $y_j = 0$), the derivative of the cross-entropy will be positive, meaning that further increasing $s_j$ (although not possible due to softmax) will further increase the cross-entropy. We therefore update our parameters in the negative direction of the gradient.

## 2.4 Feed-forward Neural Networks

Feed-forward neural networks (FNN) are considered the simplest kind of neural network where the connections between the nodes does not allow for any cycles or recurrent connections. Feed-forward neural networks are divided into several layers, where input data from the first layer is "feed forward" through the so-called hidden layers to the final output layer of the network, considered the prediction of the network. FNNs are typically fully connected networks which entails that every node of the network is connected to every node in the previous layer. One special case of FNNs is that of when there are no hidden layers in the network and the input layer is linearly transformed to the output layer and "activated" through a softmax or sigmoid function, depending on the output of output nodes. In that case, the FNN will correspond to (multinomial) logistic regression. This correspondence incentives the use of activation functions after each linear transformation of the layers, as the network would otherwise not be able to describe non-linearities in the data. Historically, the sigmoid activation function $\sigma(z) = (1 + \exp(-z))^{-1}$ has been the go-to activation function, but recently the rectified linear unit, $\text{ReLU}(z) = \max(0, z)$, has become the de facto standard.

A feed-forward neural network with $L - 2$ hidden layers is parameterized by the weight matrices $\mathbf{W}^{(l)} \in \mathbb{R}^{m \times n}$ and biases $\mathbf{b}^{(l)} \in \mathbb{R}^m$ for $l = 2 \dots L$ where $n$ is the input dimension of the layer and $m$ the output dimension. A feed-forward pass from layer $l - 1$ to layer $l$ is given by $\mathbf{a}^{(l)} = \sigma(\mathbf{z}^l)$ where $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$ and $\sigma$ is the chosen activation function. By letting the initial activation $\mathbf{a}^{(1)} = \mathbf{x}_i$ one can consider the final activation of the network the prediction of the network $\hat{\mathbf{y}}_i = \mathbf{a}^{(L)}$.

## 2.5 Weight initialization

We initialize the weight matrices using Kaiming He initialization, where the entries of the matrix $W_{ij}^{(l)}$ are drawn from a normal distribution with zero mean and $\sqrt{2/n}$ standard deviation as this will help reduce vanishing and exploding gradient problem by keeping the variance in each layer equal when using ReLU activation. [10]

## 2.6 Back-propagation

The working horse of almost all modern deep learning models is the back-propagation (aka. backprop) algorithm first popularized for training neural networks by Rumelhart, Hinton & Williams in 1986 [28]. The algorithms solves what is referred to as the *credit-assignment problem*. When learning the parameters of an artificial neural network we would like to know how changing a weight in the network contributes to the total loss, in order to change it in the direction that minimizes the loss. One naive way to do this would be simply to adjust a single random weight slightly, evaluate the new neural network on the dataset and observe the effect on the model loss. If the change leads to a decrease in loss, keep the change, otherwise repeat from the beginning. This would however be very computationally expensive, since the network would have to be evaluated on the entire dataset for each weight in the network. Fortunately, the backprop algorithm achieves this much more efficiently as we will see in the following section.

Back-propagation is an efficient method for calculating the weight updates that minimizes some loss function, $\mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i)$, which measures the difference between the predicted output

of the network, $\hat{\mathbf{y}}_i$, and the true output, $\mathbf{y}_i$. Examples of loss functions are squared error $\sum \frac{1}{2}(\hat{\mathbf{y}}_i - \mathbf{y}_i)^2$, typically used for regression and categorical cross entropy $-\sum \mathbf{y}_i \cdot \log(\hat{\mathbf{y}}_i)$ for classification. At its core, the back-propagation algorithm is simply applying the chain rule on the partial derivate of the loss function with respect to the parameters and realizing that a lot of computation can be reused or "back propagated" in order to calculate the weight and bias updates for earlier layers. It is therefore necessary that the loss function is differentiable with respect to the prediction variable. Some alternatives to back-propagation exist such as Direct Feedback Alignment [25], but is not commonly used in practice.

To demonstrate the efficiency of the back-propagation algorithm, one can consider the last and second to last layers of the network. For now, the demonstration will only consider the weights as the bias terms can integrated into the weight matrices by extending the output dimension $n$ by 1 and appending a 1 to the $\mathrm{a}^{(l)}$ vectors resulting which will give an equivalent result. Applying the chain rule on the partial derivative of the loss function wrt. $\mathbf{W}^{(L)}$ yields

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}}}_{\delta^{(L)}} \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{W}^{(L)}} = \underbrace{\mathcal{L}'(\hat{\mathbf{y}}_i)\sigma'(\mathbf{z}^{(L)})}_{\delta^{(L)}} \mathbf{a}^{(L-1)} \tag{23}$$

whose factors can be divided into the error term, $\delta^{(L)}$, and activation in the previous layer, $\mathbf{a}^{(L-1)}$. Yet again, applying the chain rule on the partial derivative of the loss function, but this time wrt. $\mathbf{W}^{(L-1)}$ hints at the source of its efficiency:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L-1)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}}}_{\delta^{(L)}} \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{a}^{(L-1)}} \frac{\partial \mathbf{a}^{(L-1)}}{\partial \mathbf{z}^{(L-1)}} \frac{\partial \mathbf{z}^{(L-1)}}{\partial \mathbf{W}^{(L-1)}} = \underbrace{\delta^{(L)} \mathbf{W}^{(L)} \sigma'(\mathbf{z}^{(L-1)})}_{\delta^{(L-1)}} \mathbf{a}^{(L-2)}. \tag{24}$$

Though the derivation above only considers $\mathbf{W}^{(L)}$ and $\mathbf{W}^{(L-1)}$ it is the general case that:

$$\delta^{(L)} = \mathcal{L}'(\hat{\mathbf{y}}_i)\sigma'(\mathbf{z}^{(L)}), \quad \delta^{(l)} = \delta^{(l+1)}\mathbf{W}^{(l+1)}\sigma'(\mathbf{z}^{(l)}) \tag{25}$$

which can be computed using a dynamic programming approach to update the weight parameters using gradient descent with learning rate $\alpha$:

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \alpha \delta^{(l)} \mathbf{a}^{(l-1)}. \tag{26}$$

The weight is updated in the opposite direction (hence the negation) of the gradient as the aim is to minimize the loss function. By reusing the computation of the error terms, $\delta^{(l)}$, backprop is able to very efficiently compute the weight updates to minimize the global loss function. As the goal is to optimize the joint likelihood over the entire dataset, back-propagation should in theory process all datapoints in the dataset in each forward-backwards pass and update the weights using gradient descent or some other optimizer. As datasets are often very large, requiring a proportional amount of memory, this is rarely done in practice. Instead, mini-batches of data, typically containing 8 to 128 datapoints are used to update the parameters with eg. stochastic gradient descent. The assumption is that the mini-batches capture the underlying distribution of the dataset. This turns out to work really well, as taking more noisy steps in the loss landscape is often favorable compared to a single step in the direction of steepest descent for the entire dataset. A good rule of thumb when choosing batch size is to pick the larges batch size able to fit in memory when training on GPUs. This will lead to more accurate gradients while taking advantage of the computational power by processing each batch in parallel.

### 2.7 Adam optimizer

The Adam optimizer [19] is an extension to stochastic gradient descent that combines the advantages of AdaGrad [4] on sparse gradients and RMSProp [13] that incorporates root mean squared momentum. Introducing momentum is often useful as it is more robust to noise in the mini-batch gradients and therefore leads to faster convergence [26]. Adam does this by keeping track of a running average of the gradient and the square of the gradient:

$$\begin{aligned} m_t &\leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\ v_t &\leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \end{aligned} \tag{27}$$

where $\beta_1$ and $\beta_2$ are hyper-parameters, in the range $[0, 1)$, that control the momentum decay rate of the gradient. They are typically set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$ as default. The moment estimates are then bias-corrected:

$$\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1} \qquad \hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2} \tag{28}$$

and used to transform the gradient:

$$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{29}$$

where $\alpha$ is the learning rate and $\epsilon$ a small constant (typically $10^{-8}$) to prevent division by zero. Adam has been shown in general to lead to very fast convergence with good performance and is therefore used to great extend when training deep learning models.

### 2.8 Batch normalization

When training deep neural networks, it is often necessary to normalize the inputs to the network in obtain good results. By normalizing the input variables to the network the gradient of the loss function is more likely to point in the direction of the local minima resulting in more efficient and stable training. According to the authors [18], batch normalization has the effect of reducing "internal covariate shift" of the networks intermediate representations. As the weights and biases of each layer in the network is updated, the distribution of the internal representations tends to change aswell. This has the effect of slowing down learning and decrease the chance of converging to a good local minima. By normalizing each mini-batch, batch normalization is able to accelerate deep learning by enabling higher learning rates and fewer training steps. Batch normalization works by using a combination of the mean and variance of the current mini-batch and a running mean and variance. At training time, batch normalization standardizes the mini-batch according to:

$$\hat{\mathbf{x}}_i^{(l)} = \frac{\mathbf{x}_i^{(l)} - \mu_B^{(l)}}{\sqrt{{\sigma^2}_B^{(l)} + \epsilon}} \tag{30}$$

where $\mu_B$ is the mean of the batch, $\sigma_B^2$ is the variance of the batch and $\epsilon$ is some small value for numerical stability (typically $10^{-5}$). Batch norm then outputs an affine transformation using learned parameters $\gamma$ and $\beta$:

$$\mathbf{y}_i = \gamma \hat{\mathbf{x}}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}\left(\mathbf{x}_i\right). \tag{31}$$

During the training fase, the mean and variance of the dataset is estimated using an exponential moving average:

$$\begin{aligned} \mu_{\text{mov}} &\leftarrow \alpha\mu_{\text{mov}} + (1-\alpha)\mu_B \\ \sigma^2_{\text{mov}} &\leftarrow \alpha\sigma^2_{\text{mov}} + (1-\alpha)\sigma^2_B \end{aligned} \tag{32}$$

where $\alpha = 0.1$ is the default value. Then at inference time, batch normalization uses the dataset's mean and variance estimations and affine scaling and translation parameters to normalize the intermediate representations according to:

$$\mathbf{y}_i = \frac{\gamma}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}} \cdot \mathbf{x}_i + \left( \beta - \frac{\gamma\mu_{\text{mov}}}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}} \right). \tag{33}$$

# 3 Biologically plausible deep learning

The following section will focus on biologically plausible deep learning and what that entails. There are many aspects of being biological plausible, however this project will only be considering those that are computationally relevant for deep learning. Initially to gain an understanding of biology a brief summary of the neuron is presented. From this biological constraints are defined and compared with current deep learning approaches. Subsequent sections will describe alternative, more biologically plausible methods for training and evaluating artificial neural networks.

## 3.1 Biological neurons

This section aims to briefly describe the biological neuron from a highlevel perspective to give an intuition of how processing is happening in the human brain. Hopefully this will lead to a better understanding of what biologically plausibility encapsulates. It is by no means fully comprehensive nor capturing all the nuanced details of the biological processes.

A neuron is an electrically excitable cell capable of communicating with other neurons by sending electrical signals called action potentials or spikes. Most often, multiple spikes are generated in sequence, called a spike train. When spikes are generated the neuron is said to be *firing*. The neuron has a cell body called the soma which is broadly considered the central element in processing. From the soma root like structures called dendrites extrude, which capture signals from other nearby neurons. Receiving spikes from other presynaptic neurons will lead to polerization of the cell and trigger the neuron to spike itself if the input spikes exceed a certain threshold. As the neuron spikes an action potential will travel from the soma, through the axon and to the synaptic terminal where it will cause the release of neurotransmitter molecules to the postsynaptic neurons dendrites. Neurons are usually in computational models described by their membrane potential or voltage which arises from the fact that neurons maintain a voltage across its membrane. It achieves this by actively regulating the intracellular concentration of charged potassium, sodium and calcium ions through ion pumps until a resting potential is reached (around -70mV). When neurotransmitter molecules bind to receptors on the dendrites, it will cause various types of ion channels to open. This will result in a change in the ionic concentration and thereby membrane potential of the neuron. Once the membrane voltage reaches a certain level called the threshold voltage (around -55mV) voltage-gated ion channels open and ions flood in to the cell, further increasing the membrane voltage to form a chain reaction. This reaction causes neurotransmitter to be released at the synaptic terminal. In other words, the neuron will spike. After spiking a phase of hyperpolerization called the refractory period might occur. In this phase, ion pumps decrease the membrane potential beyond the resting potential and prevents the neurons from spiking.

It is believed that the brain learns both by forming new connections between neurons and by varying the amount of neurotransmitter receptors resulting in strengthening or weakening of existing connections. In the book, The *Organization of Behaviour* by Donald Hebb [12], he postulates that the pre- and postsynaptic activity drive this change, known as Hebb's postulate or Hebbian learning:

> "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change

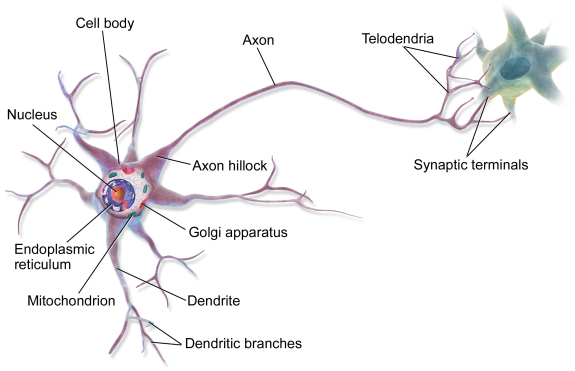



Figure 1: Illustration of biological neuron by BruceBlaus - Own work, CC BY 3.0 [1]

> takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

This postulate is congruent with the learning mechanism of spike-timing dependent plasticity (STDP) where changes in synaptic strength depend on the relative timings of the pre- and postsynaptic spikes, which has been observed to happen in the brain[2].

## 3.2 Biological constraints

To understand how the biological neurons process information and give rise to the emergent phenomenon of intelligence it is of interest to study the natural biological and physical limitations that they exhibit. One interesting question is to whether the perticular biological implementation of the neural networks in the brain are fundamental for the existense of intelligence or if it is rather a single instance in the set of possible configurations. For now, the focus will be on defining the constraints of the only tried-and-testet implementation we know of; the biological neuron. Neurons communicate using spikes of roughly the same amplitude, but are able to dynamically modulate the rate and timings of spikes depending on the presynaptic stimulation. This leads to our first **constraint of spikes** for communication. Due to the anatomical structure of neurons communication is unidirectional in the direction from the cell body, along the axon and to the synaptic terminal. No spiking signal is known to travel in the reverse direction from post-synaptic dendrites to pre-synaptic terminal. Thus we introduce a **constraint of unidirectionality**. Neurons primarily communicate with nearby neurons. Efficiency wise, it makes sense for the brain to place functionally dependent regions nearby to save energy in signal transmission. This principle of locality is also found in Hebb's postulate and STDP. We therefore say that biological learning is **constrained by locality**.

### 3.2.1 Biological violations of backprop and deep learning

In their 2016 paper, Bengio et al [1]. raises 6 problems regarding the biologically plausibility of back-propagation and artificial neural networks which are nicely summarized and expanded upon in the PhD thesis by Hunsberger [17] from which this project will use the same naming of the problems. To illustrate the problems more clearly in terms of dicussions in previous sections (2.6, 3.1) the problems have been reformulated as follows.

[1] https://commons.wikimedia.org/w/index.php?curid=28761830

1. **Weight transport problem**:

   Back-propagation uses the same connection weights in both the forward pass for prediction and in the backwards pass when calculating the gradients as seen in equation 25. A well established fact of biological neurons is that spikes travel uni-directionally from the soma, through the axons and across the synapses as neurotransmitter chemicals. Backprop does not seem to comply with this constraint.

2. **The derivative problem**:

   The derivative problem problematises the need for the derivative of the activation function (ex. sigmoid or ReLU used in the forward pass) in order to back-propagate the error signal as it is also the case in equation 25. It is not known that biological neurons can do this.

3. **The linear feedback problem**:

   The linear feedback problem is due to the fact that neurons communicate in a non-linear fashion making the linear dependence on the feedback error term, $\delta^{(l)}$, in equation 26 difficult to implement for biological neurons.

4. **The spiking problem**:

   Biological neurons communicate with spikes whereas artificial neural networks use real values. It might be possible in a sense to communicate both positive and negative values stochastically by encoding the value in the spike rate with a combination of excitory and inhibitory neurons, however back-propagation depends on differentiable activation functions to work.

5. **The timing problem**:

   In the back-propagation algorithm, forwards and backwards passes are run alternatingly and sequantially, which is not known to be happening in the human brain. In constrast, neurons function asynchrously when activated by their inputs and spikes travel between neurons with some propagation delay, making it difficult for biological neurons to implement backprop as activations and their derivates should stay constant when performing weight updates.

6. **The target problem**:

   It is unclear how labels such as those in the MNIST dataset could be present in the brain. The brain seems to make sense of the world by itself without needing constant supervision. Humans generally only need a few examples in order to identify different objects. Though supervised learning has shown the most impressive results thus far, recent focus on unsupervised or self-supervised learning has startet to show its promise, which might better explain how the brain learns.

## 3.3 Spiking neural networks

Spiking neural networks (SNNs) are a particular type of network that uses spikes for inter-neuron communication. In that way they more closely mimic biological neural networks, but share their structure and dense connectivity with feed-forward neural networks. Spiking neurons are temporal in their dynamics in the sense that they are integrating input spikes over time. When the threshold membrane potential is exceeded a spike is sent to the connected neurons in the next layer. The implementation of SNNs requires a particular neuron model of the action potential often modelled as a first order differential equation describing the exact dynamics. Several different neuron models exists ranging from simple and compute efficient models like the Integrate-and-Fire (IF) model that only captures the coarse dynamics of biological neurons to more nuanced models like the Hodgkin-Huxley model [15]. We will be using a variation of the IF neuron model which leaks membrane potential over time, called the Leaky-Integrate-and-Fire model. This particular model has the advantage of being relatively biologically plausible yet still being very efficient to compute.

### 3.3.1 Training spiking neural networks

There exist several methods for training SNNs such SpikeProp [3] and snnTorch [7] mentioned earlier. This project will use a method of shadow training where a "shadow" ANN will be trained using spike-rate coding and later run as spiking neural network at inference time using the learned weights from the ANN. This has the advantage of leveraging existing deep learning infrastructure and optimization techniques while being deployable on low-powered or analog hardware.

### 3.3.2 Leaky-Integrate-and-Fire neurons

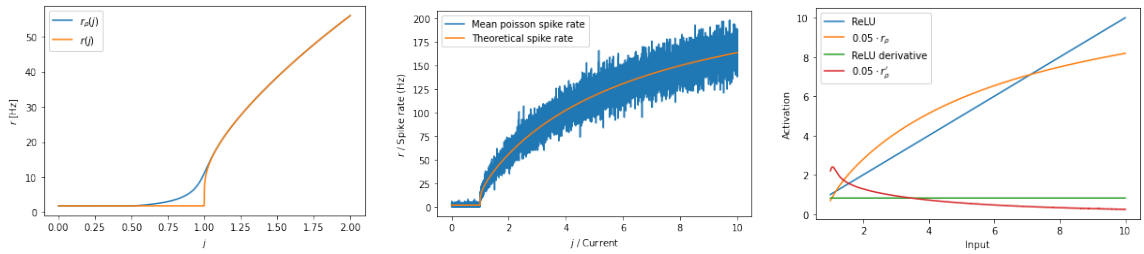
Leaky-Integrate-and-Fire neurons were inspired by the fact that biological neurons seem to lose voltage over time as ions leak through the membrane. Computationally this might be understood as a temporal filter that decreases the effect on the action potential of spikes happened longer time back in the past. The Leaky-Integrate-and-Fire (LIF) neuron model can be described by the following differential equation:

$$C\frac{dV}{dt} = J(t) - \frac{V}{R} \tag{34}$$

where $V$ is the voltage across the membrane, $R$ the membrane resistance and $J(t)$ the input current to the neuron at time $t$. The neuron will fire when the membrane voltage reaches a threshold of $V_{\text{th}}$ and reset to $V_{\text{rest}}$ in a refractory period of $t_{\text{ref}} = 0.004$. If the applied input current is not large enough to make the neuron spike, the membrane voltage will naturally decay to $V_{\text{rest}}$. Hunsberger [17] shows how the above can be normalized and rewritten in simpler terms as:

$$\tau_{RC}\frac{dv(t)}{dt} = j(t) - v(t) \tag{35}$$

where $\tau_{RC}$ is the membrane time constant and $v(t)$ and $j(t)$, resembling the membrane voltage and input current, are now unitless. Generally $\tau_{RC}$ is in the range in the tens of miliseconds for biological neurons. We use $\tau_{RC} = 0.02$. The implication of this rewrite is that the neuron now spikes at $v(t) = v_{\text{th}} = 1$ before resetting to $v_{\text{rest}} = 0$ and entering the refractory period. Forcing the input current to be static by letting $j(t) = j$ and solving the




(a) Interleaved plot of original rate function, $r(j)$ (orange), and smoothed version, $r_\rho(j)$ (blue).

(b) Theoretical (orange) and simulated spike rates (blue) as functions of input current.

(c) Scaling of the rate function in the interval $[0; 10]$ to mimic ReLU.

Figure 2: Plots of smoothed, poisson coded and ReLU-like rate functions, respectively.

differential equation allows us to describe the membrane voltage over time with constant input current:

$$v(t) = (v_0 - j)e^{-t/\tau_{RC}} + j \tag{36}$$

where $v_0$ is the voltage at $t = 0$ which most often is set $v_0 = V_{\text{rest}}$. It is now possible to derive the spike rate which for input current when the input current exceed the threshold $j > V_{\text{th}}$ by setting $v(t) = V_{\text{th}}$ and solving for $r = t^{-1}$ to get.

$$r(j) = \begin{cases} \left[t_{\text{ref}} - \tau_{RC} \log\left(1 - \frac{V_{\text{th}}}{j}\right)\right]^{-1} & \text{if } j > V_{\text{th}} \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

Notice that $t_{\text{ref}}$ is added to the time-to-spike, to incorporate the effect of the refractory period. Using the rate function directly as activation function would lead to problems as its derivative goes to infinity when $j \to 0_+$. Hunsberger et al. therefore introduces a smoothed rate function using the softplus function $\rho(x) = \gamma \log(1 + \exp(x/\gamma))$ that works as a smooth max function and does in fact becomes equivilant when $\gamma \to 0$. The smooth rate function becomes:

$$r_\rho(j) = \left[t_{\text{ref}} - \tau_{RC} \log\left(1 + \frac{V_{\text{th}}}{\rho(j - V_{\text{th}})}\right)\right]^{-1} \tag{38}$$

A plot of the original and smoothed rate functions can be seen in figure 2a.

### 3.3.3 Poisson rate coding

As SNNs operate on spiking inputs, continuous input variables have to be converted to stochastic spiking variables. Such can be accomplished with Possion rate coding where the value of the continuous variable is interpreted as the mean spike rate of the Poission distribution with the following parameters:

$$s_i^{(t)} \sim \text{Pois}\left(\gamma = \frac{x_i}{t_{\text{ref}}} dt\right) \tag{39}$$

where $dt$ is the discrete timestep used for simulation. This formulation requires that input variables $x_i$ are normalized to values between 0 and 1 as division with $t_{\text{ref}}$ maps 1's to the maximum spike rate $r_{\max} = 1/t_{\text{ref}}$. Disadvantageously, poission rate coding causes the stochastic signal to contain a relativ large amount of noise in the signal as seen in figure 2b.

### 3.3.4 Low-pass alpha-filter

When modelling biologically realistic neural networks it is equally important to model the synapses as the neural dynamics. Synapses are known to act as low-pass filters on the transmitted spikes. Hunsberger and Eliasmith [16] proposes a simple exponential low-pass filter

$$\alpha(t) = \frac{t}{\tau_s} e^{-t/\tau_a} \tag{40}$$

where $\tau_s$ is some time-constant.

## 3.4 Shadow training spiking neural networks

The method of shadow training spiking neural networks amounts to the task of training a standard neural network using back-propagation, incorporating the rate-function of the chosen dynamical neuron model as activation function, whereby a spiking neural network can be constructed with the learned weight and bias parameters. However, doing so is in some cases not always quite as trivial. This section will go into the details of shadow training feedforward neural networks and the tricks for successfully doing so.

### 3.4.1 Activation scaling

When training neural networks using the analytical spiking rate as activation function, we found that down-scaling the activation function drastically improved the converge of the network. The reason for this result can be explained by the combination of the Kaiming He initialization of the weights which keeps the variance of the intermediate representations in the network close to 1 when used in conjunction with ReLU activation. By scaling the rate function with a factor of 0.05 the rate function and its derivative will more closely mimic that of the ReLU in the interval $[0; 10]$ as seen in figure 2c.

### 3.4.2 Rate estimation using alpha-filter

By using a low-pass alpha-filter as defined in equation 40 it is possible to adaptively estimate the underlaying firing rate of a sequence of spikes. This is useful as our rate function expects a current as input which is correlated with the instantatious pre-synaptic firing rate. We found that using an alpha filter with time-constant $\tau_s = 1.18 \cdot 10^{-3}$ most precisely determined the underlaying firing rate.

### 3.4.3 Fusing batch normalization

In the process of converting shadow trained ANNs to the SNNs, custom modules such as batch normalization layers has to be converted to their spiking counterparts. Fortunately, if used in between the linear layer and activation, it is possible to "fuse" batch normalization into the weight and bias paramters of the predecessing linear layer. By replacing $\mathbf{x}_i$ in equation 33 with the transformation from our linear layer $\mathbf{z}_i^{(l)} = \mathbf{W}^{(l)}\mathbf{a}_i^{(l-1)} + \mathbf{b}^{(l)}$

$$\mathbf{y}_i = \frac{\gamma}{\sqrt{\sigma_{\text{mov}}^2 + \epsilon}} \cdot \left( \mathbf{W}^{(l)}\mathbf{a}_i^{(l-1)} + \mathbf{b}^{(l)} \right) + \left( \beta - \frac{\gamma \mu_{\text{mov}}}{\sqrt{\sigma_{\text{mov}}^2 + \epsilon}} \right) \tag{41}$$

it is possible to derive the "fused" weight and bias, parameters as follows:

$$\overline{\mathbf{W}}^{(l)} = \text{diag}\left(\frac{\gamma}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}}\right)\mathbf{W}^{(l)}, \quad \overline{\mathbf{b}}^{(l)} = \text{diag}\left(\frac{\gamma}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}}\right)\mathbf{b}^{(l)} + \left(\beta - \frac{\gamma\mu_{\text{mov}}}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}}\right). \tag{42}$$

### 3.4.4 Testing spiking neural networks

It is crucial and foundational to machine learning when constructing models to test how well they generalize on new data that the model has not been exposed to before. When testing on a well-balanced catagorical dataset, one might simply want to measure the accuracy of the model. For neural networks trained with softmax activation in the last layer, the output of the model can be seen as probability distribution of the input belonging to a certain class. Although this interpretation is useful, it is enough to take the argmax of the logits (inputs to the softmax) when trying to predict the most probable class. For spiking neural networks there are different options for determining the prediction of the network. One might predict the class of the first neuron that spikes if the time to prediction is critical. However, this method might not yield the best result as noise in the input signal might lead a neuron, not representative of the true class, to spike first. To evaluate our spiking network, we predict the class of the neuron that had the most spikes in a period of 200ms, which is in the order of the average human response time.

## 3.5 Predictive Coding

Predictive coding is hierarchial theory of perception in the brain where higher level cortical areas "explain away" lower-level visual input and only discrepancies hereof are propagated up for further processing. The theory sparked great interest in the neuroscientific community when Rao and Ballard published their seminal 1999 paper [27] showing how hierarchial predictive coding models were able explain neurological phenomenon like endstopping and other extra-classical receptive field effects where the surroundings modulate the response of a receptive field. In the predictive coding theory, the brain is seen as a generative model, able to restore the visual input of the retina by modelling the three dimensional structure of the world. This could also be formulated as inferring the underlaying hidden state of the world, $s_i$, causing the sensations, $o_i$, by modelling $p(o_i|s_i)$. Hence, predictive coding share core ideas with the bayesian brain theory. Though predictive coding was originally only ment to explain the perception, it has been suggested as unifying theory of the brain [23]. This section will focus on the correspondance between a special instance of predictive coding dubbed zero inference-learning (Z-IL) where the minimization of error units is equivalent to back-propagation as shown by Song et al [30].

### 3.5.1 Predictive coding networks and inference learning

Predictive coding networks are very similar to feedforward neural networks but include error nodes, $\epsilon^{(l)}_{i,t}$, between value nodes (as illustrated in figure 3) that encode the difference between a value node, $x^{(l)}_{i,t}$, and its prediction, $u^{(l)}_{i,t}$:

$$\epsilon^{(l)}_t = \mathbf{x}^{(l)}_t - \mathbf{u}^{(l)}_t, \quad \mathbf{u}^{(l)}_t = \mathbf{W}^{(l)}\sigma^{(l)}(\mathbf{x}^{(l-1)}_t) + \mathbf{b}^{(l)} \tag{43}$$

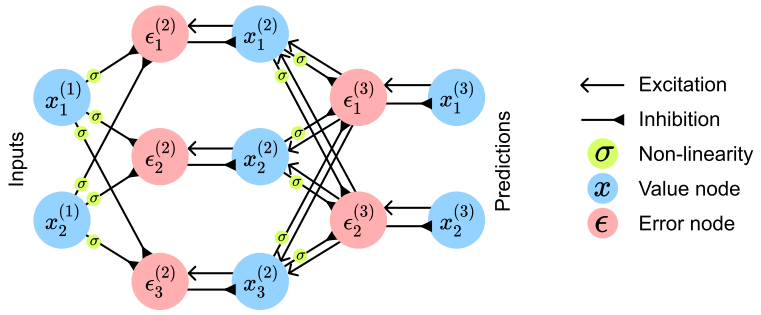



Figure 3: Predictive coding network with 2 input nodes, 3 hidden nodes and 2 output nodes.

Inference learning (IL) is a particular algorithm introduced by Song et al. [30] to learn the parameters of predictive coding networks for both supervised and un-supervised learning tasks. The algorithm runs in two phases; an inference phase and a learning phase. To train the network supervised, the input and target nodes are clamped to the input and target variables that the network should learn a mapping between. Then in the inference phase, value nodes are inferred such that they minimize the total free energy of the network:

$$F_t = \sum_{l=2}^{L} \frac{1}{2} \|\epsilon_t^{(l)}\|^2. \tag{44}$$

To minimize the error nodes, the value nodes are updating according to the following rules;

$$\dot{\mathbf{x}}_t^{(l)} = \begin{cases} 0 & \text{if } l = 1 \\ -\epsilon_t^{(l)} + \sigma^{(l)\prime}(\mathbf{x}_t^{(l)})(\epsilon_t^{(l+1)}\mathbf{W}^{(l+1)}) & \text{if } l = 2 \dots L-1 \\ -\epsilon_t^{(l)} & \text{if } l = L \text{ during prediction} \\ 0 & \text{if } l = L \text{ during learning} \end{cases} \tag{45}$$

using Eulers method such that $\mathbf{x}_{t+1}^{(l)} = \mathbf{x}_t^{(l)} + \gamma \dot{\mathbf{x}}_t^{(l)}$ where $\gamma$ is the integration step size. Once value nodes have converged to an equilibrium at time $t = T$, the parameters are updated to yet again minimize the free energy:

$$\Delta \mathbf{W}^{(l)} = \alpha \epsilon_T^{(l)} \sigma(\mathbf{x}_T^{(l-1)}), \quad \Delta \mathbf{b}^{(l)} = \alpha \epsilon_T^{(l)} \tag{46}$$

where $\alpha$ is the learning rate. A possible source of confusion when coming from feedforward neural networks is that predictive coding networks do not propagate activity directly from value node to value node. Instead, value nodes should be considered free variables that are updated according to equation 45. Interestingly, after the inference phase comes to an equilibrium where $\dot{\mathbf{x}}_T^{(l)} = 0$ and all error nodes become zero, the value nodes will correspond to the activity of the neurons in forward prediction of an classic ANN; $0 = \epsilon_T^{(l)} = \mathbf{x}_T^{(l)} - \mathbf{u}_T^{(l)} \Rightarrow \mathbf{x}_T^{(l)} = \mathbf{u}_T^{(l)}$, if only the input node is clamped. Even more intriguing is the fact that the error nodes will become approximately equal to the error terms in the back-propagation algorithm (equation 25) when both input and output nodes are clamped

$$\epsilon_T^{(L-1)} = \sigma'(\mathbf{x}_T^{(L-1)}) \left( \mathbf{x}_T^{(L)} - \mathbf{u}_T^{(L)} \right), \quad \epsilon_T^{(l)} = \sigma'(\mathbf{x}_T^{(l)}) \left( \epsilon_T^{(l+1)} \mathbf{W}^{(l+1)} \right). \tag{47}$$

Here, $\mathbf{x}_T^{(L)} - \mathbf{u}_T^{(L)}$ is the derivative of the squared loss $\mathcal{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$.

### 3.5.2 Z-IL and equivalence to back-propagation

Song et al. takes this a step further by showing that a special variant of inference learning, zero-divergence inference learning (Z-IL) gives rise to the exact same weight updates as the back-propagation algorithm [30]. They show this by first assuming that every value node, $x_{i,t}^{(l)}$ and prediction $u_{i,t}^{(l)}$ is equal to the corresponding forward activity of the equivalent ANN. This is naturally satisfied when the input node is clamped to the input value of some training pair and the network is relaxed to an equilibrium, corresponding to the forward propagation necessary in the back-propagation algorithm. The output node is then fixed to the output value of the training pair after which each value node is updated to minimize $F$ according to equation 45 with integration step size $\gamma = 1$. At last, the weights are scheduled to be updated according to equation 46 at time $t = L - l$.

### 3.5.3 Predictive coding networks for classification

Predictive coding networks by default predict continuous valued output variables and are therefore more suited for regression tasks than classification. However, it is possible to modify the network to perform classification by relying on the insight of deriving the partial derivative of the cross-entropy loss function in section 2.3. Inference learning updates value nodes in order to minimize the total free energy of the network according to:

$$\begin{aligned}\dot{\mathbf{x}}_t^{(l)} = -\frac{\partial F_t}{\partial \mathbf{x}_t^{(l)}} &= -\frac{\partial}{\partial \mathbf{x}_t^{(l)}}\left[\frac{1}{2}\|\epsilon_t^{(l)}\|^2 + \frac{1}{2}\|\epsilon_t^{(l+1)}\|^2\right] \\ &= -\epsilon_t^{(l)} - \frac{\partial \frac{1}{2}\|\epsilon_t^{(L)}\|^2}{\partial \mathbf{u}_t^{(l+1)}}\frac{\partial \mathbf{u}_t^{(l+1)}}{\partial \mathbf{x}_t^{(l)}} \\ &= -\epsilon_t^{(l)} + \sigma^{(l+1)\prime}(\mathbf{x}_t^{(l)})(\epsilon_t^{(l+1)}\mathbf{W}^{(l+1)}).\end{aligned} \tag{48}$$

as changing the value node $\mathbf{x}_t^{(l)}$ will both effect its own layers error nodes and error nodes in the next layer as seen in equation 43. In order for the predictive coding network to perform classification, we set the last weight to the identity matrix, activation function $\sigma^{(L)}(\cdot)$ to the softmax function and the derivative of the activation function to $\sigma^{(L)\prime}(\cdot) = 1$. We will now consider the second last layer, $L-1$, in the network, corresponding to the logits before being transformed by the softmax function. From the above, the value nodes in the second last layer will now change according to:

$$\dot{\mathbf{x}}_t^{(L-1)} = -\epsilon_t^{(L-1)} + \epsilon_t^{(L)}. \tag{49}$$

Here, we notice that changing $\mathbf{x}_t^{(L-1)}$ in the direction of $\epsilon_t^{(L)}$ will minimize the cross-entropy loss (as seen in equation 22), since:

$$\epsilon_t^{(L)} = -\frac{\partial H(\mathbf{x}_t^{(L)}, \mathbf{u}_t^{(L)})}{\partial \mathbf{x}_t^{(L-1)}} = \mathbf{x}_t^{(L)} - \mathbf{u}_t^{(L)} \tag{50}$$

Hence, the predictive coding network will learn to perform classification.

### 3.5.4 Batch normalization in predictive coding networks

It is possible to implement batch normalization in PCNs, although some minor modifications to the update equations are necessary. We implement a batch normalizing layer in our PCN, similarly as when introducing a softmax layer: To only allow for one-to-one connections, the weight matrix is set to the identity matrix. Bias is set to the zero vector and activation function to the batch normalization operation, such that the prediction of the next layer becomes:

$$\mathbf{u}_t^{(l)} = \mathbf{I}\sigma^{(l)}(\mathbf{x}_t^{(l-1)}) + \mathbf{0} = \mathrm{BN}_{\gamma^{(l)},\beta^{(l)}}(\mathbf{x}_t^{(l-1)}) = \mathbf{y}_t^{(l)}. \tag{51}$$

To derive the dynamics for value nodes and parameter updates of $\gamma^{(l)}$ and $\beta^{(\beta)}$ we will be taking our starting point in the gradient of the batch normalization operation as given in the original paper [18] included as appendix 6.1.
As shown in the previous section, inference learning changes the value nodes to minimize the total free energy by minimizing the error node local to the value node as well as the affected error nodes in the next layer. In layer $l$, just before a batch normalization layer, the value node will change according to:

$$\dot{\mathbf{x}}_{i,t}^{(l)} = -\epsilon_{i,t}^{(l)} - \frac{\partial \frac{1}{2}\|\epsilon_{i,t}^{(l+1)}\|^2}{\partial \mathbf{x}_{i,t}^{(l)}} \tag{52}$$

By letting the loss function be equal to the squared error unit $\ell = \frac{1}{2}\|\epsilon_{i,t}^{(l+1)}\|^2$, we see from the batch norm derivatives in the appendix, that the last term in the equation above can be substituted for $\frac{\partial \ell}{\partial \mathbf{x}_{i,t}^{(l)}}$ where $\frac{\partial \ell}{\partial \mathbf{y}_{i,t}^{(l+1)}} = \frac{\partial F_t}{\partial \mathbf{u}_{i,t}^{(l+1)}} = -\epsilon_{i,t}^{(l+1)}$ giving us the update rule:

$$\dot{\mathbf{x}}_{i,t}^{(l)} = -\epsilon_{i,t}^{(l)} - \frac{\partial \ell}{\partial \widehat{\mathbf{x}}_{i,t}^{(l)}} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^{2\,(l)} + \epsilon}} - \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^{2\,(l)}} \cdot \frac{2\left(\mathbf{x}_{i,t}^{(l)} - \mu_{\mathcal{B}}^{(l)}\right)}{m} - \frac{\partial \ell}{\partial \mu_{\mathcal{B}}^{(l)}} \cdot \frac{1}{m}. \tag{53}$$

The parameters are in the same fashion learned by minimizing free energy:

$$\Delta\gamma^{(l)} = -\alpha\frac{\partial \ell}{\partial \gamma^{(l)}} \quad \Delta\beta^{(l)} - \alpha\frac{\partial \ell}{\partial \beta^{(l)}} \tag{54}$$

# 4 Results and Discussion

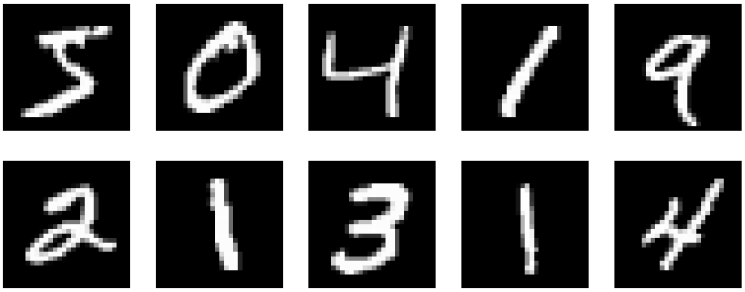


Figure 4: First 10 examples of the MNIST training dataset.

For all experiments we train classification models in a supervised manner to predict digits in the permutation invariant MNIST task (P-MNIST). Each model consists of a two hidden layer fully connected feed-forward network with 300 neuron in each hidden layer. For shadow training SNNs the Adam optimizer is used with a learning rate of 0.0001 while PCNs and its analogous FFNN use SGD with a learning rate of 0.01. All models were trained to minimize the cross-entropy loss for 20 epochs with a batch size of 64. Training was done on a 2021 MacBook Air with 8GB of RAM as not much compute was needed, but all code was written in a manner for easily taking advantage of GPU acceleration using the PyTorch framework.

## 4.1 Experiment 1 - Z-IL vs. Backprop

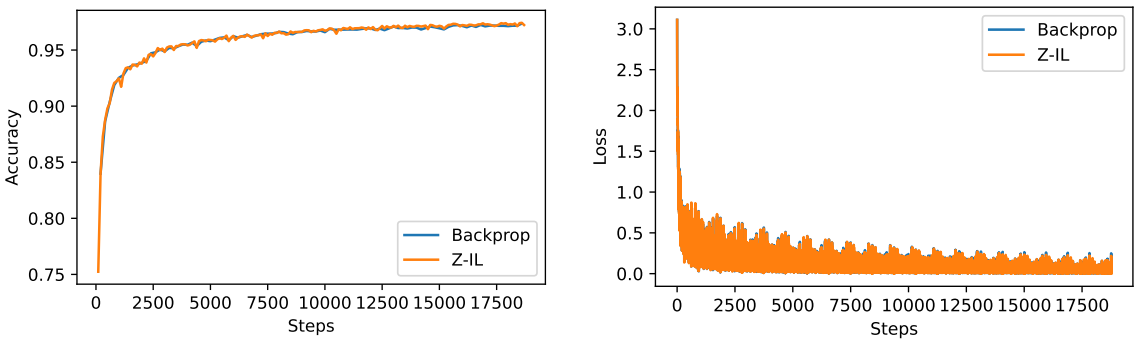



Figure 5: Test accuracy and cross-entropy loss as function of number of parameter update steps for backprop and Z-IL.

We train a PCN and its corresponding FFNN, both employing our scaled soft rate activation function, to emperically validate the equivalence of the weight updates. From figure 5 we see that Z-IL performs very well and gives rise to the same accuracy and loss curves as backprop with only minor deviations that might be caused by numerical precision in the specific implementation. Although the final accuracy of our PCN trained with Z-IL of 97.31% does not achieve state-of-the-art results, it might be possible to further improve the model with more recent methods such as convolutions, residual connections, batch normalization, etc. which can all be implemented in PCNs and trained with Z-IL as shown by Milladge et al. [24] and Salvatori et al. [29].

## 4.2 Experiment 2 - Shadow training spiking neural networks

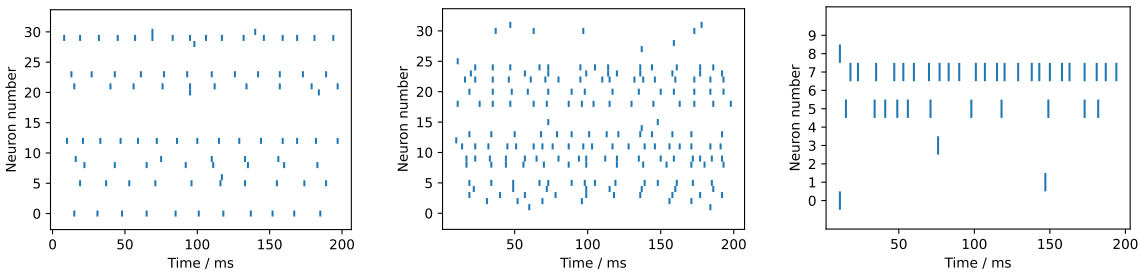



Figure 6: Neuron activity in each hidden and output layer of SNN presented with an image of the digit 7 for 200ms. For visualisation purposes each hidden layer only contains 32 neurons.

We shadow train a SNN using the soft rate function from equation 38, scaled according to section 3.4.1 with a factor of 0.05, as activation function and compare it against a feed-forward baseline model using ReLU activation. To make the input image mimic it is spiking, we scale each pixel value to the full range of firing rates $[0; t_{\text{ref}}^{-1}]$. Batch normalization was applied after each linear transform for both SNN and baseline models to improve training stability.

| **Algorithm** | **Optimizer** | **Activation** | **BN.** | **Spiking** | **Accuracy** |
|---|---|---|---|---|---|
| Backprop | ADAM | ReLU | Yes | No | 97.68% |
| Backprop | ADAM | Soft-rate | Yes | No | 97.97% |
| Backprop | ADAM | Soft-rate | Yes | Yes | 97.71% |
| Z-IL | SGD | Soft-rate | No | No | 96.30% |
| Z-IL | SGD | Soft-rate | No | Yes | 95.21% |
| Z-IL | ADAM | Soft-rate | No | No | 96.94% |
| Z-IL | ADAM | Soft-rate | No | Yes | 96.32% |
| Z-IL | ADAM | Soft-rate | Yes | No | 97.35% |
| Z-IL | ADAM | Soft-rate | Yes | Yes | 96.29% |

Table 1: Test accuracies on the P-MNIST benchmark.

From figure 1 we see that our rate-coded SNN performs on-par with our baseline model with a slightly higher accuracy. Whether this small improvement is caused by the statistical variation in the initialization of the weights or due to the smooth rate function, unlike ReLU, having non-zero gradient for values less than zero, requires more careful experimentation with precise weight initialization and larger sample sizes. Interestingly, only very little performance is lost when the learned weights in the rate-coded model is fused with batch normalization and used for simulation of the LIF dynamics. This might be due to the noise introduced in the Possion coding process along with a limited simulation time of 200ms before comparing the most spiking neuron with the correct label. Using batch normalization is known to also have a regularizing effect [18] which might explain this robustness, however further experimentation is needed to confirm this.

## 4.3 Evaluation of Biological Plausibility

We will now consider the biological plausability of the combination of predictive coding networks and spiking networks in relation to the problems of back-propagation and ANNs in section 3.2, beginning with the three constraints derived from the biological neuron. As all value nodes and weights are updated according to local learning rules, we consider the the first constraint of locality satisfied. The constraint of spikes requires that inter-neuron communication happens via spikes. As this is only the case after the shadow network is converted to its spiking form and not in the learning phase, we only consider this partially satisfied. At last, we do not consider the constraint of unidirectionality satisfied. Although the parameter updates of IL does not directly use symmetric weights, error and value nodes used for the update require the backwards flow of error information from higher-level layers. Predictive coding does therefore not solve the weight transport problem in its presented form. As PCNs are very closely related to ANNs, incorporating methods suchs as DFA/FA might be possible without limiting the learning capabilities of the network. Inference learning does also not seem to solve the derivative problem of back-propagation, as the derivative of the activation function is used to update the value nodes. However, by considering our spike rate as activation function, its derivative might be approximated as the change in duration between spikes. This interpretation also solves the spiking problem, as numerical values are communicated by the use of rate-coding. In their paper [30], Song et al. introduces a further modification to the Z-IL algorithm, Full autonomy Z-IL, that detects when weight layers should be updated. Although this solves the problem of the network depending on external control a more biologically plausible method might be continually updating the weights as it is the case in Continual Equilibrium Propagation [6]. Throughout this thesis, we have only considered the task of supervised leaning for simpliticy's sake. Although this provides a useful way to measure how well the networks perform, it is unlikely to be happening in the brain, as it would require the brain to have access to the exact labels it is trying to learn. A more likely scenario is probably that of self-supervised learning discussed in the introduction. The problem then becomes to define what objective function the network should minimize. In the case of predicting future stimuli from past experience, it might not be enough to simply minimize the prediction error. Doing this for video frames using a squared loss between pixels would lead to blurry predictions due to the uncertainty of the future. An alternative could be that the objective function is itself learned such as the discriminator in Generative Adverserial Networks [9].

## 4.4 Benefits to biological plausible learning

Today, it is often necessary to prune and quantize large deep learning models when deploying on low-powered mobile hardware. Special hardware such as as Apple's Neural Engine and Google's mobile TPUs have been designed to accelerate inference throughput while maintaining a low power usage. Although biological plausible deep learning as presented in this thesis does not give rise to more efficient learning algorithms for current available hardware, they might provide the foundational learning mechanisms for upcoming neuromorphic hardware.

# 5 Conclusion

We have in this thesis introduced, implemented and tested biological plausible alternatives to current deep learning methods. This concludes preditive coding networks as alternative to feed-forward neural networks trained with inference learning as alternative to the back-propagation algorithm. At last, we have introduced the concept of shadow training where the rate function of the Leaky-Integrate-and-Fire neuron model is used to learn the connectivity parameters of spiking neural networks. From our review of biological plausbility we found that although PCNs and shadow training SNNs both satisfy different biological constraints the intersection of the two does not yield a fully biological plausible model of learning. More work would have to be done to implement predictive coding with all communication encoded as spikes as well as getting a better understanding of what contributes to forming neural circuits. From our experiments, we found that Z-IL performed very similar to back-propagation and was able to shadow train SNNs with only minor drop in performance when simulating with spiking dynamics. We therefore conclude that the proposed alternative methods are competitative with deep learning methods in achieving good performance on the MNIST benchmark dataset. However, as vanialla inference learning requires several inner interation for the network to come to an equilibirum, computationally, back-propagation is far superior on current hardware. We argue that the introduction of neuromorphic hardware might change this fact and provide faster and more efficient learning.

## 5.1 Further research

What should be done in the future?

# References


[1] Yoshua Bengio, Dong-Hyun Lee, Jörg Bornschein, and Zhouhan Lin. Towards biologically plausible deep learning. *CoRR*, abs/1502.04156, 2015.

[2] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472, 1998.

[3] Sander M. Bohté, Joost N. Kok, and Han La Poutré. Spikeprop: backpropagation for networks of spiking neurons. In *ESANN*, 2000.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[5] Maxence Ernoult, Julie Grollier, Damien Querlioz, Yoshua Bengio, and Benjamin Scellier. Equilibrium propagation with continual weight updates. *CoRR*, abs/2005.04168, 2020.

[6] Maxence Ernoult, Julie Grollier, Damien Querlioz, Yoshua Bengio, and Benjamin Scellier. Equilibrium propagation with continual weight updates. *CoRR*, abs/2005.04168, 2020.

[7] Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*, 2021.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[12] Donald O. Hebb. *The Organization of Behaviour - A neuropsychological theory*. John Wiley & Sons, Inc., 1949.

[13] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning: Lecture 6a overview of mini-batch gradient descent. Online, 2012.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[15] A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, 117:500–544, 1952.

[16] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. 2015.

[17] Hunsberger, Eric. *Spiking Deep Neural Networks: Engineered and Biological Approaches to Object Recognition*. PhD thesis, University of Waterloo, 2018.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[19] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random feedback weights support learning in deep neural networks, 2014.

[22] William Lotter, Gabriel Kreiman, and David D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *CoRR*, abs/1605.08104, 2016.

[23] Beren Millidge, Anil Seth, and Christopher L Buckley. Predictive coding: a theoretical and experimental review, 2021.

[24] Beren Millidge, Alexander Tschantz, and Christopher L. Buckley. Predictive coding approximates backprop along arbitrary computation graphs. *CoRR*, abs/2006.04182, 2020.

[25] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks, 2016.

[26] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[27] Rajesh Rao and Dana Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2:79–87, 02 1999.

[28] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[29] Tommaso Salvatori, Yuhang Song, Thomas Lukasiewicz, Rafal Bogacz, and Zhenghua Xu. Reverse differentiation via predictive coding, 2021.

[30] Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Can the brain do backpropagation? — exact implementation of backpropagation in predictive coding networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22566–22579. Curran Associates, Inc., 2020.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[32] James Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Computation*, 29:1–34, 03 2017.

# 6 Appendix

## 6.1 Batch normalization gradients

Derivatives of the batch normalization operation as presented in the original paper [18]:

$$
\begin{aligned}
\frac{\partial \ell}{\partial \widehat{x}_i} &= \frac{\partial \ell}{\partial y_i} \cdot \gamma \\
\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-3/2} \\
\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} &= \left( \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2 \left(x_i - \mu_{\mathcal{B}}\right)}{m} \\
\frac{\partial \ell}{\partial x_i} &= \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2 \left(x_i - \mu_{\mathcal{B}}\right)}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m} \\
\frac{\partial \ell}{\partial \gamma} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i \\
\frac{\partial \ell}{\partial \beta} &= \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}
\end{aligned}
\tag{55}
$$