Technische Hochschule Ulm

Bachelor's Thesis

Biological Learning Algorithms and Predictive Coding for Image Classification

Dhiego Bersan

Munich

2021

Abstract

In the field of artificial intelligence, backpropagation is a learning algorithm for artificial neural networks that has had considerable success on different applications of machine learning. However, this brain-inspired algorithm shows an array of limitations that natural brains do not seem to have. Significant effort has then been put into developing different forms of learning algorithms, which could hint to how intelligence is implemented by the brain and possibly pave the way to more powerful forms of machine intelligence. In this thesis, first, a general overview of biological neural networks is given, and the mechanisms that underlie the functions of real neurons, such as action potentials and neural modulation are discussed and compared with the neuron representation commonly used in artificial networks. The concept of biological learning algorithms is then introduced, which are more suited to operate using brain-based mechanisms, and therefore can be considered more biologically-plausible and possibly be executed in a new class of neuromorphic chips that have gained increasing traction. One such algorithm, called Predictive Coding, is explored more in depth. The algorithm is tested for image classification using features extracted from a convolutional neural network, using a *PyTorch* implementation. The results are compared to image classification results using standard backpropagation approach. For all experiments, the Predictive Coding networks outperformed the backpropagation-based networks, except for training time.

Contents

1 Introduction

1.1 Motivation

Deep Neural Network algorithms (DNNs) based on backpropagation have showed outstanding results in machine learning tasks in a wide range of applications. These include image recognition, text generation, reinforcement learning agents, and so on. The layered neuronal topology of artificial networks is inspired by a brain-like mechanism, in which neurons communicate information through adjacent layers, using connection weights to determine the output of each neuron. These computations are represented as matrix multiplications, in conjunction with a variety of non-linear operations.

Despite their relative success, backpropagation-based deep neural networks seem to present major drawbacks that put in question their ability to deliver practical and general forms of intelligence – such as those we observe in humans. Three such hurdles may be noted, and are discussed here: energy consumption, tendency to forget and the requirement of massive datasets.

Energy consumption: It not unheard of modern, larger DNN models carrying almost astronomical electricity bills. One example, *GPT-3* (Brown et al., 2020) is currently one of the largest artificial neural network model, in terms of trainable parameters. Working under its full capacity, roughly 170 billions neuron connections are trained, and each training session consumes around 190,000 kWh of energy ("GPT-3 Power Consumption", 2021). This not only implies environmental and scalability issues, but also makes the use of these more powerful networks on off-grid systems, such as self-driving cars and intelligent mobile robots, impractical.

Tendency to forget: Another pressing issue for current deep learning systems is that such networks unlearn previously learned weights when trained with new data. This phenomenon is called *catastrophic inference* (McCloskey & Cohen, 1989). When training deep neural networks through backpropagation, the *entire* data set must be present, and if new training data is to be added later, this data must be included in the dataset and the network must be retrained. Otherwise, the model parameters are updated to fit to the new data, overwriting the previous connections. Models that account for temporal data, such as *Long Short Term Memory* Networks (LSTMs) (Hochreiter & Schmidhuber, 1997), try to circumvent this issue and allow the network to persist previous inputs in special mechanisms that mimics memory. But they show sub-par results for tasks regarding for instance, image classification. Once trained on a set of data samples, deep networks generally are unable to learn new concepts.

Large Datasets: Finally, deep neural networks require very large amounts of data to train. Moreover, very often this data has to be "labeled", which is very costly for larger datasets, and not easily scalable, because each sample needs to be manually identified. VGG-16 (Simonyan & Zisserman, 2014), for instance, was trained on the ImageNet dataset (Deng et al., 2009), with over 1.3 million labeled images. This same data is then fed over and over to the network. VGG-16 itself was trained on 74 epochs, which means every single image was processed by the network 74 times. Another example is OpenAI Five ("OpenAI Website", 2021), a type of reinforcement learning network that learned to play *Dota 2*, an

online video game, arguably surpassing human-expert level. To that, the network had to train while playing the game for over 10,000 years of simulated game time. These figures show the apparent inability of deep networks to learn quickly and generalize from limited amounts data.

Biological brains, on the other hand, do not seem to present any of the aforementioned issues. A human brain contains roughly 600 trillion neuron connections, but consumes only 20 watts of power. Such biological network is 3500 times larger than *GPT-3*, and consumes orders of magnitude less energy, displaying an unmatched level of energy efficiency. Also, new input data is successfully accounted for without requiring re-training over all past inputs, and intelligent memory mechanisms allows learning with very little supervision – 1 or 2 images of an unseen object are generally enough to learn the relevant features of such object.

Those observations make a compelling argument for exploring other learning mechanisms and even network architectures, that better approximate biological function, in hopes that this could give a hint to how the brain implements intelligence, and pave the way for next generation of Artificial Intelligence.

1.2 Contribution

This thesis tackles the question *"How could biological algorithms of learning be implemented?"* To that, a general overview of backpropagation, the workings of biological neurons, and the developments in a biologically realistic network model, Spiking Neural Networks, are given. The limitations of applying backpropagation to such biologically realistic models of networks is also discussed.

One existing biologically plausible learning algorithm, *Predictive Coding*, is then explored more in-depth. The thesis aims to verify if this algorithm could be improved for computer vision tasks, utilizing *Convolutional Neural Networks* (CNNs). This is a brain-inspired network architecture for image processing and other spatial types of data, commonly used in artificial neural networks. Experiment results from a "PyTorch" (2021) implementation of the proposed method are shown.

1.3 Thesis Structure

Section 2 discusses the backpropagation algorithm, biological networks and neurons, and introduces ideas of artificial intelligence inspired by such biological characteristics. Those include spiking neural networks, neuromorphic hardware and biological algorithms of learning.

One such learning algorithm, Predictive Coding, is presented and analysed further. In section 3, the methodology based on Predictive Coding and bottleneck feature extraction is elaborated and on section 4 the results of this implementation are discussed.

Finally, on section 5 a general conclusion of the predictive coding algorithm and a perspective on the current state of biological learning algorithms and possible future research directions are given.

2 Background

This section discusses the *Backpropagation* algorithm, gives a general overview of biological mechanisms of brain function, and presents alternative learning algorithms that would be implementable under biological constraints. One such algorithm, *Predictive Coding*, is discussed further. Finally, an overview of *Convolutional Networks* is given.

2.1 Backpropagation Algorithm

Backpropagation is the workhorse of deep learning, and has enabled unprecedented results in the Artificial Intelligence scene. In simple terms, backpropagation is an algorithm used to efficiently compute the connection weight updates of an artificial neural network (ANN), such that the desired set of responses are outputted, for a set of network inputs. In the process, the network should not memorize the correspondence between input/output pairs, but instead learn relevant representations, so that a desirable output is reached for an unseen input.

Considering initially the case of a simple deep neural network, represented in figure 1: each circle is a neuron, and the network has 3 layers, being the left-most the *input layer*, and the right-most is the *output layer*. In this case, both these layers have 2 neurons each, while the middle layer has 3 neurons. The neuron j of the layer l is referred to as x_j^l. The connections between the neurons of adjacent layers represent the *synapses*, and each connection has a given strength, called *connection weight*. The weight value of the connection between the neuron k of the layer $(l-1)$ to the neuron j of the layer number l is represented by w_{jk}^l.

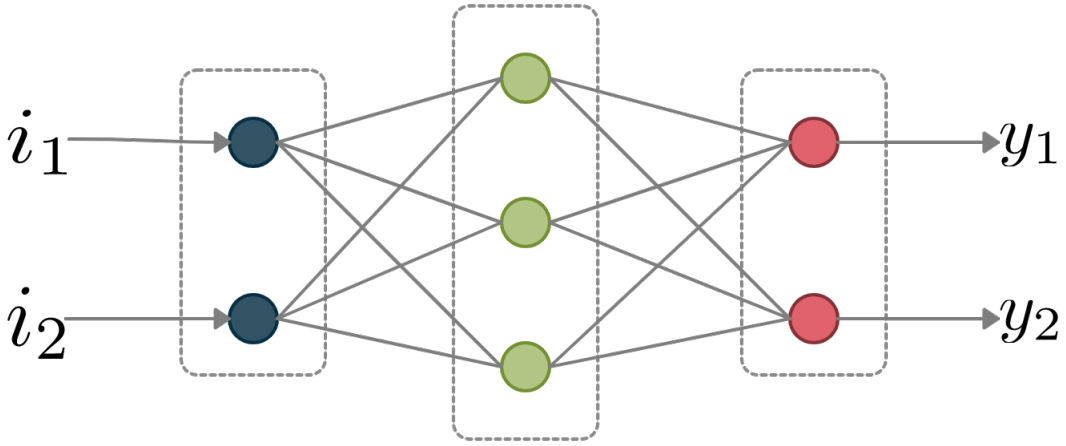

Figure 1: Representation of a simple deep neural network, with 3 layers, and input and output size equal to 2.

To obtain the value of the network output for a given input, a *feed-forward* operation is performed (which is also called an *inference* operation). Starting from the second layer, the

value of each neuron is the weighted sum of the values of the neurons from the previous layer. The fact that this operation necessarily updates neurons from each layer in sequence, gives the sense of information *flowing* to the right, hence the name *feed-forward*.

Activation functions placed at each neuron output serves the purpose of allowing the network to represent non-linear transformations. More specifically, given an activation function $\sigma(\cdot)$, the *activation* of the neuron x_i^l is given by equation 1.

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \tag{1}$$

Equation 1 may also be written in a composed form:

$$a_j^l = \sigma\left(z_j^l\right) \tag{2}$$

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \tag{3}$$

where z_j^l is called the *weighted input* of the neuron j of layer l.

The bias term, b_j^l, mathematically allows the graph of the linear combination of the summation to move vertically. On the perspective of the network itself, the bias is just a special weight, distinct for each neuron, with a corresponding activation always equal to 1. A representation of an artificial neuron under these considerations is shown in figure 2.

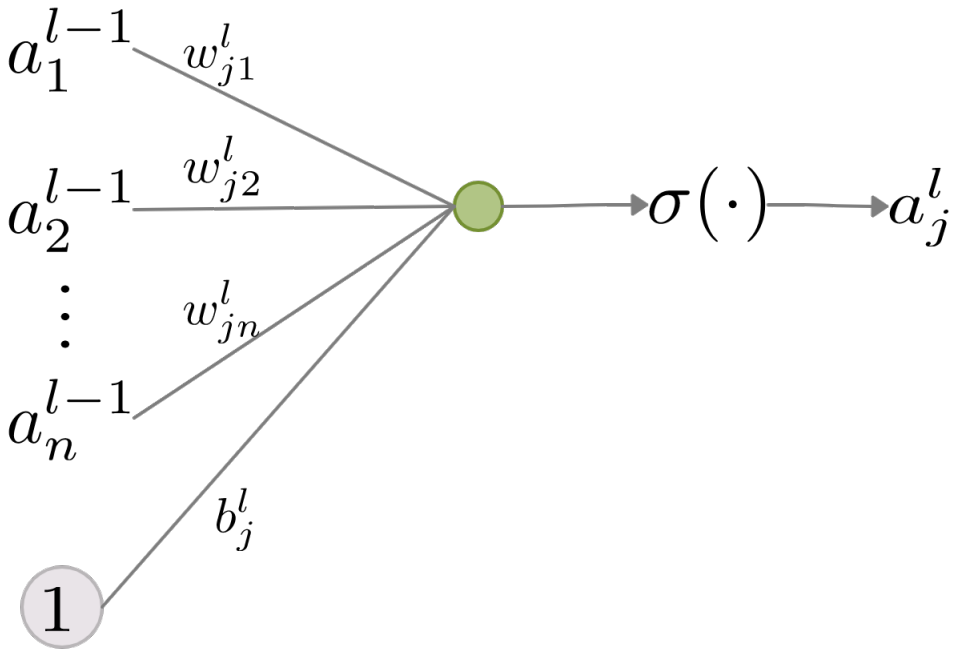

Figure 2: A single artificial neuron of a standard DNN.

Given a certain vector assigned to the left-most layer of the network $\mathbf{x} = (x_1^1, x_2^1, ...x_{(n)}^1)$ (layer $l = 1$, on figure 1), the *feed-forward* operation will propagate the values of neurons until the last layer, L, whose activation vector is denoted $\hat{\mathbf{y}} = (x_1^L, x_2^L, ...x_{(m)}^L)$. In this case, $\mathbf{x}$ is called the *input* and $\hat{\mathbf{y}}(\mathbf{x})$ is the *output* of the network, for that given $\mathbf{x}$. The distance between the output $\hat{\mathbf{y}}(\mathbf{x})$ and the ground-truth $\mathbf{y}(\mathbf{x})$ for a given $\mathbf{x}$ is the cost of the network for that input, C_x. That is, in the case when the *cost function* is the L2-norm squared, for instance:

$$C_x = \|\mathbf{y}(\mathbf{x}) - \hat{\mathbf{y}}(\mathbf{x})\|_2^2 \tag{4}$$

The idea of *learning* in deep neural network is the minimization of the average of C_x across N $(\mathbf{x}, \mathbf{y})$ pairs. This average cost is defined as the *cost* of the network, C:

$$C = \frac{1}{N} \sum_{i \in N} C_{x_i} \tag{5}$$

In order to find the correct *nudging amount* for each connection weight and bias in the network, such that the *cost* is minimized, backpropagation gives a method for calculating the derivative of this cost with respect to each weight and bias ($\partial C/\partial w$ and $\partial C/\partial b$). An intermediate variable, denoted the *error* of the neuron j at the layer l is:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{6}$$

The **equations of backpropagation** are then defined as equations 7, 8, 10 and 9 (L refers to the last layer of the network).

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L) \tag{7}$$

$$\delta_j^l = \left(\sum_k w_{jk}^{l+1} \delta_k^{l+1} \right) \cdot \sigma'(z_j^l) \tag{8}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{9}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{10}$$

The network weights may then be updated according to some update formula such as shown in equations 11 and 12, or through a more sophisticated algorithm, such as, e.g., *Adam Optimizer* (Kingma & Ba, 2014).

$$\omega_{jk}^l \leftarrow \omega_{jk}^l - \alpha \frac{\partial C}{\partial \omega_{jk}^l} \tag{11}$$

$$b_j^l \leftarrow b_j^l - \alpha \frac{\partial C}{\partial b_j^l} \tag{12}$$

2.2 Biological Networks

This section establishes biological fundamentals and parlance concerning the most relevant operations and properties related to the proposed theories of brain-inspired artificial networks.

2.2.1 Biological Networks and Learning

Figure 3 shows a simplified illustration of a brain cell (i.e., a neuron). Its inputs are the *dendrites* and its output is the *axon*. The cell body (or, the *soma*) is responsible for performing a kind of weighted sum of the electric potential of its inputs, and then send the result to the next neurons through its axon. This operation is similar to artificial neurons, as described by equation 1. While this sum in the artificial neuron is merely represented by a floating point number, in the biological neuron this value is represented the membrane voltage potential, polarized by the difference between concentration of ions inside and outside the cell.

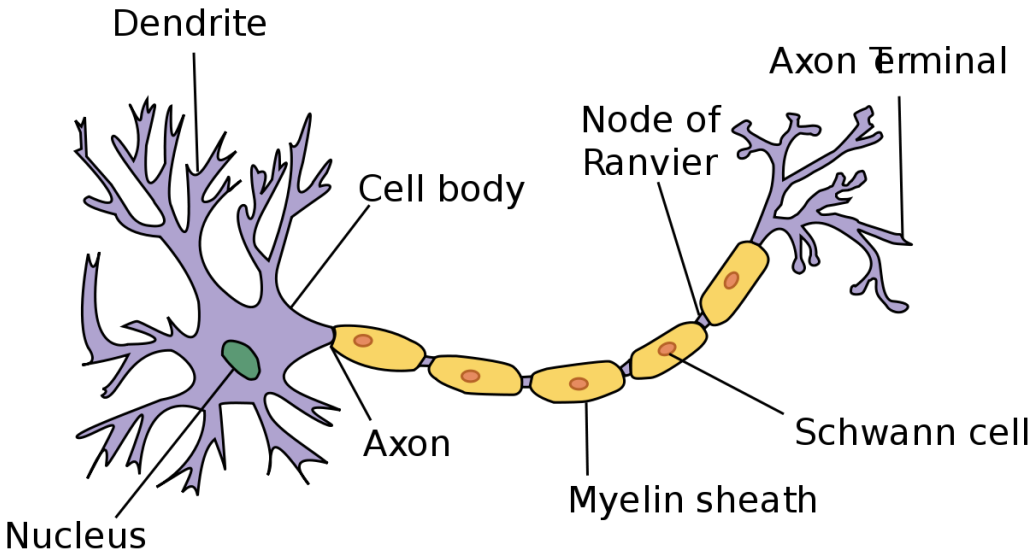

Figure 3: The representation of a biological neuron. Dendrites carry input signals from various neurons, and the axon carries the neuron's resulting output signal, that may connect to the dendrites of various other neurons.

Most similarities between artificial and real neurons, however, end there: real neurons have extremely complex structures and properties. They have a defined spatial location inside the brain, and their structures have real physical dimensions. Dendrites and axons can be much longer compared to the size of the cell body, therefore this implicates electrical conductance dynamics play a role in their function, such that different parts of the neuron have different voltage potentials (Lindsay, Lindsay, & Rosenberg, 2005). There are also different types of neurons in different regions of the brain: it is estimated that there are between 100 and 1000 different types just in the Cortex alone (Masland, 2004) ("How many neuron types are there", 2021). They may be classified based on number of dendrites,

length of axons, shape of the soma and their function (e.g, sensory neurons, interconnection neurons, etc).

The connection of one neuron's axon to the other's dendrite is enabled through a multitude of types of neurotransmitters. To name a few ("What are neurotransmitters?", 2021): *Amino acids* such as *glutamate, GABA, aspartate, etc* ; *Monoamines* such as *dopamine, norepinephrine, serotonin, etc*; *Peptides* such as *oxytocin, Cocaine and amphetamine regulated transcript, opioid peptides* . These neurotransmitters have different functions, which can be classified into: excitatoy, i.e., they increase membrane potential of the post-synaptic neuron; or inhibitory, i.e., they decrease the membrane potential of the receiving neuron. The type of the connection, inhibitory or excitatory, is mostly fixed, unless in the presence of neuromodulators, which will be covered later.

One distinct characteristic of biological neurons is the way information is encoded. Figure 4 shows examples of many patterns of neuronal responses, in terms of voltage potential in the soma of the cell. It evident these signals are composed of spikes, more commonly referred to as *action potentials*. Such spikes are believed to carry information, either through the spike rate (e.g. spikes per second) or on the precise time between spikes.

The probability of an action potential happening at any given moment is (mostly) a result of the weighted sum of the incoming action potentials from *pre-synaptic neurons*, i.e neurons whose axons are connected to the current neuron's dendrites. The weights of this sum is given by the connection strength of its dendrites. If enough spikes are inputted through the dendrites in a small enough time-window, given strong enough connections, the soma voltage will surpass a fixed voltage threshold, such that a spike happens. Figure 5 depicts such process.

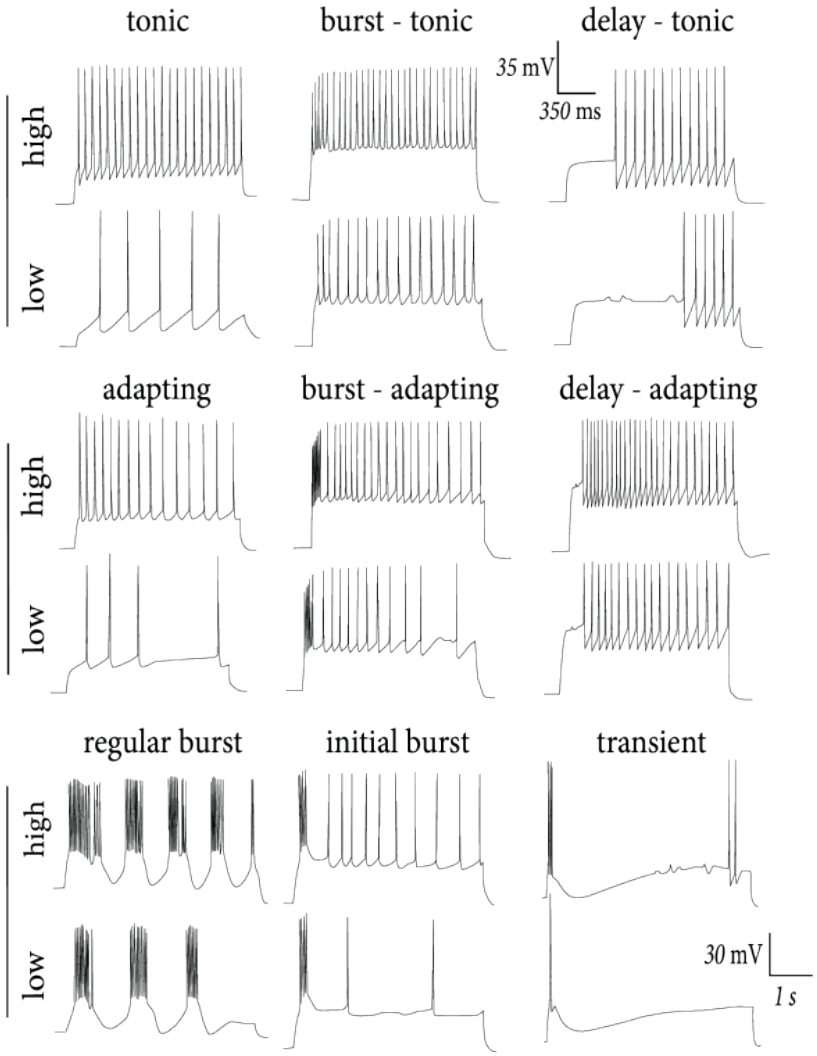

Figure 4: Various types of neuronal responses of neurons in the brain. The vertical axis indicates the voltage in the soma of the cell, while the horizontal axis is the time component. ("Neuronal Dynamics (book) Adaptation and Firing Patterns", 2021)

This probability of a spike, however, is also largely influenced by the presence of neuromodulators (Nadim & Bucher, 2014). These modulators may momentarily increase or decrease the strength of connections (*Neuromodulation of synaptic strength*) and also adjust the likelihood of neuron excitability (*Neuromodulation of neuronal excitability*). For instance, the presence of *dopamine* is known to significantly increase the strength of synapses, while *5-HT* has the inverse effect.

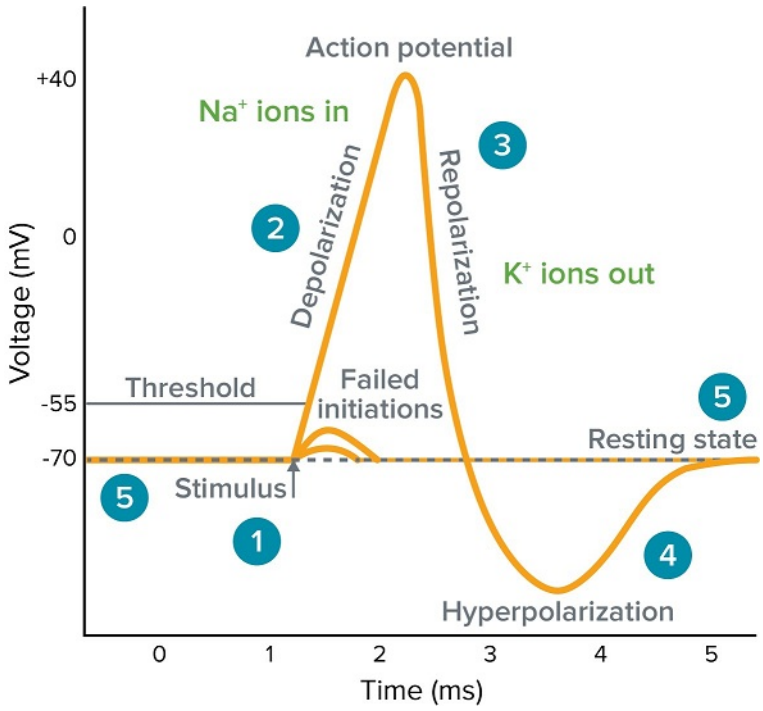

Figure 5: The graph of the voltage of a neuron during an action potential. ("What is an action potential?", 2021)

But how does a biological network actually learn? In other words, how are weights of connections between brain cells adjusted? And could that have anything to do with the backpropagation algorithm?

The answer to how biological brains operate is still a mystery, but the underlying mechanisms of how connection weights change at the level of individual neurons is fairly well understood. Generally speaking, it involves the spike timing of pre-synaptic and post-synaptic neurons (as known as *Spike Timing Dependent Plasticity*, or STDP) and the presence of neuromodulators.

STDP has its roots in *Hebbian Learning* (Hebb, 2005). Figure 6 shows a diagram of 2 neurons, where the left neuron is called *pre-synaptic* and the right one is the *post-synaptic* neuron. One of the main ideas proposed by Hebb (2005) is that when the post-synpatic neuron fires a spike *after* the pre-synpatic neuron fired a spike, the neuronal connection between them strengthens. Conversely, if the post-synaptic neuron fires *before* the pre-synaptic neuron (given a time window), then their connection weakens. STDP give a more precise definition of how much such connection weights change based on relative spike time between both neurons. This is represented by figure 7: the sooner a post-synaptic spike happens after the pre-synaptic spike, the more the connection strength is increased between them. But the inverse is also true, such that if the pre-synaptic spike happens right *after* the post-synaptic spike, their connection is weaken the most.

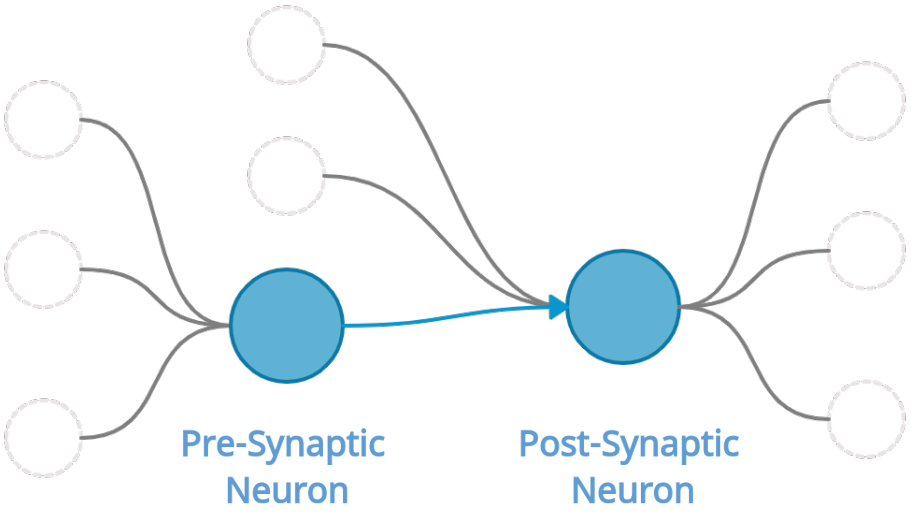

Figure 6: Simplified diagram of 2 neurons connected. *PRE* indicates a *pre-synaptic* neuron, whist *POST* is a *post-synaptic* neuron.

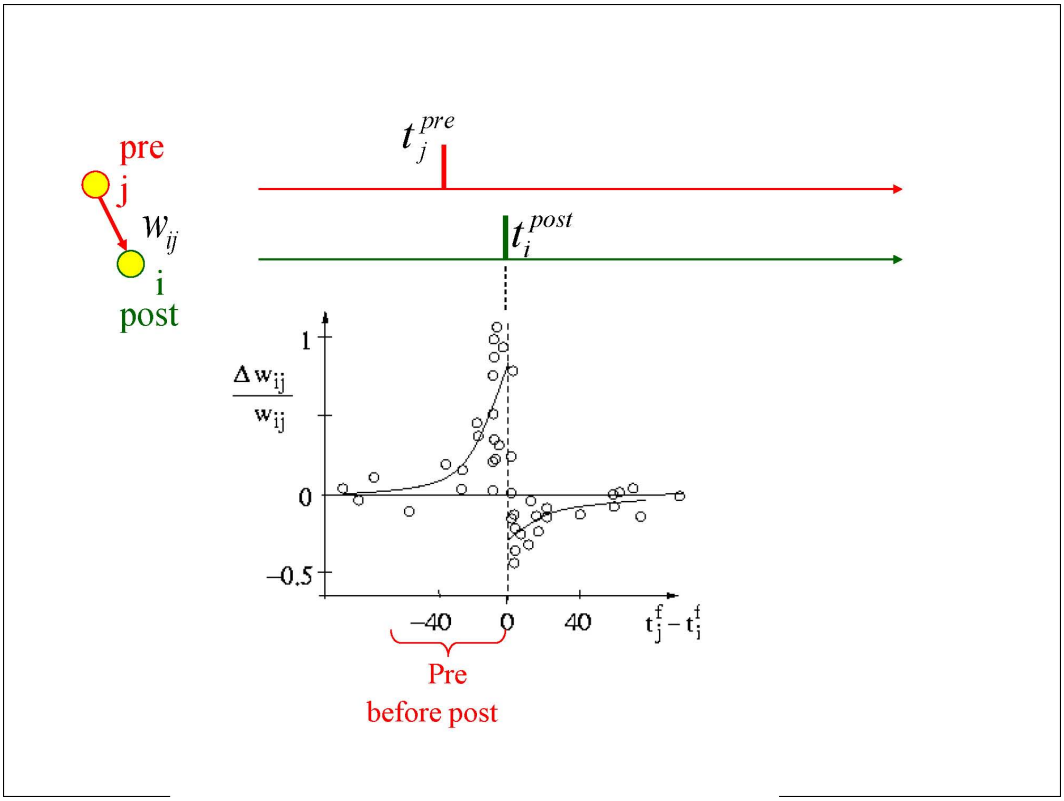

Figure 7: STDP learning rule example. The weight update of a synaptic connection between two neurons is a function of the time between firings of both neurons. ("Spike-timing dependent plasticity", 2021)

Finally, the same way neuromodulators (e.g. *dopamine*), affect the probability of spike, they also influence how and even *if* STDP learning takes place. Modulators are believed to be associated with various aspects of cognitive functions, such as reward, punishment, surprise, novelty, attention, etc. They generally influence collections of neurons at once, over longer time scales compared to that of action potentials. The combination of pre- and post-synaptic timings with modulators gives rise to the term *three-factor learning rule* (Kuśmierz, Isomura, & Toyoizumi, 2017) (Gerstner et al., 2018), illustrated in figure 8. Neuromodulation is prevalent in the brain, and are fundamental to support its various functions.

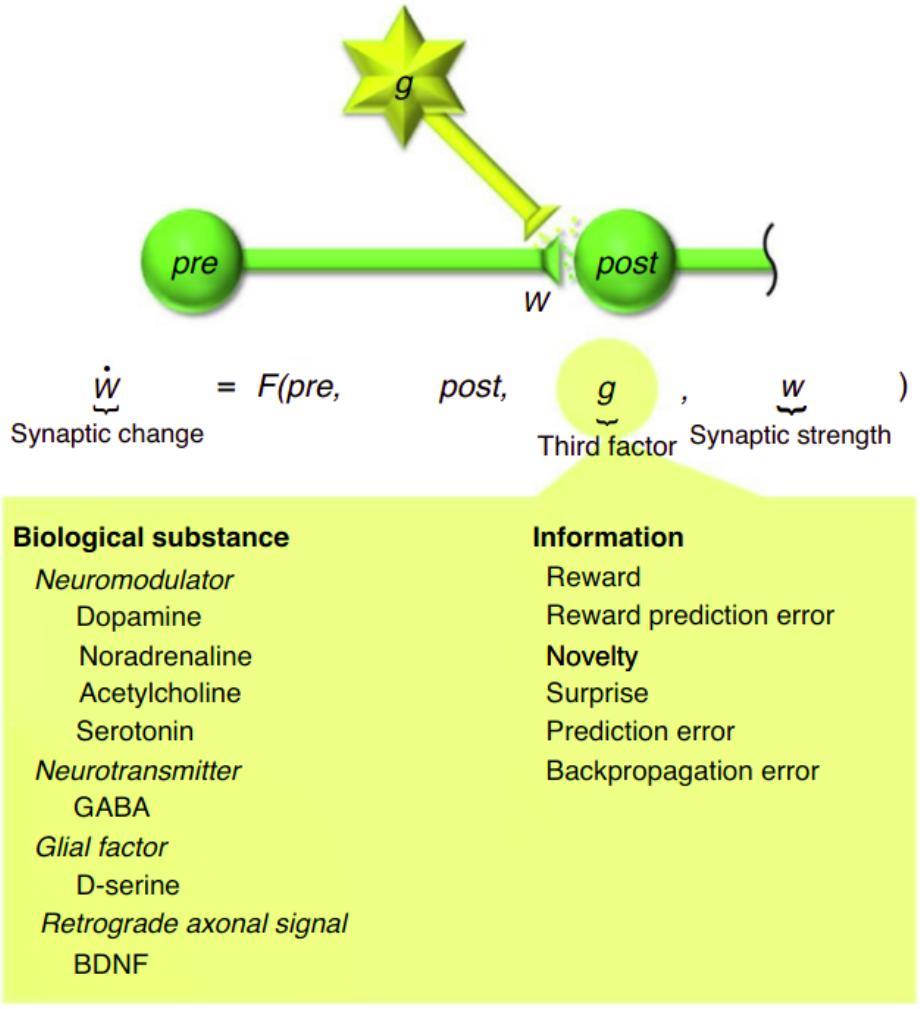

Figure 8: Three-factor learning rule, composed of the combination of pre- and -post-synaptic spike timings, and neuromodulators. (Kuśmierz, Isomura, & Toyoizumi, 2017)

2.2.2 Spiking Neural Networks and Neuromorphic Hardware

The functions described on the previous section add many degrees of freedom, as well as limitations to biological networks, which artificial networks simply do not have. For

instance, spikes are capable of transmitting information much more efficiently, in terms of energy usage, and their temporal nature arguably makes them more suitable for handling temporal data (processing image sequences, outputting motor control, etc). On the other hand, biological neurons only have access to information immediately available to them, and the mathematical operations performed are relatively simple operations such as additions of incoming spikes and multiplication by their synaptic strength.

An interesting and increasingly popular model of biologically inspired ANNs are Spiking Neural Networks (SNNs) Tavanaei et al. (2019). Also termed the *third generation* of artificial networks, SNNs incorporate the spiking nature of information that biological networks possess.

Because SNNs are *more* general than standard deep learning, their most immediate application, and currently the most popular one, is to run pre-trained deep neural network models. This is interesting because recently introduced neuromorphic hardware can emulate spiking networks, and thus are able to take advantage of the highly-parallelizable nature of biologically inspired neurons, making them extremely energy efficient. True North Hsu (2014), for instance, is a neuromorphic chip from IBM, and possesses a power consumption density 10.000 times lower than a traditional general purpose modern CPU. But energy efficiency is not the only advantage: while standard deep networks tend to take longer to process data in larger network architectures, it has been shown that, to some extent, neuromorphic hardware is capable of constant inference time, regardless of the architecture size Blouw et al. (2019).

In order to perform on-chip learning, more recent neuromorphic hardware such as Loihi supports STDP-like learning rules and also, to a certain extent, neuronal modulation (Rajendran et al., 2019) (Tang et al., 2019). As this kind of hardware becomes more powerful and widespread, the goal then is to explore and develop learning algorithms that make use of such mechanisms and so would be implementable on such platform. Such learning algorithms would, as a corollary, be a class biologically plausible algorithms.

2.3 Biological Algorithms and Artificial Intelligence

2.3.1 Issues with Backpropagation in the Brain

So far, the fundamentals of backpropagation, i.e deep neural networks, and the main principles of biological networks, i.e spiking neurons and neuromodulators, have been presented. The question to be asked now is: could those two be related, or are they completely different approaches of learning?

At first sight, the answer is no. According to (Lillicrap et al., 2020), there are four main issues that causes backpropagation to be biologically incompatible:

Complex Computation: Backpropagation is, mathematically, a relatively complicated operation. The weight update of a single connection requires information of neuron activation of all subsequent neurons in all subsequent layers. Moreover, the calculation of weight updates require access to the actual values of synaptic weights in these subsequent layers. This is troubling, given that real neurons only have knowledge of synaptic activity from directly connected neurons through its dendrites.

Synaptic Symmetry: The equations of backpropagation require that feedback connections delivering error values be equal to the forward connections. This issue is known as the *weight transport problem*, and there is no biological evidence for such connection symmetry.

Signed Error: Error signals carried through synaptic connections must be either positive or negative. Real synaptic connections, however, have a fixed sign, which is either excitatory or inhibitory.

Feedback Connections Influence Behaviour: In backpropagation, only feed-forward connections are used during inference, and feedback connections would only be used for error delivery during learning. However, biological evidence suggests that such feedback connections are actively driving response on the network. This hints that those feedback connections have a different role in information processing than in usual backpropagation networks.

2.3.2 Alternative Learning Algorithms

Despite these issues, algorithms of learning that explain how biological networks *could* perform a similar operation to backpropagation have been proposed. They make use, for instance, of computations using *simple additions and subtractions*, which can be performed with excitatory and inhibitory connections; *localized information processing*, such that a neuron uses only data which is currently available, as opposed to in standard backpropagation; *STDP-like* weight updates, such that connection weights are automatically updated according to pre- and post-synaptic neuron spike timings, and so on. Table 1 shows a comparison of four such algorithms.

	Temporal-error model		Explicit-error model	
	Contrastive learning	Continuous update	Predictive coding	Dendritic error
Control signal	Required	Required	Not required	Not required
Error encoded in	Difference in activity between separate phases	Rate of change of activity	Activity of specialised neurons	Apical dendrites of pyramidal neurons
MNIST performance	2–3	–	1.7	1.96

Table 1: Comparison of different implementations of backpropagation approximations using viable biological operations. *MNIST performance* refers to the test accuracy performance on the handwritten digit dataset for image classification LeCun and Cortes (2010). The full table is found at Whittington and Bogacz (2019)

Temporal-error models [*contrastive learning* (O'Reilly, 1996) and *continuous update* (Bengio et al., 2017)] have a simpler structure and are considerably easier to implement. Also, the connection weight updates are in accordance with the Hebbian Plasticity from biological neurons. This weight update occurs in two temporally distinct phases: the prediction phase,

when the output is defined by the network activity; and the target phase, when the output neurons are set to the desired value (the target value). Both phases concomitantly update neuron connection weights until the network learns the correct output.

One issue with this approach is that the entire network needs to be synchronized so that all neurons know which phase is currently in place. The would require a global signal, and there is no clear evidence of one in the brain.

Conversely, *Explicit-error models* require extra neurons that account for error minimization, making it structurally more complex. However, this class of algorithms do not require multi-phase learning and thus can function without the need for a global signal. *Predictive Coding* is one such algorithm, and will be explored further in this work.

2.4 Predictive Coding

2.4.1 Introduction

Predictive Coding (Rao & Ballard, 1999) (Friston, 2005) (Bogacz, 2017) is, fundamentally, a theory of how perception is implemented by the brain. It is a hierarchical, generative model of perception based on empirical Bayes, where an internal representation of the external world is continuously adapted such that sensory input data is better explained by the inferred, hidden causes of such input.

A sensory input u can be associated to a given cause v through a non-linear function g and a set of model parameters θ, as described on the equation 13.

$$u = g(v, \theta) \tag{13}$$

Causes can be thought of as the reasons underlying the sensory information. For instance, a sensory input u could be the image of a car, and the *causes* could be the car model, color, position, time of day, etc. The causes are naturally converted into sensory information by the "rendering engine" of the external world. The brain, on the other hand, does the inverse process, inferring the causes from the sensory data. This process of inferring underlying causes from sensory data is called *recognition*.

In the Predictive Coding formulation, the problem of recognition is approached in a probabilistic (Baysian) fashion. First, the probability distribution of a sensory input data u given a cause v is normal around a given mean $g(v, \theta)$, from equation 13, with variance Σ_u, as shown in equation 14.

$$p(u|v) = \mathcal{N}(u; g(v, \theta); \Sigma_u) \tag{14}$$

Note that the normal distribution of a variable x with mean μ and variance Σ is defined as:

$$\mathcal{N}(x; \mu; \Sigma) = \frac{1}{\sqrt{2\pi\Sigma}} \exp\left(-\frac{(x-\mu)^2}{2\Sigma}\right) \tag{15}$$

Similarly, the probability of the cause is also normally distributed, according to a prior expectation v_p and a prior variance Σ_p (equation 16).

$$p(v) = \mathcal{N}(v; v_p; \Sigma_p) \tag{16}$$

The probability of the cause, given the input sensory information is defined by the Bayes Theorem (equation 17):

$$p(v|u) = \frac{p(v)p(u|v)}{p(u)} \tag{17}$$

The recognition problem can then be divided into two parts: [1] finding the v that, given the model parameters θ and sensory input, maximizes the probability described by equation 17. And [2] find the model parameters θ that better describe the relation between various sensory data and the inferred causes. The former is defined as *inference*, and the latter is *learning*. Note that both imply a maximization of equation 17.

A variable F is defined as the logarithm of the numerator of the right-hand side of equation 17. For the case where v and u are scalars, F is given by equation 18.

$$F = ln(\mathcal{N}(u; g(v, \theta); \Sigma_u)) + ln(\mathcal{N}(v; v_p; \Sigma_p)) \tag{18}$$

This variable is also called *negative of the free energy*. Now, substituting equation 15 into 18, and assuming constant variances for simplicity, yields equation 19 (C represents the constant terms).

$$F = -\frac{1}{2}\left(\frac{(v - v_p)^2}{\Sigma_p} + \frac{(u - g(v, \theta))^2}{\Sigma_u}\right) + C \tag{19}$$

Therefore, *inference* is the maximization of F with respect to v, and *learning* is the maximization of F with respect to the parameter θ. This can be realized through *gradient ascent* on both variables. Assuming the function $g(\cdot)$ includes a non-linear activation function $h(\cdot)$ and has the form $g(v, \theta) = \theta h(v)$, the partial derivatives of F with respect to the variables of interest are given by equations 20 and 21.

$$\dot{v} = \frac{\partial F}{\partial v} = \frac{v_p - v}{\Sigma_p} + \frac{u - g(v, \theta)}{\Sigma_u}\theta h'(v) \tag{20}$$

$$\dot{\theta} = \frac{\partial F}{\partial \theta} = \frac{u - g(v, \theta)}{\Sigma_u}h(v) \tag{21}$$

2.4.2 Neural Circuit

It can be shown that it is possible to implement the model proposed so far in a biological network. The sensor input u and cause v can be represented as neuronal activity (i.e, the neuron membrane potential), while the learned parameter θ represents the synaptic weight. Finally, the difference terms in equations 20 and 21 are set as variables ε_p and ε_u, as shown in equations 22 and 23.

$$\varepsilon_p = \frac{v - v_p}{\Sigma_p} \tag{22}$$

$$\varepsilon_u = \frac{u - g(v, \theta)}{\Sigma_u} \tag{23}$$

These difference terms may also be represented as *error neurons*, whose value are updated according to equations 24 and 25.

$$\dot{\varepsilon_p} = v - v_p - \Sigma_p \varepsilon_p \tag{24}$$

$$\dot{\varepsilon_u} = u - \theta h(v) - \Sigma_u \varepsilon_u \tag{25}$$

Note that equations 24 and 25 converge to the values given by 22 and 23, when the updated amount converges to 0. Figure 9 depicts a possible neural implementation of the algorithm given.

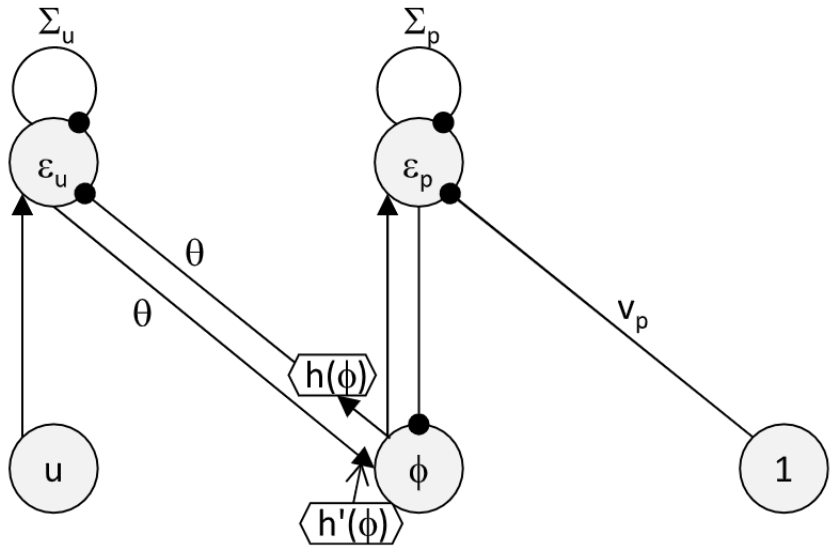

Figure 9: A possible biological network implementation of a simple predictive coding algorithm. Arrows at the end of connections indicate an excitatory connection, while circles indicate an inhibitory connection (the value is multiplied by -1). The model consists of a single input neuron, u, and a single output neuron, ϕ. Neurons ε are called *error neurons*, and encode the difference between the actual neuron value and the expected value. (Bogacz, 2017)

The biological plausibility of a neuronal model described by equations 20 and 21 stems from two observations:

- All information needed to calculate the values of neurons is available locally, using simple operations of addition and multiplication (and is thus implementable using a simple network, such as shown in picture 9).
- The synaptic weight θ is updated according to $\dot{\theta} = \varepsilon_u h(v)$, which is in accordance with the STDP rule, discussed in section 2.2.1.

2.4.3 Extending the Model

So far, the model given is extremely simple, consisting of a single input, a single output, in a single layer. Biological networks in the brain, however, are much more complex. For instance, the Visual Cortex is divided into different areas (V1, V2, V3, etc), representing different stages of information processing. This gives it a hierarchical structure, in which some processing areas operate *closer* to the raw sensory input, i.e, lower areas, which then go up through the processing pipeline.

In the case of Predictive Coding, hierarchy can be achieved by connecting layers, using the output of one as the input of the next (Whittington & Bogacz, 2017). In the case of equation 13, this could be represented as the set of equations indicated by 26.

$$\begin{aligned} u &= g(v_2, \theta_1) \\ v_2 &= g(v_3, \theta_2) \\ v_3 &= g(v_4, \theta_3) \\ &... \end{aligned} \tag{26}$$

Where u is the activity of the first layer (sensory input data).

Furthermore, each variable $v_1, v_2, ...$ may be considered a vector with variable dimension (the notation $\bar{v}_i$ denotes a vector representing activations of the neurons from layer layer i). In this case, each parameter θ becomes a matrix $\mathbf{\Theta_l} \in \mathbf{R}^{n \times m}$, being n the size of the layer $l-1$ and m the size of the layer l. One possible definition for the function $g(\cdot)$ in the multi-variable case given by equation 27 (Whittington & Bogacz, 2017). $\theta^l_{i,j}$ represents the element of the i-th row and j-th column of the parameter matrix $\mathbf{\Theta_l}$ (note the layer index becomes a superscript when the neuron indices are shown). The extra parameter b_i^{l-1} was added and it gives one more degree of freedom to the model, similar to equation 3, from the backpropagation algorithm.

$$\mu_i^l = g(\bar{v}_{l-1}, \mathbf{\Theta_{l-1}}) = \sum_j^n \theta_{i,j}^{l-1} h(v_j^{l-1}) + b_i^{l-1} \tag{27}$$

With this new structure, the probability distribution of the activation of a given neuron i of layer l is therefore dependent on the activations of all neurons from the previous layer $l-1$, as per equation 28.

$$p(v_i^l | \bar{v}_{l-1}) = \mathcal{N}(v_i^l; \mu_i^l; \Sigma_i^l) \tag{28}$$

The negative of the free energy, then, is given by equation 29.

$$F = \sum_{l=1}^{l_{max}-1} \sum_{i=1}^{n_{(l)}} \ln(p(v_i^l | \bar{v}_{l+1})) \tag{29}$$

And the error neurons can be defined as equation 30.

$$\varepsilon_i^l = \frac{v_i^l - \mu_i^l}{\Sigma_i^l} \tag{30}$$

With this, the *inference* equation (originally equation 20, for the simple model) becomes equation 31.

$$\dot{v}_i^l = -\varepsilon_i^l + \sum_{j=1}^{n(l+1)} \varepsilon_j^{l+1} \theta_{j,i}^l h'(v_i^l) \tag{31}$$

Being $n(l)$ the size of the vector $\bar{v}_l$.

After a finite number of iterations of equation 31, the network reaches a *steady state*, i.e the neuronal activity of all the neurons stop changing. If no neurons are being forced (clamped) to some value, all error neurons should converge to zero, and the network is said to be *relaxed*. Otherwise, the error neurons will stabilize on a non-zero value, and the network should adapt its parameters to better adjust to the forced values. This is done in the *learning* step, as described by 32.

$$\dot{\theta}_{j,i}^l = \varepsilon_j^l \cdot h(v_i^{l+1}) \tag{32}$$

2.5 Convolutional Networks and Vision

A popular class of methods for performing visual processing in artificial neural networks are *Convolutional Neural Networks* (CNNs). Historically, it was inspired by biological observations of how brains process visual stimuli. Hubel and Wiesel (2020) found that neurons in the visual cortex of monkeys and cats tended to respond only to a small region of the visual field, denoted receptive field. Later (Hubel & Wiesel, 1968), it was observed that some of those neurons (the "complex cells") had a larger receptive field than others, and they responded to edges inside their field regardless of their exact positioning. This gives the notion of a mechanism of visual processing that works on regions of an image and also shows some form of position invariance on that processing. Inspired by this and the work of many others, LeCun et al. (1998) proposed what is considered today the modern concept of a *convolutional neural network*.

Such networks revolve around the concept of a *convolutional filter*. Each patch of the image is processed by such *filter*, which is a square matrix $l \times l$, that is convoluted with the 2D image, generating a new image, as shown in figure 10 (in this case, the *convolutional filter* is a matrix with size $l = 3$). The result is another image, with the same dimensions or slightly smaller dimensions compared to the original image, depending on the *padding* method utilized (for instance, filters centered on pixels at the edges of the image may be discarded or convoluted with zeros n regions outside the image). Thus, different filters will respond differently to different features on the image, and this is the basis for performing feature detection, as long as the filter parameters (i.e., the filter matrices' values) are representative of interesting features to be detected.

Given that the output of a convolution operation on an image is itself another image, it is possible to apply the same operation again. From this, a layered topology emerges, which is somewhat comparable to the hierarchical structure found on the visual cortex of mammals (Van Essen & Maunsell, 1983). Furthermore, a sub-sampling operation between

convolutional layers is often used to significantly reduce the required computation with similar results. An example of such architecture is shown in figure 11.

Early versions of CNNs, such as Denker et al. (1989), had such filter parameters to be manually set, so that they would detect given features, such as edges or corners, a cross, etc. This is laborious, and filters from subsequent layers are non-trivial to estimate. A large contribution of LeCun et al. (1998) is the proposal of a method to automatically compute such filter parameters, through backpropagation (Section 2.1).

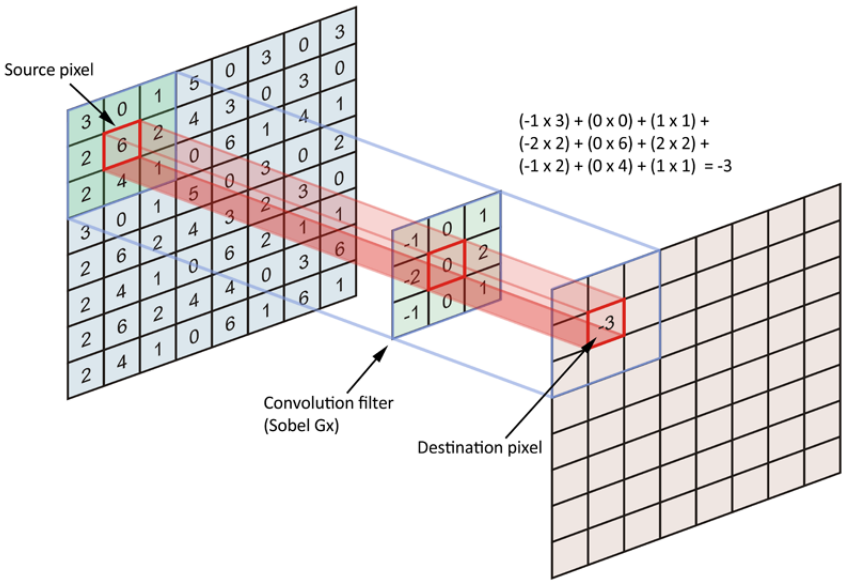

Figure 10: A Convolutional Network Filter. The result of applying a convolution operation to an image is another image. Source: Nagapawan, Prakash, and Kanagachidambaresan, 2021

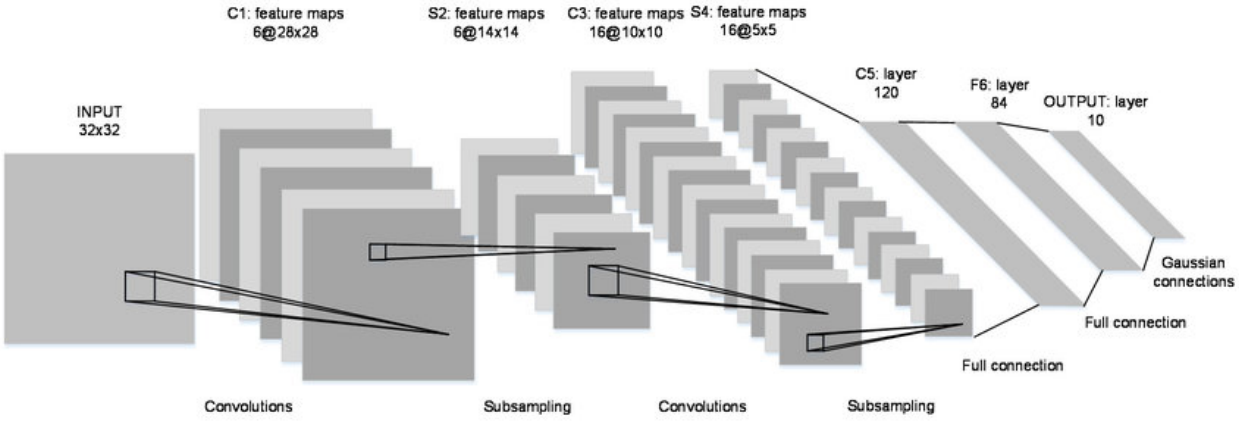

Figure 11: Topology of Le-Net-5, the convolutional network proposed by LeCun et al. (1998). *Feature maps* refer to the output of the convolution operation. Their size are reduced by half once the pooling operation is applied. Each different *convolutional filter* results in a single *feature map*, in any given layer. Therefore, the first and second convolutional layers correspond to 6 and 16 filters, respectively.

3 Methodology

3.1 Supervised Learning for Predictive Coding

Predictive Coding can be used in a supervised learning fashion as shown in the pseudo-code in algorithm 1. First, the data input is assigned to the first layer, while the last layer is is clamped to the desired output (e.g, the label). Employing equations 30 and 31, its neurons reach a steady state that best suits the input/output pair given. At this point, however, error neurons are non-zero, and indicate that there is a discrepancy between the predicted neuron activation and the actual neuron activation for each layer. This discrepancy arises due to the fact that the output layer is in a forced state, and as soon as it is released (i.e, relaxed), it will converge to the predicted value.

Learning then means updating the connection weights of each layer, so that the predicted neuron activity matches the desired activity. This is done via equation 32. After learning is performed, the discrepancy between predicted and actual neuronal activity should decrease, hence lowering the activity of error neurons.

Algorithm 1: Supervised Predictive Coding

```
foreach epoch do:
    foreach (x_in, x_out) do:
    v̄_0 ← x_in
    v̄_L ← x_out
    while neurons not converged:
      inference (equations 30 and 31)
    end
    update weights (equation 32)
end
```

3.2 Optimized Predictive Coding

3.2.1 Matrix form

The pseudo-code shown in algorithm 1, although implementable in a biological network, is relatively inefficient for computation using typical computers. Here it is shown how the predictive coding equations are implemented in matrix form, and how batch learning can be achieved. This makes it possible to simulate Predictive Coding for larger datasets (e.g, image datasets).

As mentioned before, it is possible to consider each neuron of layer l as a column vector $\bar{v}_l \in \mathbf{R}^{n\times 1}$, the connection weights as a matrix $\mathbf{\Theta}_l \in \mathbf{R}^{m\times n}$ and the connection biases as a column vector $\bar{b}_l \in \mathbf{R}^{m\times 1}$, being n the number of neurons in layer l, and m the number of neurons in layer $l+1$. In this way, equation 27 may be calculated in a matrix form, given by equation 33.

$$\bar{\mu}_l = \mathbf{\Theta}_{l-1} h(\bar{v}_{l-1}) + \bar{b}_{l-1} \tag{33}$$

The value of the error neuron (equation 30), assuming variance of each neuron equal to 1, becomes equation 34.

$$\bar{\varepsilon}_l = \bar{v}_l - \bar{\mu}_l \tag{34}$$

The inference equation for the neuronal activation (equation 31) can be written as equation 35, where $\odot$ denotes the *Hadamard product*.

$$\dot{\bar{v}}_l = -\bar{\varepsilon}_l + \mathbf{\Theta}_l^T \bar{\varepsilon}_{l+1} \odot h'(\bar{v}_l) \tag{35}$$

The weight update from equation 32 can be written according to equations 36 and 37.

$$\dot{\mathbf{\Theta}}_l = \bar{\varepsilon}_{l+1}\, h(\bar{v}_l)^T \tag{36}$$

$$\dot{\bar{b}}_l = \bar{\varepsilon}_{l+1} \tag{37}$$

3.2.2 Mini-batch calculation

The Predictive Coding in matrix form from equations 33 through can easily be modified to work in mini-batches, just as in backpropagation, which greatly improves performance. This can be done by extending the column vectors $\bar{v}_l$ and $\bar{\varepsilon}_l$ into multiple columns, pertaining to different input and output pairs, given by equations 38 and 39, where k is the mini-batch size.

$$\mathbf{v}_l = [\bar{v}_{1l}|\bar{v}_{2l}|...|\bar{v}_{kl}] \tag{38}$$

$$\mathbf{E}_l = [\bar{\varepsilon}_{1l}|\bar{\varepsilon}_{2l}|...|\bar{\varepsilon}_{kl}] \tag{39}$$

In this case, equations 33 to 35 remain unaltered. Equation 36 must be scaled by a factor of $1/k$, and $\dot{\bar{b}}_l$ on equation 37 becomes the average of $\bar{\varepsilon}_{l+1}$ over the columns.

3.3 Bottleneck Features Learning

In the context of Predictive Coding, image classification is typically carried using a fully connected network, as shown in figure 12. The image is flattened into a column vector, which is then fed into the network. Although this can yield satisfactory results, information about relative spatial positioning of pixels is lost. In other words, spatial features such as corners, lines, crosses, etc, cannot be explicitly detected, which is not in accordance with biological evidence (Elder & Sachs, 2004).

The proposed method in this work is to use features extracted from a pre-trained convolutional network, and use those as the input to the predictive coding network. This approach falls under the umbrella of transfer learning (Hussain, Bird, & Faria, 2018), and does not require a complex implementation to integrate convolutional layers into the existing Predictive Coding framework. Given the outstanding performance of Convolutional Networks for image processing tasks on backpropagation trained networks (Khan et al., 2020), it is reasonable to assume Predictive Coding networks could also be improved using such tool.

In this work, *VGG-16* (Simonyan & Zisserman, 2014) was used to extract such features. This is a popular convolutional network model, trained on millions of images from the *ImageNet* dataset (Deng et al., 2009). Figure 13 shows the network architecture of the network.

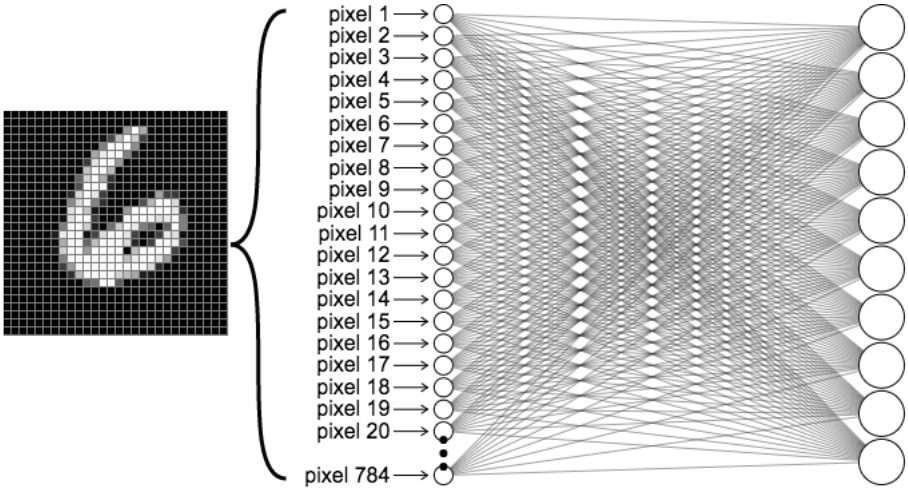

Figure 12: Example of an artificial neural network architecture ("Looking inside neural nets", 2021)

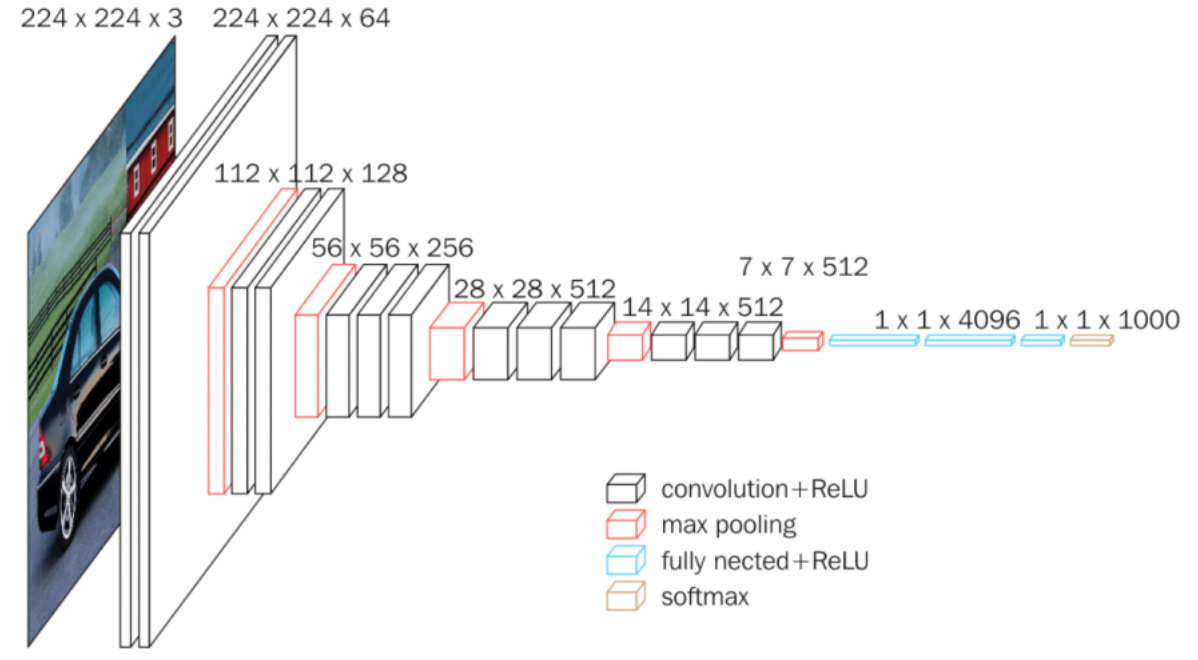

Figure 13: VGG-16 architecture ("VGG16 – Convolutional Network for Classification and Detection", 2021)

4 Results

4.1 Experiments

The proposed methodology was tested on image datasets for the task of image classification. Such datasets contain multiple images, each with a certain label, and the goal is to predict the correct label for that given image. More specifically, this is a multi-class single-label image recognition task (Dong, Zhu, & Gong, 2019), where multiple classes are present in the dataset, but only one correct label may be assigned to each image.

Apart from being tested using the proposed methodology, all experiments were conducted using the original *Predictive Coding* algorithm, and also using backpropagation-based neural network, for reference. The activation function $h(\cdot)$ from equation 36 for predictive coding and $\sigma(\cdot)$ from equation 1 for backpropagation, can be considered equivalent. Therefore the same function was used on all experiments (the *sigmoid function*), except for the last layer, where *softmax* is more suited for backpropagation. The weights were updated according to the widely used *Adam* optimizer algorithm (Kingma & Ba, 2014). On the backpropagation network, the loss function used was *categorical crossentropy* ("Tensorflow API: Categorical Crossentropy", 2021). Finally, all experiments were carried on a CPU (Intel i7, 7th Generation).

The primary metric employed to evaluate performance is *classification accuracy* ("Classification: Accuracy", 2021). It is given by equation 40, where N_{corr} is the number of correctly classified samples, and N_{tot} is the total number of samples. A secondary metric shown is the loss. It is calculated as the root mean squared error (RMS error) between the prediction vector and the correct label vector.

$$Acc(\%) = \frac{N_{corr}}{N_{tot}} \times 100\% \tag{40}$$

Finally, no data augmentation technique was employed. This could certainly improve the performance of the networks, but since the networks are being compared with backpropagation, it should suffice as a reference measure of performance.

4.2 MNIST Dataset

The *Predictive Coding* implementation given, and the proposed extension using bottleneck features were tested initially on the *MNIST* dataset (LeCun & Cortes, 2010). It is arguably the most popular dataset used in compute vision and machine learning tasks. It contains around 60,000 handwritten digit images and their corresponding labels (the corresponding number). A sample of this dataset is shown in figure 14.

The architecture of the networks, i.e., the number of neurons per layer, is: *Input Size, 500, 500, 10*, where every layer is fully connected to the next one (i.e., each neuron from layer l has one connection to each neuron on layer $l+1$). This is the same architecture used on the work of Whittington and Bogacz (2017). The *batch size* is 16 for all experiments, and the learning rate was fixed at 0.001. The number of *maximum inference steps* is set to 40 for the *Predictive Coding* networks.

The experiment results can be seen on figures 15 and 16. They show the accuracy and loss metrics across the epochs for all networks. Figure 16 shows the result of training using only a fraction of the dataset (20% of available data). This value was selected arbitrarily, but it demonstrates the ability of the networks to learn representations when less information is present.

Figure 14: Samples from the *MNIST* dataset (LeCun & Cortes, 2010). Each image corresponds to a handwritten digit from 0 to 9.

Network	**Training Time**
PC	217 sec
PC with Features	182 sec
Backpropagation	35 sec

Table 2: Time to train a single epoch on the *MNIST* dataset, on 100% of the training data. *PC* refers to the standard *Predictive Coding* network. Training was performed on a CPU *Intel i7, 7th Generation*

.

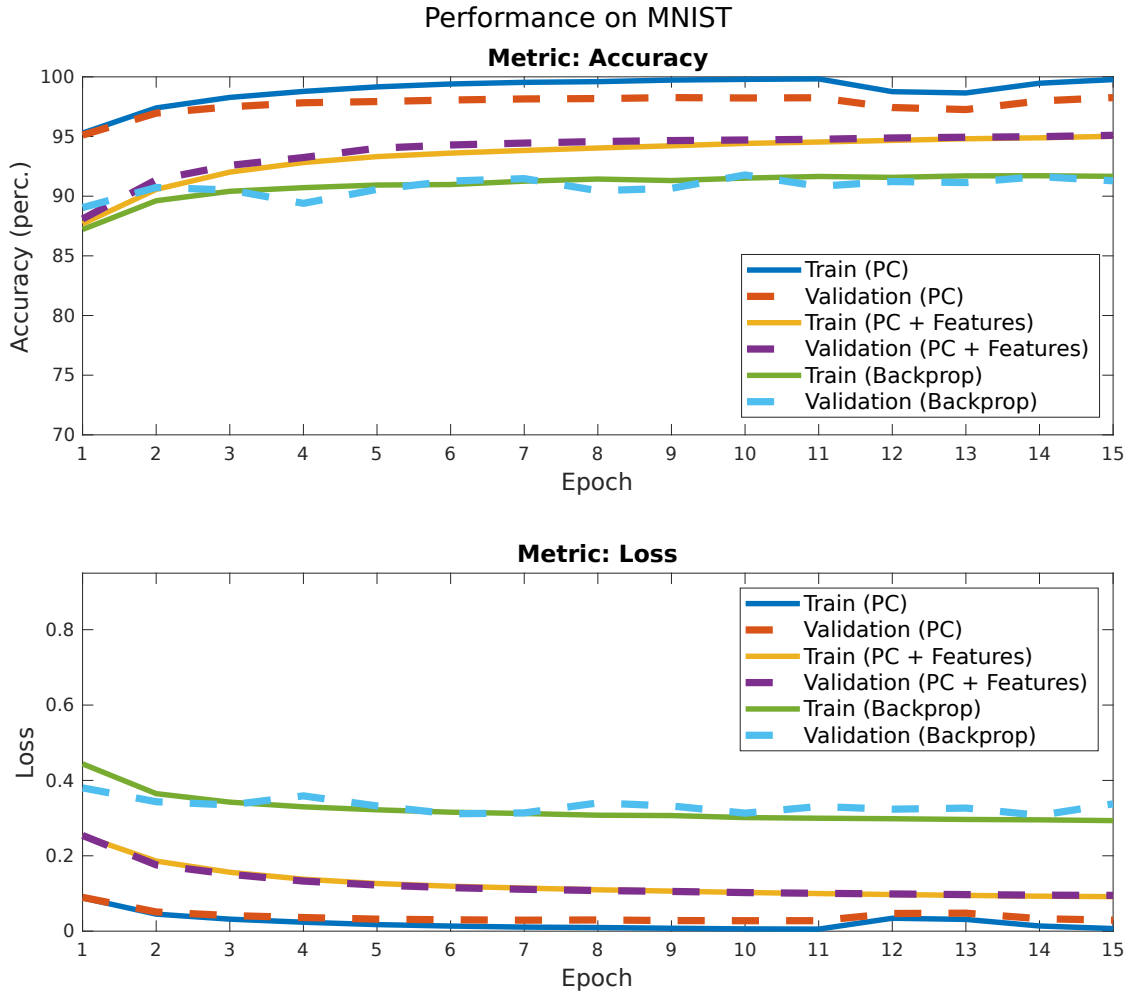

Figure 15: Experiments results for the discussed algorithms. (PC) refers to the standard predictive coding algorithm, (PC + Bottleneck Features) is the proposed method, (Backprop) is the neural network trained using backpropagation. Solid lines indicate the training results, and dashed lines show the validation results. The horizontal axis represent each training epoch, and the metrics shown were discussed in section 4.1: *Accuracy* indicates the percentage of samples that were correctly classified, and the *loss* shows the *Root Mean Squared Error* between the network prediction and the ground-truth.

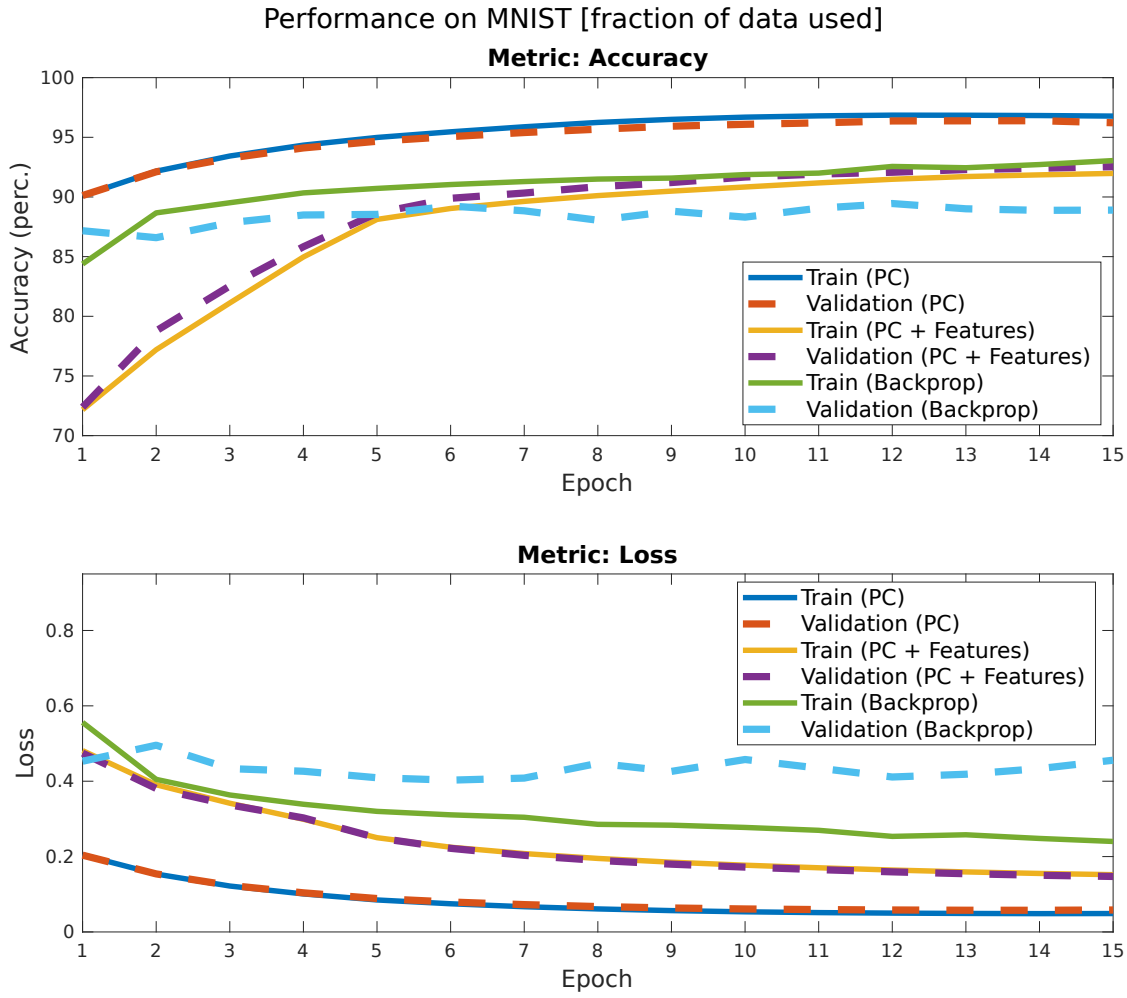

Figure 16: Same as figure 15, but using only 20% of available data for training.

4.3 ImageNet Dataset

The *Imagenet Project* (Deng et al., 2009) is a collection of datasets used for computer vision tasks. For this section, the "Imagenet32x32" (2021) dataset was used, containing 1281167 colored images pertaining to 1000 different classes, downscaled to the size 32×32. Figure 17 shows the distribution of the images of each class and figure 18 shows samples from the dataset.

The experiments in this section were performed using only a fraction of those classes (20 classes were considered), which correspond to about 25,600 images.

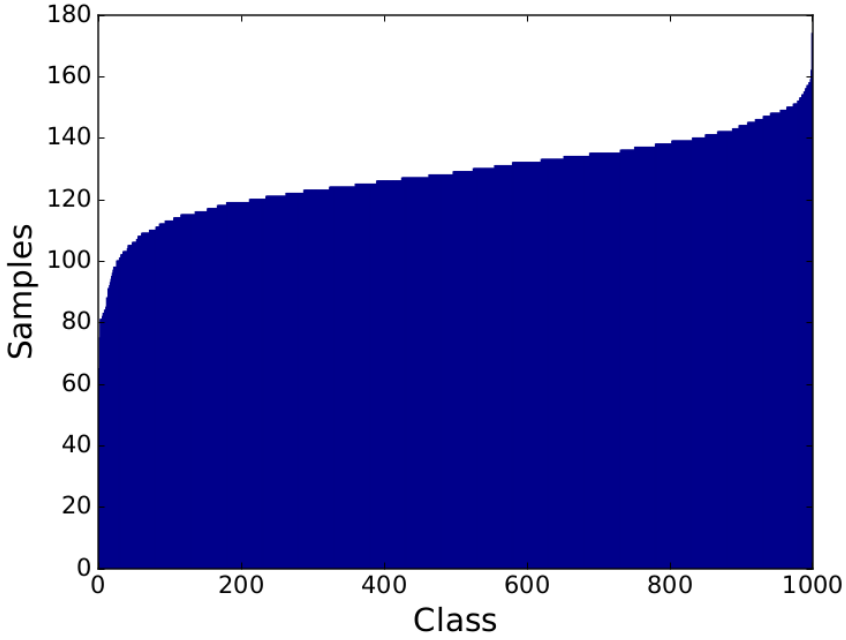

Figure 17: Class distribution of images from the "Imagenet32x32" (2021) dataset. It can be observed that the number of images of any single class can vary between 80 and 160 images.

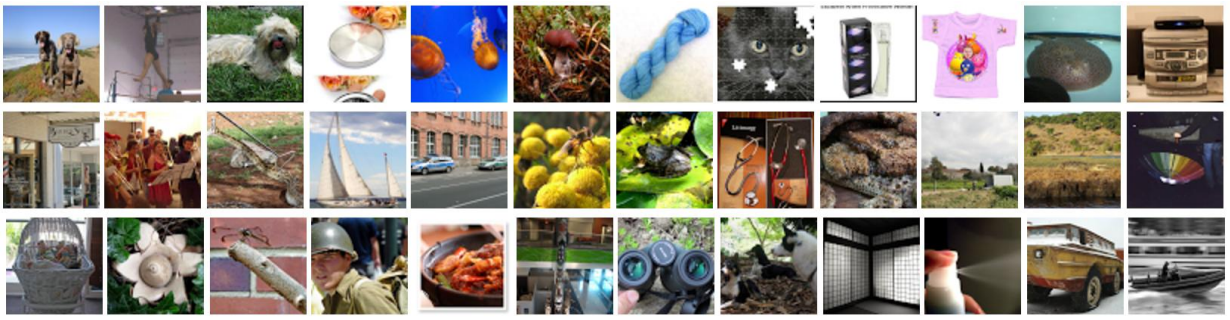

Figure 18: Samples from the "Imagenet32x32" (2021) dataset. It contains colored images from 1000 classes. Image source: Oord, Kalchbrenner, and Kavukcuoglu (n.d.)

The backpropagation training was done with a more well suited architecture for image classification, similar to VGG-16 (Simonyan & Zisserman, 2014), instead of only fully connected networks, as was done in section 4.2. This implies the use of *convolutional*, *pooling* and also *dropout* layers. The convolution layers used a *valid* padding type and 1×1 strides. The *max pooling* layers were set to size 2×2. The *dropout* layer after the fully connected layer was set to 30% rate, instead of 50%.

For the *Predictive Coding* networks, the same networks parameters as in Section 4.2 were used. The full description of the architecture of each network in this section is given as follows:

- **PC**: Same architecture as previous section.
- **CNN (backpropagation)**: 64 3×3 filters, max pooling, 128 3×3 filters, max pooling, 256 3×3 filters, 256 3×3 filters, max pooling, 512 3×3 filters, max pooling. Finally, a fully connected layer with 1024 neurons was used, with a dropout of 30%. This is similar to what was proposed by (Simonyan & Zisserman, 2014), and also a 1024-neuron fully connected layer at the end shows a reasonable performance for both *backpropagation* and the proposed method, as shown on table 4
- **PC + Features**: As means of comparison with the *CNN* network, only a single fully connected layer with 1024 neurons was used.

Table 3 shows the time taken to train one epoch of the proposed networks, using a CPU (as described in section 4.1). Figures 19 and 20 show the accuracy and loss (*Root Mean Squared Error*) metrics across 15 epochs of the networks, for 100% and 20% of the available training data, respectively.

Finally, a comparison between the proposed method (*PC+ Features*) and the backpropagation network for different architectures of the fully connected layer is given on table 4. The first row is a 2 layer fully connected with 500 neurons each, the second is a single fully connected layer with 500 neurons, and the third is a wider 1024 neurons single layer. For the backpropagation network, the convolutional layers retained the same structure as the one used in the previous experiment. The results shown correspond to the *validation accuracy* after 10 epochs of training.

Network	**Training Time**
PC	136 sec
PC with Features	55 sec
Backpropagation	123 sec

Table 3: Time to train a single epoch on the *Imagenet* dataset, on 100% of the training data. *PC* refers to the standard *Predictive Coding* network.

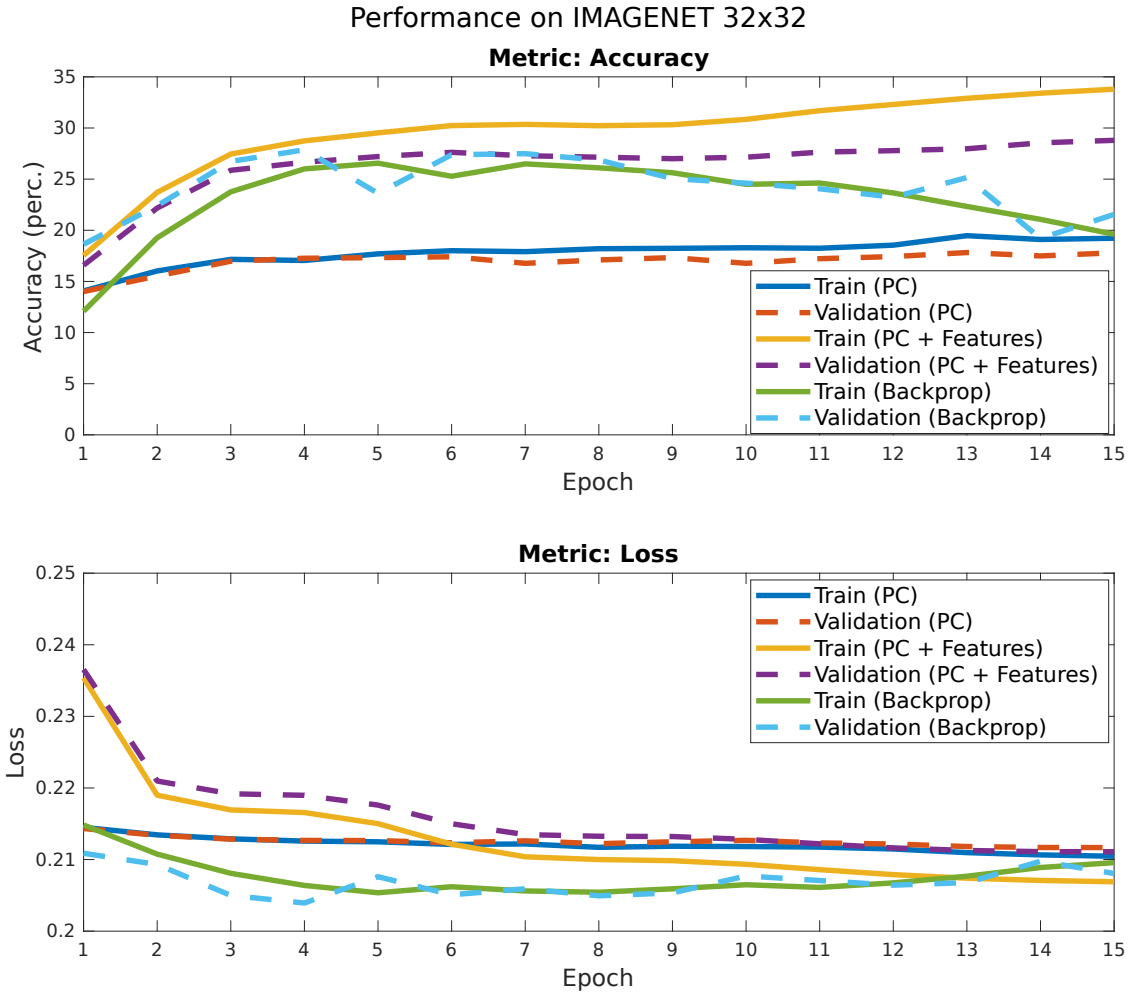

Figure 19: Experiment results for the discussed algorithms on the "Imagenet32x32" (2021) dataset. (PC) refers to the standard predictive coding algorithm, (PC + Bottleneck Features) is the proposed method, (Backprop) is the convolutional neural network (CNN) trained using backpropagation. Solid lines indicate the training results, and dashed lines show the validation results.

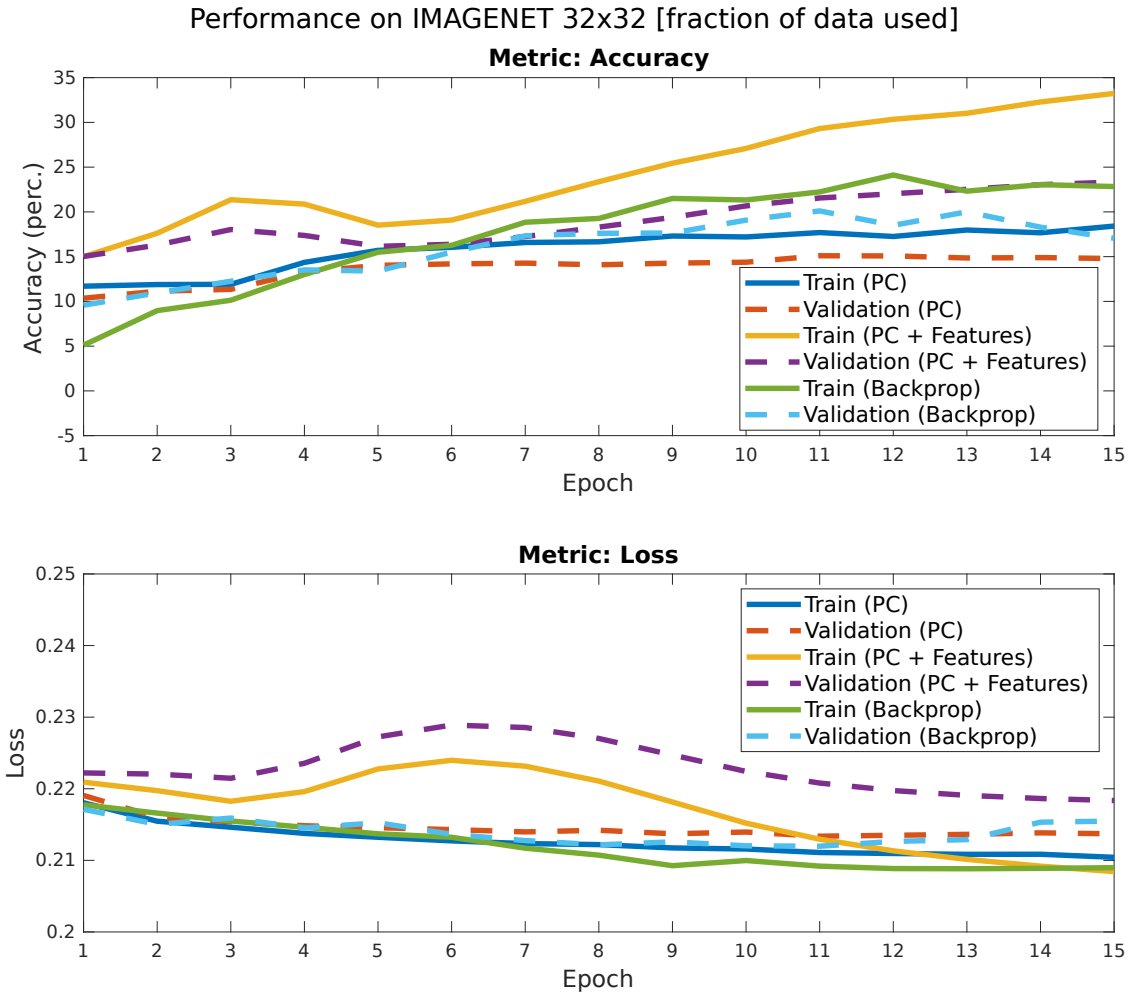

Figure 20: Same as figure 19, but using only 20% of available data for training.

	PC + Features	**CNN**
FC: 500 × 500 layers	26%	23%
FC: 500 layer	28%	18%
FC: 1024 layer	26%	24%

Table 4: Comparison of the proposed method (*PC + Features*) and a convolutional network trained using backpropagation (*CNN*), for different architectures, where *FC* denotes the number of neurons of the *Fully Connected* layers. The metric being compared is the highest *validation accuracy* of the networks, over 10 training epochs, using 50% of available training data.

4.4 Discussion

Considering, initially, the results for the MNIST dataset. Figure 15 shows that the standard Predictive Coding network (*PC*) yields the best results, at around 98% accuracy. This is close to the state of the art for this database ("Image Classification on MNIST", 2021). The proposed method (shown as *PC+Features*) is just under 95% accuracy, which is still satisfactory, and above backpropagation trained network, which stands at 90% after 15 epochs for both training and validation.

The fact that the accuracy was worse on the proposed method, compared to Predictive Coding, could be explained by the fact that VGG-16 was previously trained to extract features from the Imagenet dataset, which is significantly different from MNIST dataset (this is empirically evident, and can be observed on figures 14 and 18). Therefore, such bottleneck feature extractor would not be optimized for this type of data.

It is also important to point out that the network architecture selected for backpropagation is a simple, fully connected network, and is not well suited for image classification. The comparison is valid in the sense that all networks possess equivalent architectures (i.e., same number of neurons per layer).

The image classification benchmark for the downsampled Imagenet dataset (size 32×32) is listed at the website as 59% ("Imagenet32x32", 2021). Figure 19 shows that the best performing algorithm on this task was the proposed methodology (*PC + Features*), at 29% validation accuracy after 15 epochs, a significant improvement over the original Predictive Coding algorithm, that stood at 15% accuracy.

The backpropagation network used on this dataset was somewhat optimized for image classification, because it was equipped with convolutional, pooling and dropout layers, and followed an architecture similar to VGG-16. It still showed modest accuracy metrics, which peaked at 27% after 4 epochs, but then decreased to around 21% after 15 epochs. This performance degradation is likely not due to over fitting, since the train accuracy also went down.

Given table 3, it appears that the proposed method is the fastest in terms of training

time, but this is due to the fact that it used the simplest network architecture with the least amount of parameters, while the backpropagation network had the most. A better comparison for training times, however, is given by table 2, since all networks have the same size. It shows backpropagation is about 6 times faster than Predictive Coding. This, however, is likely due to the highly optimized implementation of the algorithm from Tensorflow, so it is hard to estimate how much of this difference comes from the nature of the algorithms themselves. But it would not be too far-fetched to suppose they have comparable complexity, due to their algorithmic similarity.

It is interesting to notice from the experiments shown in figures 16 and 20 that, with much less data available, the Predictive Coding networks (the original network and the proposed one) had significantly better results than backpropagation. For MNIST, using only 1/5 of available data, standard Predictive Coding achieved 95% validation accuracy, while backpropagation was about 87% (figure 16). On the Imagenet dataset, also using a fraction of the data, the proposed method yielded 24% of validation accuracy, while backpropagation only 16.5% (figure 20).

Finally, table 4 shows that the proposed method outperforms a typical convolutional network for image classification, after 10 epochs of training, for 3 different architectures. These results could also be different for different choices of hyper-parameters, and it is hard to draw a definitive conclusion without deeper analysis, which is out of the scope of this work. They do, however, give a positive outlook for Predictive Coding and the proposed methodology.

5 Conclusion and Future Work

This work tried to answer the question *"How could biological algorithms of learning be implemented?"*, by giving a broad overview of how learning is currently implemented in artificial networks (i.e., backpropagation), what mechanisms biological networks use to operate, and finally what alternative algorithms of learning are available that could in theory mimic backpropagation, subject to biological constraints. It is, of course, still out of grasp under the current state of knowledge how exactly learning happens in the brain, and so these algorithms are only the first steps to reaching this breakthrough.

Nevertheless, the implementation and experiments of the Predictive Coding framework have shown interesting results, compared to the well-stabilised backpropagation-based artificial networks. In all experiments, backpropagation networks were outperformed by the Predictive Coding networks, both when the network topology were equivalent (i.e., fully connected networks) and when an optimized topology for image classification, inspired by the VGG-16 architecture (Simonyan & Zisserman, 2014), was used for the backpropagation network. The proposed method also outperformed backpropagation when only a fraction of the training data was used.

Although these results suggest Predictive Coding could be a powerful tool for biologically plausible algorithm of learning, more experiments would need to be carried to assert its equivalence to backpropagation. More specifically, it would require experiments with larger datasets and deeper networks, over a larger number of epochs, with more hyper-parameter tuning.

Since *Predictive Coding* is a biologically plausible algorithm, it would be an interesting next step to this research to verify the feasibility of its implementation using Spiking Networks. This would then allow for on-chip learning for neuromorphic hardware, which are becoming increasingly popular, whose technological development is still in its infancy. This kind of application would place Predictive Coding networks, or another biologically plausible algorithm of learning, in a much more favorable position than backpropagation, since the latter cannot be used for in-chip learning on neuromorphic hardware.

In the case of this work, the model studied processed static data, that is, for image classification task, in which previous samples do not influence the response to current ones. But *Predictive Coding* can also account for sequential data, adding a temporal dimension to the model (Friston, 2005). This temporal component, when applied to *Predictive Coding* gives rise to a new term, denoted *Prospective Coding* (Brea et al., 2016), and is a promising field of research. Finally, another possible direction of future research would be to experiment with the addition of neuromodulators to the algorithm in question, and apply it for algorithms such as reinforcement learning, similar to the line of work of Urbanczik and Senn (2014).

References

Bengio, Yoshua et al. (2017). "STDP-compatible approximation of backpropagation in an energy-based model". In: *Neural computation* 29.3, pp. 555–577 (cit. on p. 16).

Blouw, Peter et al. (2019). "Benchmarking keyword spotting efficiency on neuromorphic hardware". In: *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, pp. 1–8 (cit. on p. 15).

Bogacz, Rafal (2017). "A tutorial on the free-energy framework for modelling perception and learning". In: *Journal of mathematical psychology* 76, pp. 198–211 (cit. on pp. 17, 19).

Brea, Johanni et al. (2016). "Prospective coding by spiking neurons". In: *PLoS computational biology* 12.6, e1005003 (cit. on p. 37).

Brown, Tom B et al. (2020). "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165* (cit. on p. 4).

Classification: Accuracy (2021). URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy (visited on 02/02/2021) (cit. on p. 27).

Deng, Jia et al. (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255 (cit. on pp. 4, 26, 31).

Denker, John S et al. (1989). "Neural network recognizer for hand-written zip code digits". In: *Advances in neural information processing systems*. Citeseer, pp. 323–331 (cit. on p. 22).

Dong, Qi, Xiatian Zhu, and Shaogang Gong (2019). "Single-label multi-class image classification by deep logistic regression". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01, pp. 3486–3493 (cit. on p. 27).

Elder, James H and Adam J Sachs (2004). "Psychophysical receptive fields of edge detection mechanisms". In: *Vision Research* 44.8, pp. 795–813 (cit. on p. 25).

Friston, Karl (2005). "A theory of cortical responses". In: *Philosophical transactions of the Royal Society B: Biological sciences* 360.1456, pp. 815–836 (cit. on pp. 17, 37).

Gerstner, Wulfram et al. (2018). "Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules". In: *Frontiers in neural circuits* 12, p. 53 (cit. on p. 14).

GPT-3 Power Consumption (2021). URL: https://www.theregister.com/2020/11/04/gpt3_carbon_footprint_estimate/ (visited on 02/02/2021) (cit. on p. 4).

Hebb, Donald Olding (2005). *The organization of behavior: A neuropsychological theory*. Psychology Press (cit. on p. 12).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780 (cit. on p. 4).

How many neuron types are there (2021). URL: https://jonlieffmd.com/blog/how-many-different-kinds-of-neurons-are-there?utm_content=bufferaffcf (visited on 02/02/2021) (cit. on p. 9).

Hsu, Jeremy (2014). "IBM's new brain [News]". In: *IEEE spectrum* 51.10, pp. 17–19 (cit. on p. 15).

Hubel, David H and Torsten N Wiesel (1968). "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of physiology* 195.1, pp. 215–243 (cit. on p. 21).

— (2020). "8. Receptive fields of single neurones in the cat's striate cortex". In: *Brain Physiology and Psychology*. University of California Press, pp. 129–150 (cit. on p. 21).

Hussain, Mahbub, Jordan J Bird, and Diego R Faria (2018). "A study on cnn transfer learning for image classification". In: *UK Workshop on computational Intelligence*. Springer, pp. 191–202 (cit. on p. 25).

Image Classification on MNIST (2021). URL: https://paperswithcode.com/sota/image-classification-on-mnist (visited on 02/02/2021) (cit. on p. 35).

Imagenet32x32 (2021). URL: https://patrykchrabaszcz.github.io/Imagenet32/ (visited on 02/02/2021) (cit. on pp. 31, 33, 35).

Khan, Asifullah et al. (2020). "A survey of the recent architectures of deep convolutional neural networks". In: *Artificial Intelligence Review* 53.8, pp. 5455–5516 (cit. on p. 25).

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (cit. on pp. 8, 27).

Kuśmierz, Łukasz, Takuya Isomura, and Taro Toyoizumi (2017). "Learning with three factors: modulating Hebbian plasticity with errors". In: *Current opinion in neurobiology* 46, pp. 170–177 (cit. on p. 14).

LeCun, Yann and Corinna Cortes (2010). "MNIST handwritten digit database". In: URL: http://yann.lecun.com/exdb/mnist/ (cit. on pp. 16, 27, 28).

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on pp. 21–23).

Lillicrap, Timothy P et al. (2020). "Backpropagation and the brain". In: *Nature Reviews Neuroscience*, pp. 1–12 (cit. on p. 15).

Lindsay, Alan E, KA Lindsay, and JR Rosenberg (2005). "Increased computational accuracy in multi-compartmental cable models by a novel approach for precise point process localization". In: *Journal of computational neuroscience* 19.1, pp. 21–38 (cit. on p. 9).

Looking inside neural nets (2021). URL: https://ml4a.github.io/ml4a/looking_inside_neural_nets/ (visited on 02/02/2021) (cit. on p. 26).

Masland, Richard H (2004). "Neuronal cell types". In: *Current Biology* 14.13, R497–R500 (cit. on p. 9).

McCloskey, Michael and Neal J Cohen (1989). "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, pp. 109–165 (cit. on p. 4).

Nadim, Farzan and Dirk Bucher (2014). "Neuromodulation of neurons and synapses". In: *Current opinion in neurobiology* 29, pp. 48–56 (cit. on p. 11).

Nagapawan, YVR, Kolla Bhanu Prakash, and GR Kanagachidambaresan (2021). "Convolutional Neural Network". In: *Programming with TensorFlow*. Springer, pp. 45–51 (cit. on p. 22).

Neuronal Dynamics (book) Adaptation and Firing Patterns (2021). URL: https://neuronaldynamics.epfl.ch/online/Ch6.S1.html (visited on 02/02/2021) (cit. on p. 11).

O'Reilly, Randall C (1996). "Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm". In: *Neural computation* 8.5, pp. 895–938 (cit. on p. 16).

Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu (n.d.). "Downsampled ImageNet 32x32". In: (). URL: http://image-net.org/small/download.php (cit. on p. 31).

OpenAI Website (2021). URL: https://openai.com/projects/five/ (visited on 02/02/2021) (cit. on p. 4).

PyTorch (2021). URL: https://pytorch.org/ (visited on 02/02/2021) (cit. on p. 5).

Rajendran, Bipin et al. (2019). "Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches". In: *IEEE Signal Processing Magazine* 36.6, pp. 97–110 (cit. on p. 15).

Rao, Rajesh PN and Dana H Ballard (1999). "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects". In: *Nature neuroscience* 2.1, pp. 79–87 (cit. on p. 17).

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (cit. on pp. 4, 26, 32, 37).

Spike-timing dependent plasticity (2021). URL: http://www.scholarpedia.org/article/Spike-timing_dependent_plasticity (visited on 02/02/2021) (cit. on p. 13).

Tang, Guangzhi et al. (2019). "Introducing astrocytes on a neuromorphic processor: Synchronization, local plasticity and edge of chaos". In: *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, pp. 1–9 (cit. on p. 15).

Tavanaei, Amirhossein et al. (2019). "Deep learning in spiking neural networks". In: *Neural Networks* 111, pp. 47–63 (cit. on p. 15).

Tensorflow API: Categorical Crossentropy (2021). URL: https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy (visited on 02/02/2021) (cit. on p. 27).

Urbanczik, Robert and Walter Senn (2014). "Learning by the dendritic prediction of somatic spiking". In: *Neuron* 81.3, pp. 521–528 (cit. on p. 37).

Van Essen, David C and John HR Maunsell (1983). "Hierarchical organization and functional streams in the visual cortex". In: *Trends in neurosciences* 6, pp. 370–375 (cit. on p. 21).

VGG16 – Convolutional Network for Classification and Detection (2021). URL: https://neurohive.io/en/popular-networks/vgg16/ (visited on 02/02/2021) (cit. on p. 26).

What are neurotransmitters? (2021). URL: https://qbi.uq.edu.au/brain/brain-physiology/what-are-neurotransmitters (visited on 02/02/2021) (cit. on p. 10).

What is an action potential? (2021). URL: https://www.moleculardevices.com/applications/patch-clamp-electrophysiology/what-action-potential (visited on 02/02/2021) (cit. on p. 12).

Whittington, James CR and Rafal Bogacz (2017). "An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity". In: *Neural computation* 29.5, pp. 1229–1262 (cit. on pp. 20, 27).

Whittington, James CR and Rafal Bogacz (2019). "Theories of error back-propagation in the brain". In: *Trends in cognitive sciences* 23.3, pp. 235–250 (cit. on p. 16).