# Predictive Coding and Biologically Plausible Neural Networks

Bachelor Project

Anders Bredgaard Thuesen
s183926@student.dtu.dk

January 2022

## Abstract

# Contents

# 1 Introduction

## 1.1 Motivation

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 biological inspiration of the neuron to the perceptron where data inputs are weighted, summed together and thresholded, several new modern architectures, like recurrent, residual and transformer neural networks have pushed the limits and achieved state of the art results in speech recognition, computer vision and natural language understanding. Despite of these networks being originally inspired by the brain, the backpropagation (backprop) algorithm for learning the weights and deep learning in general has been criticized for being biologically implausible [1]. This project will primarily be dealing with on of them: The weight transport problem which arises from the way backprop uses the connection weights in both the forward pass (inference) and the backwards parse (calculating the gradients), requiring that both forward and backward connections have symmetric weights and that information is able to flow backwards through the weights.

Besides having both philosophical as well scientific interest, studying the computational aspects of how the human brain processes sensory input might lead to great improvements in deep learning and artificial intelligence.

## 1.2 Related work

Several attempts has been made to make modern deep learning more biologically plausible. These can be divided into two types of categories. The first category consists of methods that try to optimize the inference on low-powered neuromorphic hardware such as the Intel Loihi or IBM TrueNorth chips, by converting existing neural network architectures into their spiking counterparts. The other category consists of methods that aim to make the learning phase biologically plausible by only relying on local weight updates in order to optimize for some objective. This is in alignment with the Hebbian learning theory from the neuroscience litterature which states that the synaptic plasticity is only dependent on the pre- and post-synaptic activity, possibly modulated through some global signaling mechasnism (ex. dopamine). One example hereof is the work by Bengio et al. on Continual Equilibrium Propagation [2].

## 1.3 Research questions

The project will address the following three research questions:

- According to the current literature, what are the biological constraints of biological learning?
- To what extent can predictive coding be used to approximate backpropagation under the above mentioned biological constraints?
- In what ways can modern deep learning benefit from biological plausible learning algorithms?

# 2 Classical deep learning

## 2.1 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 [5]. 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.

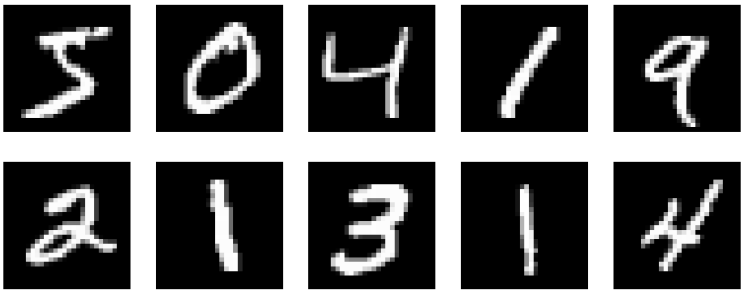

Figure 1: First 10 examples of the MNIST training dataset.

## 2.2 Feedforward neural networks

Feedforward 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. Feedforward 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 correspondance incentivises 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 feedforward neural network with $L - 2$ hidden layers is parametarized 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 feedforward 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 $\hat{\mathbf{y}}_i = \mathbf{a}^{(L)}$.

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. [3]

## 2.3 Back-propagation

The working horse of allmost all modern deep learning models is the back-propagation (aka. backprop) algorithm first popularized for training neural networks by Rumelhart, Hinton & Williams in 1986 [7]. 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 [6], 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

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

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:

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

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

$$\delta^{(L)} = \mathcal{L}'(\hat{\mathbf{y}}_i)\sigma'(\mathbf{z}^{(L)}), \quad \delta^{(l)} = \delta^{(l+1)}\mathbf{W}^{(l+1)}\sigma'(\mathbf{z}^{(l)}) \tag{3}$$

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

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \alpha\delta^{(l)}\mathbf{a}^{(l-1)}. \tag{4}$$

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. In theory the weights should be updated according to all datapoints in the dataset to maximize the likelihood, however in practice stochastic gradient descent is used with mini-batches of datapoints that stochastically resembles the overall distribution of the dataset.

# 3 Biologically plausible deep learning

The follow sections will describe alternative, more biologically plausible methods for training and evaluating artificial neural networks. It is the goal of these methods to satisfy the biological constraints put forward in section 3.2. These methods will primarily deal with the **the spiking problem** and **the weight transport problem**.

## 3.1 Biological neurons

## 3.2 Biological constraints

In their 2016 paper, Bengio et al [1]. raises 6 problems regarding the biologically plausibility of back-propagation which are nicely summarized and expanded upon in the PhD thesis by Hunsberger [4] 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 (2.3, 3.1) the problems have been reformulated as follows.

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 3. A well established fact of biological neurons is that of spikes travilling unidirectionally 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 3. 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 4 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.

### 3.2.1 Spiking neural networks

Spiking neural networks (SNNs) are a perticular type of network that uses spikes for interneuron communication. In that way they more closeley mimic biological neural networks, but share their structure and dense connectivity with feedforward 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 perticular 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. 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.

### 3.2.2 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:

$$C\frac{dV}{dt} = J(t) - \frac{V}{R} \tag{5}$$

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 $\tau_{\text{ref}} = 0.004$. If the applied input

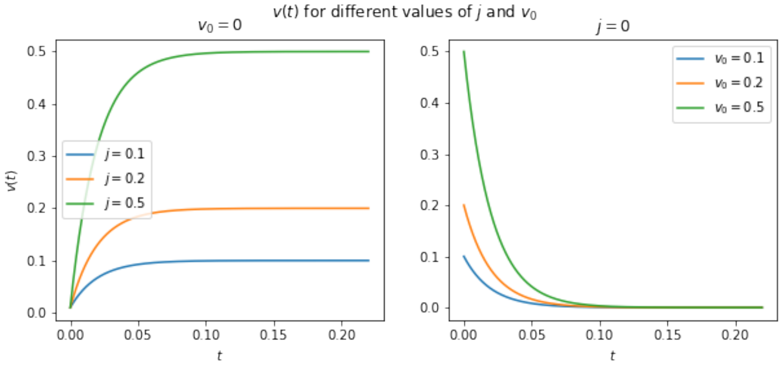

Figure 2: Voltage of LIF neuron over time for different values of $j$ and $v_0$.

current is not large enough to make the neuron spike, the membrane voltage will naturally decay to $V_{\text{rest}}$. Hunsberger [4] shows how the above can be rewritten in simpler terms as:

$$\tau_{RC}\frac{dv(t)}{dt} = j(t) - v(t) \tag{6}$$

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

$$v(t) = (v_0 - j)e^{-t/\tau_{RC}} + j \tag{7}$$

where $v_0$ is the voltage at $t = 0$ which most often is set $v_0 = V_{\text{rest}}$. Plotting $v(t)$ with different values of $j$ and $v_0$ can be seen in figure 2. It is possible to derive the spike rate when $j > V_{\text{th}}$ by setting $v(t) = V_{\text{th}}$ and solving for $r = t^{-1}$ to get.

$$r = \left[\tau_{\text{ref}} - \tau_{RC}\log\left(1 - \frac{V_{\text{th}}}{j}\right)\right]^{-1} \tag{8}$$

Notice that $\tau_{\text{ref}}$ is added to the time-to-spike, to incorporate the effect of the refractory period.

It is possible to use the LIF model to construct a SNN whose weights can be learned through ordinary back-propagation after which it may be run by either simulating the LIF dynamics or tranferring the network to neuromorphic hardware for more efficiency.

### 3.2.3 Poisson rate coding

In order to input non-spiking variables into our SNN they should first be converted into spikes. This can be accomplished by poission rate coding. The generated spikes are sampled from a poission distribution $X_i \sim \text{Pois}(x_i)$.

### 3.2.4 ANN to SNN conversion

How do we convert an ANN to a SNN and run the stuff live?

## 3.3 Predictive Coding and Z-IL

What is predictive coding? Mention Rao and Ballard. How can they calculate gradients?

### 3.3.1 Equivalence to back-prop

### 3.3.2 Variational free energy

## 3.4 Variational Inference

## 3.5 Energy Based Models

# 4 Results

# 5 Discussion

# 6 Conclusion

# References

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

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

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

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

[5] 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.

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

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