Biologically Realistic Artificial Neural Networks

Matthijs Kok
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
m.r.kok@student.utwente.nl

ABSTRACT

Artificial Neural Networks (ANNs) play an important role in machine learning nowadays. A typical ANN is traditionally trained using stochastic gradient descent (SGD) with backpropagation (BP). However, it is unlikely that a real biological neural network is trained similarly. Neuroscience theories, such as Hebbian Theory could inspire adaption from the traditional training method SGD to make ANNs more biologically plausible. Two mathematical descriptions of Hebbian theory will be suggested based on the limitations of the mathematical framework of Hebbian theory: A competitive Hebbian learning rule and an imply Hebbian learning rule. Finally, this research will propose a method, called the Hebbian Plasticity Term (HPT) method, that incorporates the mathematical description of Hebbian theory to modify the traditional training method SGD. Therefore two variants of the HPT-method are proposed: HPT-Competitive and HPT-Imply. The influence of the hebbian plasticity term φ on SGD shows more biologically realistic plasticity of a synaptic connection between neurons in the ANN at the cost of performance.

Keywords

Artificial Neural Network, Stochastic Gradient Descent with Backpropagation, Hebbian Theory, Hebbian Plasticity Term method

1. INTRODUCTION

Deep learning is a very important area in the study of machine learning nowadays, with wide applications [21, 18, 10] such as speech recognition [9, 15], natural language processing [7], and image recognition [11, 32]. It describes a powerful set of techniques for learning in artificial neural networks (ANNs) [4, 25, 31]. A traditional method to train ANNs is stochastic gradient descent (SGD), which makes use of backpropagation (BP) [13, 25].

In the biological sciences, understanding the biological basis of learning is considered to be one of the ultimate challenges [17]. Theoretical or computational neuroscientists strive to make a link between observed biological processes and biologically plausible theoretical mechanisms (computational models) for neural learning, to gain an understanding of how biological systems, such as the brain, work. Furthermore, neuroscience also aims to make contributions towards progress in artificial intelligence, such as inspiration and validation for new computer algorithms and architectures [5]. To improve the understanding in both fields, models that better resemble biological neural networks are of utmost importance.

ANNs are vaguely inspired by biological neural networks. However, the method SGD-BP that is used to train ANNs does not resemble the way in which the human brain learns. This research aims to contribute to closing the gap between artificial and biological neural networks, as it will investigate a neuroscience theory called Hebbian theory [16] as starting point to modify the traditional training method SGD of ANNs aiming to make the learning process more biologically plausible. Within this research, two mathematical formulations are proposed for different Hebbian learning rules, which are incorporated in traditional SGD by a newly developed method. The results of this research can be used as a basis for developing more biologically realistic artificial neural networks.

This paper will investigate the following research questions:

RQ 1 How could Hebbian theory be used to mathematically describe a biologically realistic evolution of a synapse?

> **RQ 1.1** What are the limitations to Hebbian theory when using it to formulate a mathematical description of a weight update?
>
> **RQ 1.2** Which types of mathematical descriptions of Hebbian theory could be used, as a result of these limitations?

RQ 2 Could the training algorithm BP-SGD of a typical standard ANN be modified using Hebbian theory to become a more biologically plausible training method?

> **RQ 2.1** How could a mathematical description of Hebbian theory be incorporated into the standard training algorithm BP-SGD?
>
> **RQ 2.2** Does a connection weight between two neurons of an artificial neural network evolve more biologically plausible as a result of incorporating the new method?

RQ 3 To what extent does the performance of an ANN trained using the newly developed more biologically realistic training method differ from the performance of an ANN trained using the traditional training method BP-SG?

33^{th} Twente Student Conference on IT July 3^{rd}, 2020, Enschede, The Netherlands.

2. ANALOGY BETWEEN THE ANN AND THE BRAIN

2.1 Neurons in the brain

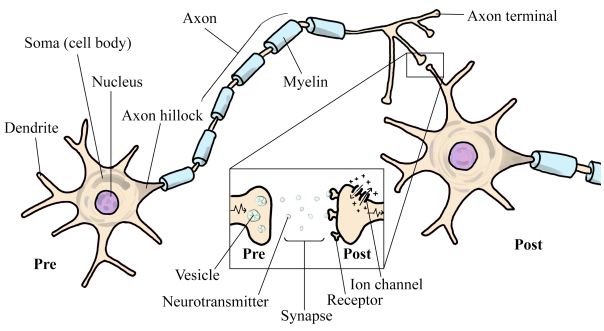


Figure 1. Schematic image of two neurons in a brain

Figure 1 depicts two neurons in a brain that are separated by a synapse. When the leftmost neuron, the pre-synaptic neuron, is excited by an action potential, the vesicles in its axon terminals fuse with its membrane and release their contents, neurotransmitters, into the synaptic cleft. Once released, the neurotransmitters can bind to the receptors on the dendrite of the rightmost neuron, the post-synaptic neuron, acting as a chemical signal. That binding opens ion channels on its dendrite that allow charged ions to flow in and out of the cell, converting the chemical signal into an electrical signal. If the combined effect of multiple dendrites (which can be connected to other neurons) of the post-synaptic neuron changes the overall charge of the cell enough, then it triggers an action potential, also called a spike [24, 27].

2.2 The relation between an ANN and neurons in the brain

A typical feedforward ANN could be described as a structured collection of nodes with edges between them, as depicted in Figure 2. The ANN that is depicted in Figure 2 represents connected neurons in a brain, as connected nodes. The spike in a neuron is considered a binary phenomenon: it is there or not. Therefore, the activation in an ANN represents the firing rate of the neuron. The more often a pre-synaptic neuron fires, the higher the chance that the post-synaptic neuron will also increase its firing rate. The activation a_j^l (of a node) in an ANN of a post-synaptic neuron is dependent on the activation a_i^{l-1} of the pre-synaptic neurons, according to equation (1).

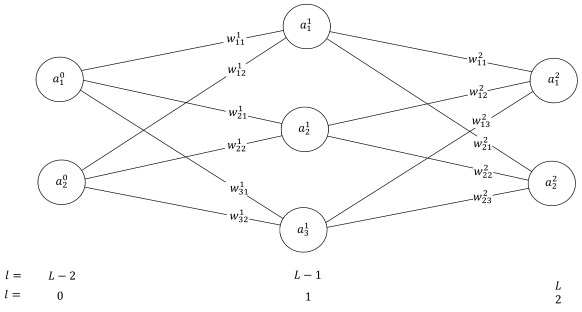


Figure 2. Example feedforward ANN with 2 neurons in the input layer, 3 neurons in the hidden layer, and 2 neurons in the output layer

$$a_j^l = \sigma\left[\sum_i (a_i^{l-1} \cdot w_{ji}^l) + b_j^l\right] \tag{1}$$

Here, σ denotes a sigmoid-activation function and the bias is unique to a neuron and is denoted as b_j^l. The weighted input sum $\sum_i (a_i^{l-1} \cdot w_{ji}^l) + b_j^l$ will be summarized as z_j^l throughout the rest of this paper. Furthermore, the superscript is written as l to denote *any* layer, as integer to denote a *specific* layer, or as $L, L-1, L-2, ..,$ where L denotes the *last* layer. Both, i and j denote the position of the neuron within a layer. The use of subscript letters will be alphabetically coherent with the order of layers (e.g. $..., a_h^{l-2}, a_i^{l-1}, a_j^l, a_k^{l+1}, ...$). The dependence of the post-synaptic neuron a_j^l on multiple pre-synaptic neurons a_i^{l-1} is also present in real biological post-synaptic neurons, which can be connected to multiple (pre-synaptic) neurons through their (multiple) dendrites, which determine together whether or not an action potential is triggered.

An ANN shows connections as edges between nodes. The edge or connection is analogous to the synapse between two neurons in a brain. In an ANN, a weight w_{ji}^l is associated to a connection and is considered to represent the strength of the respective connection. In other words, a large weight in an ANN implies a large synaptic efficacy of the analogous biological synapse. The subscript ji denotes the position of the synapse within a layer. Here, j denotes the post- and i the pre-synaptic neuron, which together form the synaptic connection w_{ji}^l.

3. TRAINING AN ANN

A common ANN training method is stochastic gradient descent (SGD), which requires backpropagation (BP). This section discusses how traditional SGD is performed on an ANN, to serve as a basis for section 6 and 7.

In an ANN, one feedforward is performed by computing equation (1) sequentially for each layer. The weights w_{ji}^l and biases b_j^l used to compute equation (1) are initialized as random variables drawn from a standard normal distribution. Since the computation of each separate component is cumbersome, vector-wise multiplication is used. Below, matrix multiplication for a feed-forward is shown (in this example layer l has 2 units and layer $l+1$ has 4 units):

$$\begin{bmatrix} z_1^{l+1} \\ z_2^{l+1} \\ z_3^{l+1} \\ z_4^{l+1} \end{bmatrix} = \begin{bmatrix} w_{11}^{l+1} & w_{12}^{l+1} \\ w_{21}^{l+1} & w_{22}^{l+1} \\ w_{31}^{l+1} & w_{32}^{l+1} \\ w_{41}^{l+1} & w_{42}^{l+1} \end{bmatrix} \cdot \begin{bmatrix} a_1^l \\ a_2^l \end{bmatrix} + \begin{bmatrix} b_1^{l+1} \\ b_2^{l+1} \\ b_3^{l+1} \\ b_4^{l+1} \end{bmatrix} \tag{2}$$

where

$$\begin{bmatrix} a_1^{l+1} \\ a_2^{l+1} \\ a_3^{l+1} \\ a_4^{l+1} \end{bmatrix} = \sigma\left(\begin{bmatrix} z_1^{l+1} \\ z_2^{l+1} \\ z_3^{l+1} \\ z_4^{l+1} \end{bmatrix}\right) \tag{3}$$

In short, this can be notated as:

$$\mathbf{z^{l+1}} = W^{l+1} \cdot \mathbf{a^l} + \mathbf{b^{l+1}} \tag{4}$$

and

$$\mathbf{a^{l+1}} = \sigma(\mathbf{z^{l+1}}) \tag{5}$$

Here, W^l is a matrix and $\mathbf{a^l}$, $\mathbf{b^l}$, $\mathbf{z^l}$ are vectors.

The prediction or classification of a digit by the network is determined to be the highest activation a_j^L in the output-layer. To be able to improve the performance, the cost

function in equation (6) is calculated.

$$C = \frac{1}{2} \cdot \sum_{j}^{n} (a_j^L - y_j)^2 \tag{6}$$

Here, y_j is a component of the vector of desired output activations $\mathbf{y}$, also called the label. $y_j = 1$ when the digit corresponding to output-neuron j is displayed on the image and $y_j = 0$ otherwise. The value of the loss function (6) is large for a big difference and small for a small difference between the output activation $\mathbf{a^L}$ and the label $\mathbf{y}$.

To be able to train the network, the weights and biases are modified to minimize the cost function, which is equivalent to maximizing the performance. Since the cost function is complex and multi-variate, it is very difficult to calculate the global minimum. Therefore, a (possibly local) minimum is approached in steps by adjusting all weights and biases in the direction of the negative gradient of the cost function. This method is called gradient descent [8, 25]. The gradient ∇C of a function is the direction of steepest ascent at a point P. A point P means a value for each of the variables w_{ji}^l and b_j^l. ∇C is a (column-)vector whose components are the partial derivatives of the cost function with respect to all weights and biases in the network, shown in equation (7).

$$\nabla C(w, b) = \begin{bmatrix} \frac{\partial C}{\partial w_{11}^1} & \frac{\partial C}{\partial b_1^1} & \cdots & \frac{\partial C}{\partial w_{ji}^L} & \frac{\partial C}{\partial b_j^L} \end{bmatrix}^T \tag{7}$$

Each step towards a minimum of the cost function, the value of every weight and bias in the network is adjusted according to equation (8) and (9):

$$w_{ji}^l \leftarrow w_{ji}^l - \eta \cdot \frac{\partial C}{\partial w_{ji}^l} \tag{8}$$

$$b_j^l \leftarrow b_j^l - \eta \cdot \frac{\partial C}{\partial b_j^l} \tag{9}$$

Here, the step size or weight update $\Delta w_{ji}^l = \eta \cdot \frac{\partial C}{\partial w_{ji}^l}$ is determined by the partial derivative of the cost function with respect to that particular weight $\frac{\partial C}{\partial w_{ji}^l}$ and the learning rate η. The partial derivative $\frac{\partial C}{\partial w_{ji}^l}$ is the corresponding component of the gradient of the cost function ∇C. The learning rate η can be chosen to control the magnitude of the step size.

To be able to compute ∇C, every component $\frac{\partial C}{\partial w_{ji}^l}$ an $\frac{\partial C}{\partial b_j^l}$ of ∇C must be computed. A component can be computed, using the Chain rule [35], according to equation (10) and equation (11).

$$\frac{\partial C}{\partial w_{ji}^l} = \frac{\partial z_j^l}{\partial w_{ji}^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot \frac{\partial C}{\partial a_j^l} = a_i^{l-1} \cdot \sigma'(z_j^l) \cdot \frac{\partial C}{\partial a_j^l} \tag{10}$$

and the partial derivative with respect to a bias in layer l of the network becomes:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial z_j^l}{\partial b_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot \frac{\partial C}{\partial a_j^l} = 1 \cdot \sigma'(z_j^l) \cdot \frac{\partial C}{\partial a_j^l} \tag{11}$$

where both in equation (10) and (11) holds that:

$$\frac{\partial C}{\partial a_j^l} = \sum_j \frac{\partial z_k^{l+1}}{\partial a_j^l} \cdot \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} \cdot \frac{\partial C}{\partial a_k^{l+1}} = \sum_j w_{kj}^{l+1} \cdot \sigma'(z_k^{l+1}) \cdot \frac{\partial C}{\partial a_k^{l+1}} \tag{12}$$

Also, note that $\frac{\partial z_j^l}{\partial w_{ji}^l} = a_i^{l-1}$, $\frac{\partial z_j^l}{\partial b_j^l} = 1$, $\frac{\partial a_j^l}{\partial z_j^l} = \sigma'(z_j^l)$, and of the last layer $\frac{\partial C}{\partial a_j^L} = a_j^L - y_j$.

To see how these chain rule expressions (10) and (11) are determined, the dependence diagram [1] of Figure 3 is useful.

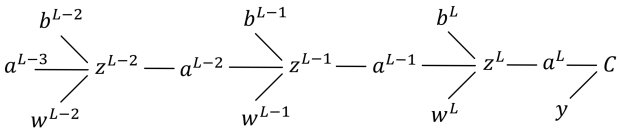


Figure 3. Dependence in the ANN

While propagating backwards through the network, terms of partial derivatives get longer in a consistent way. Generally, downstream (closer to the output layer) weight updates have shorter terms, whereas upstream (closer to the input layer) weight updates have longer terms. When backpropagating, the partial derivatives of the cost function with respect to each weight and bias are calculated for each layer as a vector, based on the partial derivatives of more downstream vectors as follows:

$$\frac{\partial C}{\partial b_i^{L-1}} = 1 \cdot \sigma'(z_i^{L-1}) \cdot \sum_j (w_{ji}^L \cdot \frac{\partial C}{\partial b_j^L}) \tag{13}$$

In (per-layer) vectors, this can be translated into equation 14:

$$\frac{\partial \mathbf{C}}{\partial \mathbf{b^{l-1}}} = \sigma'(\mathbf{z^{l-1}}) \odot (W^l)^T \cdot \frac{\partial \mathbf{C}}{\partial \mathbf{b^l}} \tag{14}$$

Here, the Hadamard product, or element-wise product, is denoted as $\odot$.

The partial derivative with respect to the weight can then be computed based upon the following relation:

$$\frac{\partial \mathbf{C}}{\partial \mathbf{w^l}} = \frac{\partial \mathbf{C}}{\partial \mathbf{b^l}} \cdot (\mathbf{a^{l-1}})^T \tag{15}$$

3.1 Mini-Batch SGD

A good middle ground between the stochastic gradient descent algorithm (which picks one training example and updates immediately after) and the standard gradient descent algorithm (which updates only every epoch after calculating the average gradient over all training examples) is a form of stochastic gradient descent that is called 'mini-batch' stochastic gradient descent [28, 25]. In mini-batch stochastic gradient descent, you pick a random set or 'batch' of training examples. Subsequently, you compute the average gradient over that batch of training examples and update using that gradient. Now, you have a nice combination of converging fast to a minimum while also keeping a smooth line if you would plot the cost function over time. The reason for this is that you update more often (after calculating the average gradient of the batch) and you average over a few training examples (those of the batch), respectively. Also, it is computationally less heavy, because you only update all weights and biases after computing the average gradient over the training examples in the batch.

4. NEUROSCIENCE THEORIES

4.1 Hebbian Theory

The Hebbian theory, or Hebb's rule, is a neuroscientific theory that attempts to explain synaptic plasticity [16]. The theory claims that if the pre-synaptic neuron repeatedly fires and stimulates a post-synaptic neuron, some growth process or metabolic change takes place in one or both cells that increases the efficiency of the pre-synaptic

neuron in firing the post-synaptic neuron. The theory is often summarized as "Cells that fire together, wire together" [22]. However, Hebb implicitly describes that the pre-synaptic cell should fire before, and not at the same time as, the post-synaptic cell.

This firing causation mentioned by Hebb is described in what is currently known as spike-timing-dependent plasticity (STDP), where temporal precedence is required [6]. This is a biological process that adjusts the strength of connections between neurons in the brain based on the relative timing of a neuron's output and input action potentials. Resulting in pre-synaptic neurons causing the post-synaptic neuron's excitation to be more likely to contribute in the future, whereas inputs that are not the cause of the post-synaptic spikes are made less likely to contribute in the future. Eventually, a subset of connections between the pre- and post-synaptic neurons remain.

According to equation (10), the *"fire together, wire together"* postulate is already partly implemented in a traditional ANN. This can be summarized into two points [25]. First, the weight of a connection between two neurons w_{ji}^l in an ANN will learn fast if the input neuron is high-activation ($a_i^{l-1} \approx 1$) and slowly if it is low-activation ($a_i^{l-1} \approx 0$), because the magnitude of a weight update $\frac{\partial C}{\partial w_{ji}^l}$ depends on the magnitude of the activation of the input neuron a_i^{l-1}. Second, if the output neuron has already "learned", increasing the weights to that output neuron is not very useful anymore, because the neuron already reached its target value (e.g. $a_j^l \approx 1$ or $a_j^l \approx 0$). It appears the weight of a connection between two neurons will learn slowly if the output neuron has saturated (i.e. is either high- or low-activation), because then $\sigma'(z_j^l) \approx 0$ and the weight update $\frac{\partial C}{\partial w_{ji}^l}$ depends on the size of $\sigma'(z_j^l)$.

4.2 BCM Theory

The BCM theory, or BCM synaptic modification, is a physical theory of learning in the visual cortex [3]. Part of the development of the BCM theory is based on limits of the Hebbian theory. Hence, Hebbian theory was normalized by introducing a loss in connection strength when there is no or unsynchronized activity, allowing for a decay of synapses.

The BCM model states that synaptic plasticity is stabilized by a dynamic adaptation of the time-averaged post-synaptic activity. According to the BCM model, a post-synaptic neuron will tend to undergo long-term potentiation (LTP) if it is in a high activity state when a pre-synaptic neuron fires or long-term depression (LTD) if it is in a lower-activity state. LTP or LTD is a persistent strengthening or weakening, respectively, of synapses based on recent patterns of activity that produce a long-lasting increase or decrease, respectively, in signal transmission between two neurons.

5. LIMITATIONS TO THE BIOLOGICAL PLAUSIBILITY OF AN ANN

Nowadays, there are still elements that are not biologically plausible in an ANN, such as the weight decay and temporal precedence described in BCM theory. There are implementations of algorithms that propose to take this into account, such as [34] that include STDP in ANNs.

Another limitation to ANNs is the biologically implausible backpropagation algorithm [20]. Whittingthon et al. [36] explore this in a review article with a main focus on the lack of local error representation. What is pointed out in [36] is that it appears unclear how the synaptic plasticity afforded by the backpropagation algorithm could be achieved in the brain, since biological synapses change their connection strength based solely on local signals, that is, the activity of the neurons they connect. According to equation (10) and (12), this is currently not the case since a weight update depends on multiple non-local and more downstream terms. There are several studies that try to overcome this [23, 26]. In [36] alternatives are suggested such as temporal-error models like contrastive learning [30] and an energy-based continuous update model [2], or explicit error models like a predictive coding model [37] and a dendritic error model [29, 14]. However, all models introduce new biologically implausible ideas, such as control signals, phases, unrealistic architectures, and extra connections.

Additionally, a biologically problematic aspect described by [36] is the unrealistic model of a neuron in an ANN with regard to their continuous output. Real neurons use spikes. Sporea et al. [33] introduce a supervised learning algorithm for multilayer spiking neural networks (SNNs), which do not fire at each propagation cycle (as in a typical ANN), but rather fire only when a membrane potential reaches a specific value (the threshold). Hence, SNNs use a binary output instead of a continuous output of ANNs. Unfavorably, the current activation level of a neuron is modeled as a differential equation in a SNN, which eliminates training methods like BP-SGD. Furthermore, SNNs have much larger computational costs than ANNs.

6. QUANTIFYING HEBBIAN THEORY

In order to measure the influence of a Hebbian learning rule, the theory must be quantified mathematically. Hebb's rule says that the synaptic efficiency depends on the firing of the pre-synaptic neuron and the post-synaptic neuron. In other words, the weight w_{ji}^l of a connection depends on the activation of the pre-synaptic neuron a_i^{l-1} and the activation of the post-synaptic neuron a_j^l. Since Hebbian Theory only describes conditions for connection strengthening (w_{ji}^l increase), conditions should be added that allow for weight-decay.

To devise a formula for a weight update rule that adheres to Hebbian Theory, it is easiest to think of activations in a binary way. A possible set of conditions in all 2^2 possible scenarios (when activations are binary) is the following:

1. $a_i^{l-1} = 0 \wedge a_j^l = 0 \implies w_{ji}^l(t+1) = w_{ji}^l(t)$
2. $a_i^{l-1} = 0 \wedge a_j^l = 1 \implies w_{ji}^l(t+1) < w_{ji}^l(t)$
3. $a_i^{l-1} = 1 \wedge a_j^l = 0 \implies w_{ji}^l(t+1) < w_{ji}^l(t)$
4. $a_i^{l-1} = 1 \wedge a_j^l = 1 \implies w_{ji}^l(t+1) > w_{ji}^l(t)$

In this paper, equation (16) is proposed that adheres to these conditions:

$$w_{ji}^l(t+1) = (a_i^{l-1} \cdot a_j^l - |a_i^{l-1} - a_j^l|) \cdot c + w_{ji}^l(t) \quad (16)$$

Where c is a real constant that can take on any (positive) value, and time t is indicated between brackets.

It is easily observed that equation (16) is not limited to binary activations, but also works for real activations. Although this is a way to quantify Hebbian learning for a weight, it is not the only way. Many equations could be adhering to the conditions stated above.

Furthermore, the conditions itself are rather arbitrary. Only condition 4 is certain when talking about Hebbian learning, because Hebbian theory doesn't mention anything

about weight-decay. Hence, the resulting relation between $w_{ji}^l(t+1)$ and $w_{ji}^l(t)$ could be diversified for condition 1-3. For example, another configuration of the conditions could be one where condition 1, 3, 4 remain the same and condition 2 is changed: $a_i^{l-1} = 0 \wedge a_j^l = 1 \implies w_{ji}^l(t+1) = w_{ji}^l(t)$. This is inspired by the *imply*-operator in logic truth tables. In Table 1, the truth table for an imply operator is shown. Imagine $p = a_i^{l-1}$ and $q = a_j^l$, where $(p \implies q) = 0$ means a weight-decay and any other case no weight decay. In this paper, equation (17) is proposed

Table 1. Truth table of logical implication

p	q	$p \implies q$
0	0	1
0	1	1
1	0	0
1	1	1

that adheres to this configuration of conditions:

$$w_{ji}^l(t+1) = (2 \cdot a_i^{l-1} \cdot a_j^l - a_i^{l-1}) \cdot c + w_{ji}^l(t) \tag{17}$$

For the sake of clarity, the configurations for the conditions of equation (16) and (17) could be translated to Table 2. Fortunately, more of such configurations have been extensively explored in earlier work [12]. For example, another equation that adheres to the condition configuration according to the last column of Table 2 that is proposed in [12] is $w_{ji}^l(t+1) = (a_j^l - c) \cdot a_i^{l-1} + w_{ji}^l(t)$ for $0 < c < 1$.

Table 2. Conditions for equation (16) and (17)

pre	post	equation (16)	equation (17)
a_i^{l-1}	a_j^l	$w_{ji}^l(t+1) \sim w_{ji}^l(t)$	$w_{ji}^l(t+1) \sim w_{ji}^l(t)$
0	0	$=$	$=$
0	1	$<$	$=$
1	0	$<$	$<$
1	1	$>$	$>$

Which configuration of conditions one chooses to implement Hebbian learning in an ANN is not necessarily very important, as long as one understands the meaning of the conditions. For example, what would be the meaning behind the difference in the 2^{nd} condition, where for equation (16) it causes a weight-decay, but for equation (17) it doesn't? It turns out that equation (16) is a more competitive version of Hebbian learning, whereas equation (17) is not. Say, the post-synaptic neuron a_j^l has multiple pre-synaptic neurons (e.g. a_1^{l-1}, a_2^{l-1}, a_3^{l-1}), which is usually the case in both an ANN and the brain, then the antecedents will start to compete for a "strong" connection. If the post-synaptic neuron is very activated ($a_j^l \approx 1$), then this is caused by a high activation in at least one of the pre-synaptic neurons a_i^{l-1}. Consider the situation where only the second one is activated (i.e. $a_1^{l-1} \approx 0$, $a_2^{l-1} \approx 1$, $a_3^{l-1} \approx 0$), then only the connection between the 2^{nd} pre-synaptic neuron and the post-synaptic neuron w_{j2}^l will strengthen. This causes the connections between the post-synaptic neuron and both the 1^{st} and 3^{rd} pre-synaptic neuron w_{j1}^l, w_{j3}^l to weaken. In other words, the increase of the strength of certain synapses onto the post-synaptic neuron is accompanied by a simultaneous decrease of the strength of other synapses onto that same post-synaptic neuron [3].

In [3] it was found that a complementary statement to Hebb's principle that gives conditions for synaptic decrease, usually results in a form of synaptic competition, which is also called "spatial competition between convergent afferents". BCM theory [3] tries to overcome this by proposing a synaptic modification that results in *temporal* competition between input patterns rather than a *spatial* competition between different synapses. Whether the synaptic strength increases or decreases depends on the magnitude of the post-synaptic response as compared with a variable modification threshold:

$$\dot{w_{ji}^l} = \phi(a_j^l) \cdot a_i^{l-1} - \epsilon \cdot w_{ji}^l \tag{18}$$

where

$$\phi(a_j^l) = sign(a_j^l - \theta_w) \tag{19}$$

Here, $\phi(a_j^l)$ is a scalar function of the post-synaptic activity a_j^l that changes sign at a value θ_w of the output called 'the modification threshold'.

$$\phi(a_j^l) \begin{cases} < 0 \text{ for } a_j^l < \theta_w, \\ > 0 \text{ for } a_j^l > \theta_w \end{cases} \tag{20}$$

The term $-\epsilon w_{ji}^l$ produces a uniform decay over all junctions. This does not affect the behaviour of the system if ϵ is small enough. In [3] it was proposed to use the average output activation $\overline{a_j^l} = \sum_i w_{ji}^l \cdot \overline{a_i^{l-1}}$, which averages over a distribution of inputs $\overline{a_i^{l-1}}$, to determine the threshold θ_w:

$$\theta_w = (\frac{\overline{a_j^l}}{a_0})^p \cdot \overline{a_j^l} \tag{21}$$

where a_0 and p are hyper-parameters.

Although the quantifications of Hebbian theory and BCM theory can now be used to mathematically describe the life-time of a connection strength w_{ji}^l, they cannot replace SGD directly. Just implementing the Hebbian rules in an ANN, described by equation (16) and (17), will cause the weights to update accordingly, but the network will not learn. After training, the imply Hebbian learning rule gains an accuracy of 0.09872 with a corresponding loss of 0.57892 over 20 epochs. The competitive Hebbian learning rule gains an accuracy of 0.08780 with a corresponding loss of 0.44378. Both don't get better than a random classifier ($\approx 10\%$). The reason for this is simple: an ANN cannot learn without knowing something about the error between the prediction and the target output. A solution to this problem would be to modify traditional SGD to make it behave more biologically plausible. This way, information about the error remains available.

7. HEBBIAN PLASTICITY TERM

This paper proposes a new method that is a Hebbian modification to SGD, which will be called the **Hebbian Plasticity Term (HPT) method**:

$$z_j^l = \sum_i (w_{ji}^l + \varphi_{ji}^l) \cdot a_i^{l-1} + b_j^l \tag{22}$$

where $\varphi_{ji}^l = f((a_i^{l-1})_{avg}, (a_j^l)_{avg}))$.

Equation (22) is a modified version of weighted input sum z_j^l in the feedforward described by equation (1). The difference is the addition of the *Hebbian Plasticity Term (HPT)* φ. You could say the weight in the feedforward is replaced by a new weight w_{new}, such that $z_j^l = \sum_i w_{new} \cdot a_i^{l-1} + b_j^l$, where $w_{new} = (w_{ji}^l + \varphi_{ji}^l)$.

The HPT φ suggests a 'nudge' in the right direction for the weight w_{ji}^l based on values of the activation of the pre-a_i^{l-1} and post-synaptic neuron a_j^l, according to a Hebbian learning rule f similar to equation (16) or (17).

A literal quote described in Hebb's postulate is *"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 takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased"* [16]. Important here, is the word *repeatedly*. This must mean that the synaptic efficiency is based on a multitude of pre- and post-activations from the past. The synapse grows based on the history of the two cells that connect it. So somehow, a weight of a connection w^l_{ji} should be dependent on some Hebbian function of the **history** of both pre- and post-synaptic activation.

$$\varphi^l_{ji} = f((a^{l-1}_i)_{avg}, (a^l_j)_{avg}) \tag{23}$$

Here $(a^{l-1}_i)_{avg}$ and $(a^l_j)_{avg}$ are both a moving average over all previous training examples that were already feedforwarded.

To explain this idea, consider the current feedforward of a training example to be on time t, then the feedforward of the previous training example would be at time $t-1$. Using the post-synaptic activation $a^l_j(t)$ to calculate the HPT φ^l_{ji} would not be possible at time t in the HPT method (22), because it is impossible for the HPT φ^l_{ji} to be dependent on the post-synaptic activation a^l_j since information about a^l_j is not available yet when calculating z^l_j. This is a circle argumentation. The weight cannot be dependent on the post-synaptic activation as the post-synaptic activation is also dependent on that weight. This results in an infinite circular dependence, which is incontractible. However, the post-synaptic activation of a **previously** feedforwarded training example could be used to compute the HPT φ^l_{ji}, since the feedforward of the training example at time $t-1, t-2, ...,$ or $t-n$ (where n is the total number of completed feedforwarded training examples) is already completed and is therefore already known/available at time t.

From this, an average post-synaptic activation $(a^l_j)_{avg} = \frac{a^l_j(t-1)+a^l_j(t-2)+...+a^l_j(t-n)}{n}$ over the previous training examples can be computed. After every feedforward of a new training example, the average $(a^l_j)_{avg}$ would slightly change, making it a moving average over all previously feed-forwarded training examples.

As explained in section 6, there are several Hebbian rules that could be chosen for f in equation (23), based on the configuration of conditions you choose. Therefore, two variants of the HPT method will be proposed. In the first variant, the HPT φ is based on a competitive Hebbian learning rule. In this variant, the HPT φ adheres to the conditions of equation (16) displayed in Table 2. This version will be called **HPT-Competitive**:

$$f = ((a^{l-1}_i)_{avg} \cdot (a^l_j)_{avg} - |(a^{l-1}_i)_{avg} - (a^l_j)_{avg}|) \cdot c \tag{24}$$

In the second variant, the HPT φ is based on an imply Hebbian learning rule. In this variant, the HPT φ adheres to the conditions of equation (17) displayed in Table 2. This version will be called **HPT-Imply**:

$$f = (2 \cdot (a^{l-1}_i)_{avg} \cdot (a^l_j)_{avg} - (a^{l-1}_i)_{avg}) \cdot c \tag{25}$$

In both equation (24) and (25), c can take on any real nonnegative value. Increasing c would mean to increase the influence of the Hebbian plasticity term φ on w_{new} in equation (22). In other words, it would mean to increase the influence of the Hebbian plasticity term on the feedforward.

An increase in c also means an increase of the influence of the HPT φ on the weight update $\frac{\partial C}{\partial w^l_{ji}}$ and bias update $\frac{\partial C}{\partial b^l_j}$ during backpropagation. According to equation (10), (11) and (12), both updates depend on $\frac{\partial C}{\partial a^l_j}$, which in turn depends on $\frac{\partial z^{l+1}_k}{\partial a^l_j}$, and for the HPT method $\frac{\partial z^{l+1}_k}{\partial a^l_j} = w^{l+1}_{kj} + \varphi^{l+1}_{kj}$

It is important that the HPT φ is located exactly where it is in the feedforward equation (22). Locating it elsewhere would result in undesired behaviour. Consider $z^l_j = \sum_i w^l_{ji} \cdot (a^{l-1}_i + \varphi) + b^l_j$. If $w^l_{ji} < 0 \wedge \varphi^l_{ji} < 0$ (where the weight should become smaller according to the Hebbian function f), then the resulting z^l_j (which, according to equation (1) in turn affects the output activation a^l_j) becomes larger since a positive number is obtained when multiplying negatives $(w^l_{ji} \cdot \varphi^l_{ji}) > 0$. However, the reason why φ obtained a negative value in the first place was to decrease the efficiency of the synapse, so that the output activation a^l_j will decrease. Also, when $w^l_{ji} < 0 \wedge \varphi^l_{ji} > 0$, then $(w^l_{ji} \cdot \varphi^l_{ji}) < 0$ will make z^l_j become smaller, which is again not desired since z^l_j should become bigger when $\varphi > 0$.

8. IMPLEMENTATION

Algorithm 1 shows the pseudocode for the algorithm that was created as the implementation for backpropagating in the ANN. It is created using Python 3.6.2, with libraries Numpy for vector calculations and Matplotlib.pyplot for displaying results graphically. Equation (14) is incorporated here on `line 6`. This code is used for the HPT method, where modifications to standard SGD are marked in cyan.

Algorithm 1: Backpropagation with HPT method

```
// Initialize ∂C/∂b^L and ∂C/∂w^L by the last layer
1  ∂C/∂b^L ← σ'(z^L) ⊙ (a^L − y)
2  ∂C/∂w^L ← ∂C/∂b^L · (a^{L−1})^T

// Add both to the list
3  gradient_bias ← [∂C/∂b^L]
4  gradient_weights ← [∂C/∂w^L]

// Backpropagate through all layers starting
   from the one-but last layer, since you
   already added the last layer with
   initialization
5  for k ← 1 to N do
6  |  ∂C/∂b^L ← σ'(z^{L−k}) ⊙ (W^{L−k+1}+φ^{L−k+1})^T · ∂C/∂b^L
7  |  ∂C/∂w^L ← ∂C/∂b^L · (a^{L−k−1})^T
8  |  gradient_bias.append(∂C/∂b^L)
9  |  gradient_weights.append(∂C/∂w^L)
10 end

// Reverse the lists so they contain vectors
   from first to last layer instead of from
   last to first layer
11 gradient_bias.reverse()
12 gradient_weights.reverse()
```

Important things to note are the following. All terms are considered matrices. A vector is a column-vector (a matrix with one column and multiple rows) unless referred to otherwise, like a row-vector.

φ^l in Algorithm 1 is calculated according to equation (24) and (25) for HPT-Competitive and HPT-Imply, respectively. Both variants of the HPT method need the moving average of activations $(a_i^{l-1})_{avg}$ and $(a_j^l)_{avg}$ of every layer in the network. This list of the moving average of activations for all per-layer (moving average of-)activation vectors is initialized as a list `avg_acts`. After every mini-batch, the average activation of the mini-batch for every (per-layer) vector, located in the list `batch_avg_acts`, is added to every (per-layer) vector located in `avg_acts`. This is displayed in Algorithm 2. Interesting to note is

Algorithm 2: avg_acts calculation

```
// Update moving average avg_acts after a
   batch
for act_vec ∈ batch_avg_acts do
    avg_acts[act_vec] += batch_avg_acts[act_vec]
end
avg_acts = [history / 2 for history in avg_acts]
```

that the moving average of all previous activations counts as much as the newly added batch-average of activations, since it is divided by 2. This way, the most recent activations in the network have the most influence on $(a)_{avg}$ and thus on φ. The source code for the complete project can be found on: https://github.com/matthijsruben/NeuralNets

9. RESULTS

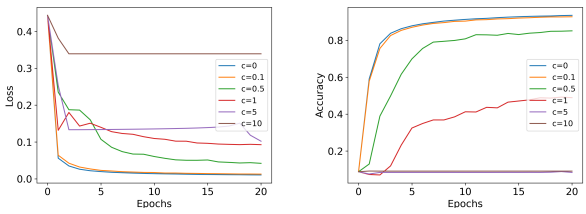
9.1 HPT method

In this section, the performances of novel methods proposed in this paper will be shown and compared with the performance of a standard ANN. All networks are trained on the MNIST dataset [19], because of its large amount of training examples. This dataset contains 28x28 pixel (greyscale valued: 0-255) images of handwritten digits. The benchmark ANN from which the different methods will be evaluated is a four-layer deep feed-forward ANN with 784, 30, 20, and 10 neurons in each respective layer. Since neurons in this network are activated with a sigmoid function, the 784 greyscale pixel values of each image are downscaled to values between 0 and 1, and form the input layer $a_1^0, a_2^0, ..., a_{784}^0$ of the network. The output layer contains ten neurons, since there are ten classes (10 different digits). The weights and biases of the network are initialized as random variables drawn from a standard normal distribution. The loss is calculated using a mean squared error function, but for backpropagation equation (6) is used. The network is trained in 20 epochs using mini-batch stochastic gradient descent with a batch-size of 28 training examples and a learning rate of 0.5.

In Figure 4, the results of the HPT-Competitive method are shown. For larger values of c in equation (24), the HPT φ gains a larger influence over w_{new}, but it is observed to slow down learning.

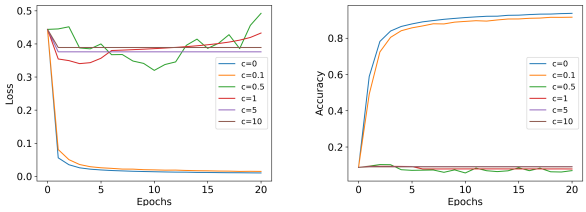
In Figure 5, the results of the HPT-Imply method are shown. Again, larger values of c are observed to slow down learning. The slowdown of learning in the HPT-Imply method even shows to be way more susceptible to an increase of c, than the slowdown of the HPT-Competitive method.

The results are summarized in Table 3 for different values of c. For $c = 0$, the HPT $\varphi = 0$ and the new weight equals the standard weight $w_{new} = w_{ji}^l + 0$. Therefore, standard SGD is applied when $c = 0$, because there is no influence



(a) Loss function (b) Accuracy

Figure 4. HPT-Competitive method for different values of c



(a) Loss function (b) Accuracy

Figure 5. HPT-Imply method for different values of c

Table 3. Losses and Accuracy's of the HPT method for different values of c

	HPT-Competitive		HPT-Imply	
c	Loss	Accuracy	Loss	Accuracy
0	0.01053	0.93537	0.01034	0.93675
0.1	0.01276	0.92770	0.01495	0.91547
0.5	0.04201	0.85137	0.49220	0.06995
1	0.09267	0.48920	0.43276	0.07898
5	0.10206	0.08398	0.37599	0.09192
10	0.33938	0.09112	0.38906	0.09083

from the HPT φ.

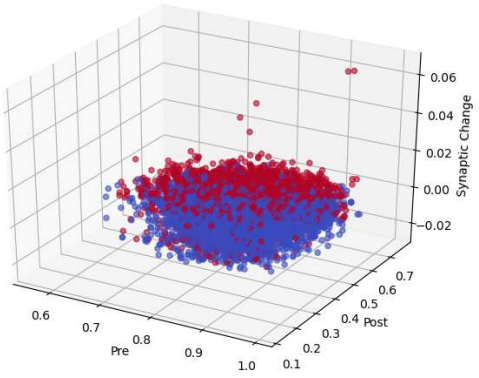
It can be concluded that the overall performance of standard SGD is better after 20 epochs than the overall performance of the HPT method. Perhaps more epochs will even the performance between standard SGD and the HPT method.

9.2 Biologically plausible synaptic behaviour

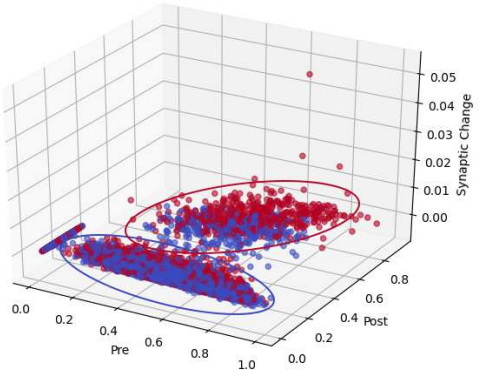
From the results, it can be observed that the HPT method trains slower than standard SGD. However, the synaptic change of a connection between two neurons (randomly chosen)[1] in the network shows to behave more biologically natural when using the HPT method (Competitive), as can be seen by comparing Figure 6a and 6b. Both graphs show a scatter plot, where each dot represents the value of an observed weight update of a connection between two neurons (pre- and post) by the ANN after training 5 epochs. The red dots show the positive weight updates and the blue dots the negative weight updates. In other words, a red dot represents a connection reinforcement and a blue dot represents a connection weakening.

In Figure 6a, there is no clear relation visible between the activation of the pre-synaptic neuron, the activation of the post-synaptic neuron, and the synaptic change (the

[1]To generate these results, the activation of the first neuron of the second layer and the activation of the first neuron of the third layer was chosen, as well as the weight updates of the connection between those neurons



(a) using standard SGD



(b) using the HPT method

Figure 6. Synaptic change of the connection between two neurons in the ANN

observed weight update). However in Figure 6b, a pattern is observed. There are two distinctive groups: the group circled in red and the group circled in blue. The group circled in blue contains mostly blue dots and the post-synaptic activations are low whatever the activations of the pre-synaptic neuron: no influence is observed from the pre- on the post-synaptic activation. So in most of the cases, the synaptic connection weakens if the pre- and post-activations are unrelated. The group circled in red contains mostly red dots. Also, visually some kind of linear dependence between the pre-synaptic activation and the post-synaptic activation can be recognized, observed by the diagonal shape of the red group in the 'pre' and 'post' dimensions of Figure 6b. This means that in most of the cases the synaptic connection strengthens if the pre- and post-activations are related. This is a desired behaviour from a Hebbian (and therefore biological) point of view, especially relating to its postulate *"Cells that fire together, wire together"*.

The conclusions are solely based on visual observation of the graphs in Figure 6, since no statistical significance test has been performed on these results. However, they do provide an intuition as to why the HPT method causes synapses to behave more biologically plausible.

10. CONCLUSIONS

In order to close the gap between artificial and biological neural networks, this paper aimed to incorporate mathematical translations of Hebbian theory into SGD.

Just like the authors of [3] found, a limit to Hebbian theory is that it only describes conditions for connection strengthening. Therefore, conditions should be added that allow for weight decay. Consequently, two variants of a quantification of Hebbian theory are proposed: the competitive Hebbian learning rule, equation (16), and the imply Hebbian learning rule, equation (17). Even though they are more biologically plausible, both variants don't show good learning behaviour.

The HPT method that is proposed in this paper modifies the standard training algorithm SGD that is commonly used for ANNs. Based on the proposed variants of Hebbian learning, two variants to the HPT method are proposed: HPT-Competitive and HPT-Imply. By including the HPT φ in the feedforward equation (1), both the feedforward and backpropagation are shown to be influenced by Hebbian learning. Because of this, not only behaviour but also the way connections evolve (plasticity) is affected by Hebbian learning. This influence can be extended and controlled to a certain degree by parameter c, as shown in equation (24) and (25).

The influence of the size of the parameter c of the HPT method is observed to slow down learning. However, the performance of the HPT-Competitive method is less susceptible to the increase of c than the HPT-Imply method, because the HPT-Competitive method shows better performance than the HPT-Imply method for similar values of c. From this can be concluded that the type of Hebbian learning rule used in the HPT method has a major influence on the performance of the ANN.

The HPT-method makes connections behave and evolve more biologically realistic at a slight loss in performance. When the activation of the pre- and post-synaptic neurons are visually observed to be related, the connection between those neurons shows to be reinforced more often than weakened. When the activation of the pre- and post-synaptic neurons are visually observed to be unrelated, the connection shows to be weakened more often than reinforced. Hence, there is a trade-off between the performance and biologically realistic behaviour in the deep feedforward ANN that was used.

11. FUTURE WORK

As explored in [12], there are many more variants of Hebbian learning and mathematical descriptions can vary endlessly. Future research could try to compare multiple different Hebbian learning rules and what their effect is on the performance of the HPT method, since they show to be varyingly sensitive to the size of parameter c.

Also, `avg_acts` from section 8 is now calculated based on an unevenly weighted moving average, where the most recent activations have the most influence. It would be interesting to see an implementation that compares the results of other types of moving averages, such as one that is evenly weighted or exponentially weighted.

12. ACKNOWLEDGMENTS

I want to thank my supervisor Elena Mocanu, who guided me during my research, provided me with so much helpful and extensive feedback, and always wanted to get the best out of me.

13. REFERENCES

[1] 3BLUE1BROWN. Backpropagation calculus | Deep learning, chapter 4 - YouTube, 11 2017.

[2] Yoshua Bengio, Thomas Mesnard, Asja Fischer, Saizheng Zhang, and Yuhuai Wu. STDP-compatible approximation of backpropagation in an energy-based model. *Neural Computation*, 29(3):555–577, 3 2017.

[3] Elie L. Bienenstock, Leon N. Cooper, and Paul W. Munro. Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2(1):32–48, 1982.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer (India) Private Limited, New York, 2006.

[5] Rodney Brooks, Demis Hassabis, Dennis Bray, and Amnon Shashua. Turing centenary: Is the brain a good model for machine intelligence? *Nature*, 482(7386):462–463, 2 2012.

[6] Natalia Caporale and Yang Dan. Spike Timing–Dependent Plasticity: A Hebbian Learning Rule. *Annual Review of Neuroscience*, 31(1):25–46, 7 2008.

[7] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167, 2008.

[8] Haskell B. Curry. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261, 10 1944.

[9] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 8599–8603, 10 2013.

[10] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(3-4):197–387, 2014.

[11] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):677–691, 4 2017.

[12] Wulfram Gerstner and Werner M. Kistler. Mathematical formulations of Hebbian learning. *Biological Cybernetics*, 87(5-6):404–415, 2002.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[14] Jordan Guerguiev, Timothy P. Lillicrap, and Blake A. Richards. Towards deep learning with segregated dendrites. *eLife*, 6, 12 2017.

[15] Prashanth Gurunath Shivakumar and Panayiotis Georgiou. Transfer learning from adult to children for speech recognition: Evaluation, analysis and recommendations. *Computer Speech and Language*, 63, 9 2020.

[16] Donald Olding Hebb. *The Organization Of Behavior*. Wiley & Sons, New York, 1949.

[17] Eric R Kandel. *Principles of Neural Science*. McGraw-Hill Education, 5 edition, 2012.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 5 2015.

[19] Yann LeCun, Corinna Cortes, and Chris Burges. MNIST handwritten digit database.

[20] Timothy P. Lillicrap, Adam Santoro, Luke Marris, Colin J. Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, pages 1–12, 4 2020.

[21] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 4 2017.

[22] Siegrid Löwel and Wolf Singer. Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity. *Science*, 255(5041):209–212, 1 1992.

[23] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. Deep supervised learning using local errors. *Frontiers in Neuroscience*, 12(AUG), 11 2017.

[24] Neuroscientifically Challenged. 2-Minute Neuroscience: Synaptic Transmission - YouTube, 7 2014.

[25] Michael A. Nielsen. Neural Networks and Deep Learning, 2015.

[26] Randall C. O'Reilly. Biologically Plausible Error-Driven Learning Using Local Activation Differences: The Generalized Recirculation Algorithm. *Neural Computation*, 8(5):895–938, 7 1996.

[27] Osmosis. Neuron action potential - physiology - YouTube, 12 2016.

[28] Sushant Patrikar. Batch, Mini Batch & Stochastic Gradient Descent - Towards Data Science, 10 2019.

[29] João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. Technical report, 2018.

[30] Benjamin Scellier and Yoshua Bengio. Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation. *Frontiers in Computational Neuroscience*, 11:24, 5 2017.

[31] Jürgen Schmidhuber. Deep Learning in neural networks: An overview. *Neural Networks*, 61:85–117, 1 2015.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015.

[33] Ioana Sporea and André Grüning. Supervised learning in multilayer spiking neural networks. *Neural Computation*, 25(2):473–509, 2013.

[34] Roland A Suri. A computational framework for cortical learning. *Biological Cybernetics*, 90:400–409, 2004.

[35] George B. Thomas, Joel. Hass, and Maurice D. Weir. *Thomas' calculus*. Pearson Addison Wesley, 13 edition, 2 2014.

[36] James C R Whittington and Rafal Bogacz. Theories of Error Back-Propagation in the Brain. *Trends in Cognitive Sciences*, 23(3), 2019.

[37] James C.R. Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Computation*, 29(5):1229–1262, 5 2017.