Master of Science in Engineering with Specialisation in Data Science

Master Thesis

Deep-Learning-based Pattern Recognition and the Principle of Self Organization

Incorporating Findings from Neuroscience about Natural Intelligence into Modern Deep Learning Architectures

Pascal Sager

March 14, 2023

Zurich University of Applied Sciences

"Max Planck said, 'Science progresses one funeral at a time.'
The future depends on some graduate student who is deeply
suspicious of everything I have said."

– Geoffrey Hinton, University of Toronto, 2017.

Zusammenfassung

Deep Learning Systeme haben in den letzten Jahren beeindruckene Resultate erzielt und können unter anderem mit hoher Qualität Texte übersetzen, Unterhaltungen führen oder Bilder generieren. Diese Systeme werden typischerweise mit Backpropagation of Error über sämtliche Netzwerklayer hinweg als ein grosses System trainiert. Dieser Prozess ist zwar für spezifische Task der Bildverarbeitung wie Klassifizierung sehr effizient, ist aber neurowissenschaftlich nicht plausibel und hat verschiedenste Schwächen wie fehlende Robustheit und Interpretierbarkeit sowie nicht die Fähigkeit kausale Schlussfolgerungen zu ziehen.

In dieser Thesis werden neurowissenschaftliche Erkenntnisse identifiziert, die für die Intelligenz von Lebewesen wie dem Menschen elementar sind und auf Deep Learning Systeme adaptiert. Der Fokus liegt dabei auf dem visuellen Cortex als biologische Inspirationsquelle und Deep Learning Architekturen zur Bildverarbeitung als Zielsystem. Neurowissenschaftliche Erkenntnisse beziehen sich auf biologische Neuronen, welche sich gegenseitig durch zeitabhängige Spannungsspitzen anregen oder hemmen. Deep Learning Systeme hingegen haben keine Zeitdynamik und haben Aktivierungen basierend auf mathematischen Funktionen anstelle von Stromimpulsen. Folglich besteht ein grosser Beitrag dieser Arbeit darin die neurowissenschaftlichen Erkenntnisse in den Kontext von Deep Learning zu überführen. Spezifisch werden Vorschläge gemacht, wie selbst-organisation, Netzfragmente und laterale Verbindungen zur Verbesserung von Deep Learning Systemen genutzt werden könnten.

Die abgeleiteten Erkenntnisse werden in Form von zwei konkreten Modellen implementiert. Dabei ist der Fokus neue Architekturideen zu untersuchen und nicht Metriken wie Genauigkeit auf Benchmark-Datensätzen zu überbieten. Zudem orientieren sich die vorgeschlagenen Architekturen nahe am bewährten Deep Learning Framework und sind nicht als biologisch plausible Systeme zu verstehen. Konkret wird ein Modell mit vertikaler und ein Modell mit horizontaler Selbst-Organisation vorgeschlagen. Das Modell mit vertikaler Selbst-Organisation optimiert jedes Modell-Layer separat. Dabei werden neue Konzepte vorgestellt, die es erlauben alle Layer gleichzeitig zu trainieren und trotzdem hierarchische Features über die Layer hinweg zu erlernen. Das zweite Modell mit horizontaler Selbst-Organisation teilt die Eingabedaten in kleinere Einheiten auf und verteilt diese auf unabhängig trainierte Variational Auto-Encoders. Dabei sieht jedes Modell nur ein Teil der Eingabedaten und ist auf eine lokale Interaktion mit seinen Nachbarmodellen angewiesen, um eine passende Bildrepräsentation abzuleiten.

TODO: Schlussfolgerung / Erkenntnisse sobald Experimente abgeschlossen sind.

Abstract

Deep learning systems have achieved impressive results in recent years and can, among other things, translate texts, conduct conversations as chatbots, or generate high-quality images. However, these systems are typically trained with backpropagation of error in an end-to-end fashion across all network layers. While this process is very efficient for specific tasks such as classification, it is not neuroscientifically plausible. Moreover, it has various weaknesses, such as a lack of robustness, missing interpretability, and inability to do causal reasoning.

In this thesis, neuroscientific findings that are elementary for the intelligence of living beings such as humans are identified and transferred into a deep learning setting. The focus is on the visual cortex as a biological source of inspiration, while deep learning architectures for image processing are the target system. Neuroscientific findings refer to biological neurons that mutually excite or inhibit each other through time-dependent electric spikes. On the other hand, deep learning systems have no time dynamics but activations based on mathematical functions instead of electric pulses. Consequently, a significant contribution of this thesis is to transfer the neuroscientific findings into the context of deep learning. Specifically, suggestions are made on how self-organisation, net-fragments and lateral connections could be used to improve current deep learning systems.

The identified findings are implemented in the form of two concrete models in a deep learning setting. The aim is to demonstrate new architectural ideas rather than outperforming some metrics, such as the accuracy of well-known benchmark datasets. In addition, the proposed architectures are kept close to the well-established deep learning framework and are not intended to be biologically plausible systems. Specifically, a model with vertical self-organisation and a model with horizontal self-organisation is proposed. The model with vertical self-organisation optimises each model layer separately. Therefore, new concepts are introduced that allow all layers to be trained simultaneously while the model can still learn hierarchical features across layers. The second model with horizontal self-organisation splits the input data into smaller units and distributes them to independently trained variational auto-encoders. Each model sees only a patch of the input data and relies on local interaction with its neighbouring models to derive an appropriate image representation.

TODO: Conclusion / findings once experiments are completed.

Preface

As soon as I have handed in this thesis and it has been assessed with a sufficient grade, I may use the title "Master of Science". Thus, I should "master science" or at least be able to work scientifically. Science can be defined as the systematic analysis of the real or virtual world through observations and experiments and the further development of existing technology. While this definition sounds straightforward, working successfully in science requires a lot of experience and commitment. I have been lucky enough to learn more about science and apply my knowledge in research projects at the Centre for Artificial Intelligence (CAI) of the Zurich University of Applied Sciences (ZHAW) since the beginning of my master's studies. I am also privileged to be supported and mentored by Prof. Dr. Thilo Stadelmann, Head of the CAI. He uses to say (also in accordance with his blog-post) "Great methodology delivers great theses". It is always desirable to have an excellent outcome in a thesis, such as a system that can execute a task and achieves or even overcomes state-of-the-art performance. However, it is equally or even more essential to reason why and how something works, to justify choices, and to show limitations. Moreover, scientific breakthroughs always require a little courage to try something completely new, even if this means that there may not be a desirable result in the end. I write my thesis with these thoughts and hope readers can follow my reasoning (in contrast to neuronal networks, which are the subject of this thesis, but whose reasoning often cannot be followed).

This thesis comprises two fields; computer science and neuroscience. I try to link these fields as clearly as possible and to write in a way such that readers from both disciplines can follow my argumentation. However, this also means that some aspects are described rather extensively. So if you as a reader consider yourself a specialist in one of these fields, feel free to skip (parts of) like the "Fundamentals" in Chapter **??**. In general, the chapter "Fundamentals" describes principles that are not necessary for understanding this thesis but are helpful for people with a deep learning background to gain an overview of the field of neurocomputing and vice versa.

I also want to thank colleagues and friends for supporting this thesis. First, I think of my mentor Prof. Dr. Thilo Stadelmann, who got me excited about AI years ago and later introduced me to research. He always encourages creative ideas, thinking outside the box, and striving for greatness. Thank you for your support, help, and guidance; I have grown personally and professionally. Further thanks go to Dr. Jan Deriu. He has always helped to translate abstract ideas into concrete algorithms and to get them running. It's impressive how your understanding of deep learning can make complex problems look so simple. To Prof. Dr. Christoph von der Malsburg for his seemingly endless patience in introducing me to neuroscience. You have inspired me regularly with ideas and opened up a new way of thinking about deep learning (this was also the inspiration for Geoffrey Hinton's quote on the second page, although I wouldn't presume to say that this thesis will change the future). Even though we couldn't implement all of your ideas in this thesis, I learned a lot in our discussions and hope that I will be able to tackle more of your thoughts in the future.

The biggest thanks, however, goes to my family, who made this journey possible for me: To my parents, who supported and encouraged me in every way. To my younger brother, who inspired me to study. To my wife and son, who have been understanding and supportive and have always been the perfect counterbalance to the daily routine. Without the support of my family, I would never have been able to embark on this academic path.

Contents

List of Figures

List of Tables

Mathematical Terms & Definitions

1 Notation

Depending on the context, a variable can be a scalar value, a vector, or a matrix. The following formatting is used:

Formatting	Example	Meaning
No formatting, lower case	a	A scalar value
Bold, lower case	$\boldsymbol{a}$	A vector
Bold, upper case	$\boldsymbol{A}$	A matrix
Brackets with a point	$a(\cdot)$	A function, where $(\cdot)$ is a placeholder for a variable
Superscript rectangular brackets	$a^{[l]}$	l is the index of the layer, e.g. $\boldsymbol{W}^{[l]}$ is the weight matrix of layer l

2 Variables

Variable	Meaning
η	The learning rate of an optimization algorithm
ψ	An estimate of the synaptic activity, used for Hebbian learning (i.e. an estimate of a)
θ	A threshold value that was used instead of the bias b in early versions of neural networks
a	The output of an activation function of a single neuron
$\boldsymbol{a}$	The output of an activation function in an intermediate layer of a multi-layer network, the output of the model is typically denoted as $\hat{\boldsymbol{y}}$
b	The bias of a single neuron
$\boldsymbol{b}$	The bias of a network layer
k	The number of neurons within a layer
L	The number of layers of a network
l	The index of a layer, e.g. $\boldsymbol{W}^{[l]}$ is the weight matrix of layer l
m	The number of input samples within a (mini-)batch or the state of the memory in context of Hoppfield networks
n	The length (size) of a vector, for example, an input sample is defined as $\boldsymbol{x} = (x_1, ..., x_n)$
w_{ij}	The weight between neuron i and neuron j
$\boldsymbol{w} = (w_1, ..., w_n)$	A weight vector of a neuron
$\boldsymbol{W} = (\boldsymbol{w}_1, ..., \boldsymbol{w}_k)$	A weight matrix of a network layer
$\boldsymbol{x} = (x_1, ..., x_n)$	An input sample that is fed into a model or a network layer, typically a vector of length n
$\boldsymbol{y}$	The expected output of a model or a network layer (i.e. the ground truth), typically a vector ($\boldsymbol{y}$) for a multi-class classification task or a scalar value (y) for a regression or single-class classification task
$\hat{\boldsymbol{y}}$	The actual output of a model or a network layer (i.e. the prediction), typically a vector ($\hat{\boldsymbol{y}}$) for a multi-class classification task or a scalar value ($\hat{y}$) for a regression or single-class classification task (in a multi-layer network, the output of an intermediate layer is typically denoted as $\boldsymbol{a}$)
z	The output of the aggregation function $g(\cdot)$ for a single neuron
$\boldsymbol{z}$	The output of the aggregation function $g(\cdot)$ for a network layer

3 Functions

Function	Meaning
$\mathcal{L}(\cdot)$	Loss function to calculate the goodness of the model's prediction
$E(\cdot)$	An encoder (typically multiple network layers) to transform an input to a latent representation
$f(\cdot)$	An activation function such as $\sigma(\cdot)$, $(\cdot)^+$, or $\tanh(\cdot)$ (c.f. equation (**??**))
$g(\cdot)$	The aggregation function that calculates the scalar value that is fed into the activation function $f(\cdot)$ (c.f. equation (**??**), equation (**??**), equation (**??**))
$G(\cdot)$	A function to update the memory of Hopfield networks
$O(\cdot)$	An output function to calculate the output of a Hopfield network based on the input and the memory
$I(\cdot)$	The energy function of a Hopfield network
$\sigma(\cdot)$	The sigmoid activation function (c.f. equation (**??**))
$(\cdot)^+$	The ReLU activation function (c.f. equation (**??**))
$\tanh(\cdot)$	The hyperbolic tangent activation function (c.f. equation (**??**))

1 Introduction

Humanity has always tried to simplify its life through technological progress. The computer has shaped progress in every field imaginable in the last hundred years. In fact, it has been so successful that it has spawned its own scientific discipline, computer science. Nowadays, it is impossible to imagine life without computers: almost every household and company owns at least one. Computers facilitate various tasks, from communication to the acquisition of knowledge. For all these tasks, software developers have created appropriate programs. Software development is the process of writing a script with a programming language to specify how the computer should behave for a given input. Simply put: A software program tells the computer what to do if the user enters a command. Writing software can be done easily if the tasks are clearly defined and can be described precisely. However, there exist tasks that are almost impossible to program, such as writing a script that detects cats in images because we cannot describe to a computer how a cat looks precisely[1].

1: at least not on a pixel-level basis so that we can compare a given image with our description

Therefore, scientists came up with the idea to not just program such tasks but to let the computer learn them. Machine learning (ML) algorithms can learn and adapt without following explicit instructions such as program code. Instead, they use statistical models to analyse data, find patterns, and make predictions. Machine learning has become an indispensable part of our everyday lives. For example, we use it for machine translation, transport and logistics organisation, product recommendations, fraud detection, self-driving cars, unlocking smartphones, improving video games, speech recognition, and much more. A sub-branch of machine learning is deep learning (DL) which achieved awe-inspiring results in the last decade and is considered *the* state-of-the-art technology for many of the aforementioned tasks.

Even though this technology works very well for many tasks, such models have some crucial flaws by definition (c.f. Section 2.3), which cannot be resolved in the current DL framework. One of the godfathers of deep learning is Turing Award[2] winner Geoffrey Hinton. Especially his contribution to learning with backpropagation of error (c.f. Section 2.2) has created the foundation for modern deep learning systems. More than 30 years later, Hinton says he is "deeply suspicious" about end-to-end backpropagation of error. In his opinion, we have to "throw it all away and start again" to improve current systems fundamentally [1]. This seems rather extreme considering what DL systems have achieved. However, it also shows that the current learning algorithm of such systems has serious flaws.

2: the Turing Award is recognized as the highest academic award in computer science and sometimes also called the "Nobel Prize of Computing"

[1]: Axios Media Inc. (2017)

Some of the most critical limitations of deep learning systems are listed in Section 2.3. This thesis addresses these problems by exploring different architectures inspired by findings from neuroscience[3]. The inspiration is drawn from neuroscience, as the human brain does not have these problems. Thus, incorporating findings from neuroscience into the deep

3: the field of neuroscience studies how the human nervous system and especially the brain works

learning framework might help overcome some of the current limitations. In this thesis, the "Theory of Natural Intelligence" by von der Malsburg et al. [2] in particular is used as inspirational source. The aforementioned theory focuses on the visual processing in the human brain, and consequently, deep learning architectures for image processing are examined. Thereby, the main focus of this thesis is on *finding new principles* of deep learning architectures rather than on achieving state-of-the-art performance in terms of accuracy or f-score on existing benchmarking datasets.

[2]: von der Malsburg et al. (2022)

The thesis differs from neurocomputing[4] in the sense that the architectures investigated are still closely related to the DL framework and, for example, do not use time-dynamic signals. For the completeness of this thesis, an overview of neurocomputing is given in Section 2.5. However, since these neurocomputing algorithms have not (yet) become established in practical applications, they are not investigated in this thesis. Thus, the thesis is strongly oriented along the field of deep learning and focuses on incorporating neuroscience findings in the deep learning framework, while natural plausibility is considered less important.

4: neurocomputing is a sub-field of neuroscience that deals with the implementation of learning algorithms with a high degree of biological plausibility

1.1 Contribution

This thesis makes the following contributions:

- First, the basics of deep learning and neurocomputing are described in detail. Together with the related work, this provides a survey of the most important work that deals with improving learning generally.
- Second, promising concepts from the neuroscience literature that may be necessary for more intelligent systems are identified, and suggestions are made on integrating these concepts into modern DL frameworks.
- Third, two concrete DL architectures that use such neuroscience-inspired concepts are presented, and their strengths and weaknesses are analysed in detail.
- Fourth, this work provides an intuition of which concepts currently little used in DL settings seem promising and recommends future research directions.

1.2 Organisation of Thesis

The remainder of the thesis is organised as follows: In Chapter 2, the fundamentals of deep learning and neurocomputing are explained in detail. Experts may skip this chapter; especially the second section on neurocomputing can optionally be omitted, as it is not fundamental for understanding this thesis, but only intended for interested readers or those who want to get a better overview of the field. Chapter 3 introduces the related work. Next, Chapter 4 identifies promising findings from neuroscience that might help to improve current deep learning systems. Furthermore, it is proposed how these neuroscientific concepts could

be implemented in a deep learning setting. In Chapter 5 and Chapter 6, two different architectures incorporating these concepts are presented. Finally, in Chapter 7, future work is described, and some insights and intuitions of the author are given about which neuroscience concepts seem promising and which do not, and what the next steps for developing more intelligent systems could be.

Fundamentals 2

Machine learning uses mathematical functions to map an input to an output. These functions usually extract patterns from the input data to build a relationship between input and output. The term machine learning stems from the fact that we use *machines* to correlate the input and the output to a function (i.e. to *learn* a function) during a training period. Deep learning is a sub-branch of machine learning and is considered state-of-the-art for many learning tasks, especially on high-dimensional data such as texts, audio recordings, 2D and 3D images, and videos. Deep learning algorithms use neural networks that are heavily inspired by networks of neurons within the human brain.

Therefore, Section 2.1 first briefly explains how biological neurons work and relates them to artificial neurons. Next, Section 2.2 describes artificial neural networks that connect many artificial neurons. In Section 2.3, problems of such artificial neural networks are pointed out. Next, Section 2.4 describes some of the differences between deep learning and biological learning. Finally, Section 2.5 describes biologically more plausible learning methods that are investigated in the field of neurocomputing. These last two sections are not important for understanding this thesis and can optionally be skipped. However, they are intended as a supplement for interested readers who want to get a better overview of the whole field.

2.1 Biological Neurons

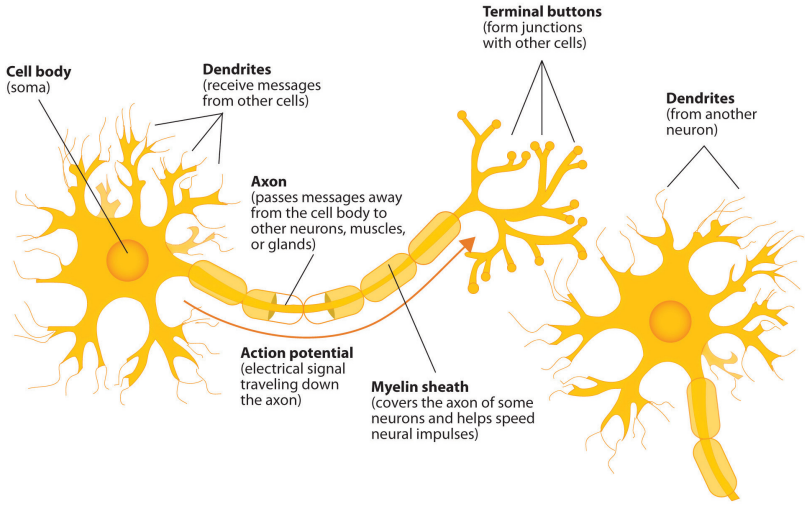

Figure 2.1: A Diagram of the components of a biological neuron. The image is from Wikipedia [3].

A biological neuron (c.f. Figure 2.1) is a cell that communicates with other neurons through connections called synapses. Communication takes place through precisely timed electrical pulses called spikes. Biological neurons are electrically excitable by voltage changes across their membranes. If the changes are significant enough within a short interval, the

neuron generates a pulse called an action potential. This action potential travels through the axon and activates synaptic connections. Other neurons, connected through synapses, receive this signal. The synaptic signal can be excitatory [4] or inhibitory [5], making the post-synaptic neuron more or less likely to fire an action potential. Biological neurons can be classified into sensory neurons, motor neurons, and interneurons. Sensory neurons respond to external stimuli such as light or sound and send signals to the spinal cord or the brain. Motor neurons receive brain and spinal cord signals to control muscles or organs. Interneurons connect neurons within the same region of the brain or of the spinal cord. Multiple connected neurons form a neural circuit. The neural network in the brain is not static but changes through growth and reorganisation. This process is referred to as neuroplasticity or neural plasticity [6].

[4]: Takagi (2000)

[5]: Coombs et al. (1955)

[6]: Costandi (2016)

Like a biological neuron, an artificial neuron is connected to other neurons. Artificial neurons are usually organised in layers that forward signals sequentially. Although the neurons in the first layer could be considered sensory neurons, the neurons in the last layer could be considered motor neurons, and the neurons in the middle layer could be considered interneurons, such a distinction makes less sense because the artificial neurons function similarly regardless of their layer except for the activation function. Several variants for artificial neurons have been proposed in the literature. These variants are described in the following Section 2.2. Like biological neurons, multiple artificial neurons are connected to artificial neural networks.

2.2 Artificial Neural Networks

The idea for artificial neural networks (ANN) stems from biology and aims to capture the interaction of biological neurons with a mathematical model. McCulloch and Pitts proposed the first model for a neuron that can be connected to other neurons in 1943 [7]. Similar to how a neuron of the human brain transmits electrical impulses through the nervous system, the artificial neuron of McCulloch and Pitts receives multiple input signals and transforms them into an output signal. A neuron takes an input vector $\boldsymbol{x} = (x_1, ..., x_n)$ where $x_i \in \{0, 1\}$ and maps it to an output $\hat{y} \in \{0, 1\}$. The mapping from the input to the output is done by using an aggregation function $g(\cdot)$ that sums up the input vector $\boldsymbol{x}$ and an activation function $f(\cdot)$ that outputs 1 if the output of $g(\cdot)$ is bigger than a threshold θ and 0 otherwise.

[7]: McCulloch et al. (1943)

$$z = g(\boldsymbol{x}) = g(x_1, ..., x_n) = \sum_{i=1}^{n} x_i \tag{2.1}$$

$$\hat{y} = f(z) = \begin{cases} 1, & \text{if } z \geq \theta \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

In 1958, Rosenblatt [8] developed the Perceptron, which works with real numbers as input. The input vector $\boldsymbol{x} \in \mathbb{R}^n$ is multiplied with a weight

[8]: Rosenblatt (1958)

vector $\boldsymbol{w} \in \mathbb{R}^n$ with the same length n.

$$z = g(\boldsymbol{x}) = g(x_1, ..., x_n) = \sum_{i=1}^{n} w_i \cdot x_i \tag{2.3}$$

The output $\hat{y} \in \{0, 1\}$ is similar to the McCulloch and Pitts neuron 1 if the aggregated value is greater than a threshold θ and 0 otherwise as described in equation (**??**). With real numbers as input, the equations (**??**) and (**??**) can be rewritten as

$$\hat{y} = f(g(\boldsymbol{x})) = \begin{cases} 1, & \text{if } z - \theta \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

Later, the step-function $f(\cdot)$ was replaced with other functions so that the output can also be a real number $\hat{y} \in \mathbb{R}$. Often-used activation functions are

$$\begin{aligned} \text{Sigmoid: } \sigma(z) &= \frac{1}{1 + \mathrm{e}^{-z}} \\ \text{Rectified linear unit (ReLU): } (z)^+ &= \max\{0, z\} \\ \text{Hyperbolic tangent (tanh): } \tanh(z) &= \frac{\mathrm{e}^{z} - \mathrm{e}^{-z}}{\mathrm{e}^{z} + \mathrm{e}^{-z}} \end{aligned} \tag{2.5}$$

By convention, a positive bias b is often used instead of the negative threshold $-\theta$, which leads to:

$$z = g(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = \left(\sum_{i=1}^{n} w_i \cdot x_i \right) + b \tag{2.6}$$

The neuron's output is calculated with the activation function of z:

$$\hat{y} = f(z) \tag{2.7}$$

So far, only the output of a single neuron has been discussed. However, the brain consists of multiple neurons which are connected through synapses. Therefore, also ANNs consist not only of one neuron but combine multiple neurons in a network. These neurons are organized in layers. In the simplest case, all neurons from one layer are connected with all neurons of the subsequent layer. This is called a fully connected layer. For a network with multiple neurons within one layer, the input $\boldsymbol{x}$ is fed into all neurons to obtain $\hat{\boldsymbol{y}}$. If the layer has k neurons, the output of the aggregation function becomes a vector $\boldsymbol{z} = (z_1, ..., z_k)$. The same applies to the output of the activation function $\boldsymbol{y} = (y_1, ..., y_k)$ and the bias $\boldsymbol{b} = (b_1, ..., b_k)$. The weight vector, on the other hand, becomes a matrix $\boldsymbol{W} = (\boldsymbol{w}_1, ..., \boldsymbol{w}_k)$. For a layer with multiple neurons, equation (**??**) and equation (**??**) can be rewritten with matrix operations:

$$\boldsymbol{z} = \boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b} \tag{2.8}$$

$$\hat{\boldsymbol{y}} = f(\boldsymbol{z}) \tag{2.9}$$

which is equal to

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_1 \cdot \boldsymbol{x} + b_1 \\ \boldsymbol{w}_2 \cdot \boldsymbol{x} + b_2 \\ \dots \\ \boldsymbol{w}_k \cdot \boldsymbol{x} + b_k \end{bmatrix} = \begin{bmatrix} \left(\sum_{i=1}^{n} w_{1i} \cdot x_i\right) + b_1 \\ \left(\sum_{i=1}^{n} w_{2i} \cdot x_i\right) + b_2 \\ \dots \\ \left(\sum_{i=1}^{n} w_{ki} \cdot x_i\right) + b_k \end{bmatrix} \tag{2.10}$$

$$\hat{\boldsymbol{y}} = \begin{bmatrix} y_1 = f(z_1) \\ y_2 = f(z_2) \\ \dots \\ y_k = f(z_k) \end{bmatrix} \tag{2.11}$$

[9]: Cybenko (1989)

The universal approximation theorem[9] proves that a shallow network with one hidden layer (i.e. one layer between input and output layer) and enough neurons can approximate any mapping function between inputs and outputs. However, very complex mapping functions may need too many neurons in the hidden layer. Sequentially arranging multiple layers is much more efficient for approximating complex functions. A sequential arrangement allows learning a hierarchy of features by dividing the mapping function over several successive processing steps.

In an MLP, the input $\boldsymbol{x}$ is fed into the first layer, and each subsequent layer l uses the output of the previous layer $l-1$ as input. For a network with L layers, we denote the layer index as superscript square brackets, i.e. $(\cdot)^{[l]}$. For example, the weights of layer l are denoted as $\boldsymbol{W}^{[l]}$, the bias as $\boldsymbol{b}^{[l]}$, the output of the aggregation function as $\boldsymbol{z}^{[l]}$, and the output of the activation function as $\boldsymbol{a}^{[l]}$. The input in the first layer is the input data, i.e. $\boldsymbol{a}^{[0]} = \boldsymbol{x}$, and the output of the last layer is the model's prediction, i.e. $\boldsymbol{a}^{[L]} = \hat{\boldsymbol{y}}$. Thus, the mathematical model of an MLP is defined as

$$\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]} \tag{2.12}$$

$$\boldsymbol{a}^{[l]} = f(z^{[l]}) \tag{2.13}$$

So far, only the forward pass used to calculate the output $\hat{\boldsymbol{y}}$ has been discussed. However, the model output $\hat{\boldsymbol{y}}$ will only be close to the target output $\boldsymbol{y}$ if the weights $\boldsymbol{W}^{[l]}$ and biases $\boldsymbol{b}^{[l]}$ are properly defined in every layer l. These parameters are learned during a training period. The training can take place in a supervised, semi-supervised, self-supervised (sometimes also called unsupervised), or reinforcement learning setting. In supervised learning, the output of the model $\hat{\boldsymbol{y}}$ is compared to a given target output $\boldsymbol{y}$. On the other hand, unsupervised learning tries to find patterns in the input data $\boldsymbol{x}$ and cluster the samples into meaningful groups without using pre-defined target labels. Typically, the target $\boldsymbol{y}$ is derived from the data automatically (e.g. predict a masked part of the data); since the model creates the target by itself, this approach is also called self-supervised. Semi-supervised learning is a hybrid approach of the aforementioned principles that combines a small amount of labelled data with a large amount of unlabelled data. Lastly, reinforcement learning algorithms aim to maximize the reward that they receive from an environment based on some action they executed.

These learning principles have in common that a loss function (also called objective function) $\mathcal{L}(\cdot)$ can calculate a loss value based on the model output $\hat{\boldsymbol{y}}$ and the target output $\hat{y}$. For example, the mean square error (MSE) can be used for regression problems or the negative log-likelihood for classification problems. The chosen loss function is minimized iteratively with stochastic gradient descent (SGD)[1] until the network converges to a (local) minima. The idea behind stochastic gradient descent is to make use of the fact that the negative gradient of the loss value points to the direction of the steepest descent (i.e. in the direction where the loss

1: there also exist other optimization algorithms such as SGD with momentum, RMSprop, or Adam [10]

becomes smaller). Therefore, SGD updates the network parameters by taking a step of size η toward their negative gradient:

$$\begin{aligned} \Delta \boldsymbol{W}^{[l]} &= -\eta \cdot (\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}) \\ \boldsymbol{W}^{[l]} &:= \boldsymbol{W}^{[l]} + \Delta \boldsymbol{W}^{[l]} \end{aligned} \tag{2.14}$$

and

$$\begin{aligned} \Delta \boldsymbol{b}^{[l]} &= -\eta \cdot (\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}) \\ \boldsymbol{b}^{[l]} &:= \boldsymbol{b}^{[l]} + \Delta \boldsymbol{b}^{[l]} \end{aligned} \tag{2.15}$$

The term $(\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L})$ is the gradient of the weights $\boldsymbol{W}^{[l]}$ with respect to the loss $\mathcal{L}(\cdot)$ and the term $(\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L})$ is the gradient of the bias $\boldsymbol{b}^{[l]}$ with respect to $\mathcal{L}(\cdot)$. The gradients of the weights can efficiently be calculated with an algorithm called backpropagation of error [11], which is, in fact, just an intelligent implementation of the chain rule[2].

[11]: Rumelhart et al. (1986)

2: While a detailed discussion on backpropagation is out of scope for this thesis, we refer interested readers to the deep learning course by Andrew Ng [12]

One of the most critical design decisions in creating ANNs is how the neurons are connected. So far, only the case where every neuron of one layer is connected to every neuron of the following layer (so-called fully connected layer) has been described. Besides such dense connections, there are several alternatives. Since this work deals with the processing of visual scenes, the most well-known image-processing network architecture, namely Convolutional Neuronal Networks (CNN), is presented below. This architecture is also primarily used in this thesis. However, various alternative architectures exist for computer vision, such as Vision Transformer (ViT) [13] or MLP Mixer [14], which, however, would go beyond the scope of this introduction to deep learning.

[13]: Dosovitskiy et al. (2021)

[14]: Tolstikhin et al. (2021)

2.2.1 Convolutional Networks

Convolutional Neural Networks (CNNs) are particularly useful for finding patterns in images but can also be used to analyse non-image data such as audio files or time series. Similar to FCNs, a CNN is composed of an input layer, an output layer, and multiple hidden layers in between. A typical CNN consists of subsequently connected convolutional layers and pooling layers. Usually, an activation function is applied after each convolutional layer, while no activation function is used after pooling layers. Depending on the task, the last layers can be different; for example, the last layers are often fully connected for image classification.

Convolutional layers use convolution filters or kernels that slide along input the input and create translation-equivariant[3] [15] responses known as feature maps [16]. When applying the filter, the dot product between an input area (of the same size as the filter) and the filter is calculated, and the resulting scalar value is set at one position of the output matrix (i.e. the feature map). Afterwards, the filter shifts by a stride, repeating the process until the entire input has been processed and all values of the output matrix have been calculated. Since only the kernels have to be learned, convolutional layers consist of much fewer parameters than equivalently sized fully connected layers. This process of re-using the same weights at different input locations is also known as parameter sharing. As described earlier, multiple convolutional layers can follow each other. By doing so, CNNs become hierarchical: Since the convolutional operation squeezes information from the surrounding pixel into the output of one pixel,

3: The placement of the objects remains consistent between layer input and output, as the same filter is applied to all image positions

[16]: Zhang et al. (1990)

using multiple layers sequentially continuously enlarges the receptive fields[4].

4: the field of input pixels that can influence a single value in the feature map of a layer

Pooling layers, on the other hand, reduce the size of the input by conducting dimensionality reduction. Similar to convolutional layers, a filter slides along the input. However, this filter has no learned parameter but applies an aggregation function. Usually, the filter selects the pixel with the highest value (max pooling) or calculates the average (average pooling) within the considered input area and uses this value as output. The filter is then shifted by the filter size so that non-overlapping patches of the image are processed. Pooling layers usually discard a lot of information but help to reduce complexity and increase robustness.

In the last decades, various CNN architectures have been proposed, usually consisting of different convolutional and pooling layers combinations. Further improvements have been achieved by using parallel paths of convolutional layers, batch normalisation [17], and skip connections[5]. Describing such specific architectures is out of scope for this thesis, but some references to well-known architectures are provided in the following; LeNet [18], AlexNet [19], VGGNet [20], GoogLeNet [21], ResNet [22], U-Net [23], Mask R-CNN [24], SSD [25], and YOLO [26].

[17]: Ioffe et al. (2015)

5: skip connections skips some of the layers in the network and add the output of a layer to the input of a later layer

[18]: Lecun et al. (1998)

[19]: Krizhevsky et al. (2012)

[20]: Simonyan et al. (2015)

[21]: Szegedy et al. (2014)

[22]: He et al. (2016)

[23]: Ronneberger et al. (2015)

[24]: He et al. (2017)

[25]: Liu et al. (2016)

[26]: Redmon et al. (2016)

Training of Convolutional Neural Networks

CNNs can be used for various applications such as image recognition, video analysis, natural language processing, or anomaly detection in time series. Since this thesis deals with alternative learning algorithms for computer vision models, this chapter is limited to the typical image analysis tasks; image classification, object detection, and image segmentation. An overview of these tasks is given in Figure 2.2. Image classification aims to predict an image-level label, i.e. to answer the question of what object is in the image. A classic example of this problem is predicting whether a cat or a dog is in a picture. Image classification is typically applied to images that contain only one object. With a multi-label classifier, however, it is also possible to predict whether none, one or several classes are present, i.e. whether a cat, a dog or both are present in the image. However, when only predicting an image-level label, it is unclear where in the image these objects are located. Object detection provides a remedy. The aim of object detection is not only to determine what is visible in the image but also where. The object's position is usually indicated by a bounding box (i.e. a rectangle). Especially when there are several objects within a picture, it is helpful to know which object is where, e.g., the cat is to the left of the dog. For some applications, predicting the position with bounding boxes is not sufficient. In this case, semantic segmentation is often used. Semantic segmentation is the task where each pixel is classified (the image is divided into segments), which leads to a pixel-wise mask for each object in the image. Thus, semantic segmentation provides detailed information about the shapes of the objects.

Depending on the task, different kinds of labels are required for supervised learning. Image classification requires labels on image-level (i.e. one or multiple labels per image) [18–22], object detection requires the coordinates of the bounding box [24–26], and semantic segmentation

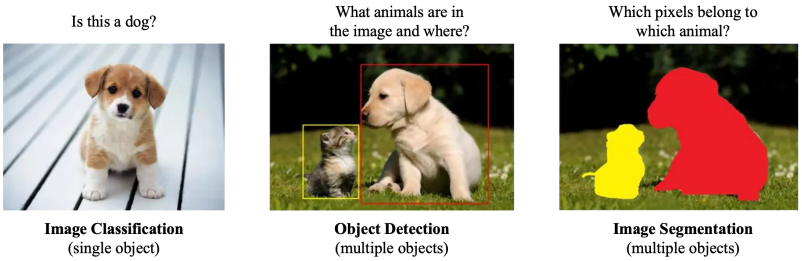

Figure 2.2: Overview of different image analysis tasks. The image is taken from Venture Beat [27] and was slightly adapted.

requires the labels on pixel-level [23, 28] (i.e. each pixel has a label of an object or a label "background" for irrelevant pixels).

Besides supervised learning, there are various methods to learn with partial labels or without labels. These techniques are called weakly-supervised or un-supervised learning and are well summarised ny Simmler et al. [29]. Not only specific tasks can be learned, but also task-independent representations. These representations are typically learned unsupervised[6] and can be used for one or several downstream tasks. More details on visual representation learning are provided in Section 3.4. Especially auto-encoders [30] are explained in this section as they are applied in this thesis.

[29]: Simmler et al. (2021)

6: also called self-supervised learning because the target labels are derived from the data itself

[30]: Rumelhart et al. (1985)

2.3 Limitations

The rise of deep learning over the past decade has only been possible because of significant technological advances in hardware. Moore's law [31] states that the number of transistors in a dense integrated circuit doubles about every two years and is one of the only known physical processes that follows an exponential curve. However, the exponential increase comes to an end since the size of transistors hit physical limitations [32]. Nevertheless, not only has the hardware improved massively recently, but deep learning models consume more computing resources in general: An analysis by OpenAI shows that since 2012 the amount of computing used by AI models has increased exponentially with a doubling time of 3.4 months [33]. Thus, the compute usage increases even faster than the progress of hardware.

[31]: Moore (2006)

[32]: Kumar (2015)

[33]: Open AI (2018)

Many recent improvements in deep learning models are due to massively growing model and data set sizes. For example, the growth of language models over the last five years is remarkable: While a state-of-the-art language model from 2018 [34] had around 94M parameters, the state-of-the-art in 2020 [35] already had 175B parameters. Training such a model on a single V100 GPU would take about 355 years and cost about 4.6M dollars in the cloud [36]. A recent language model from Microsoft and NVIDIA [37] even has 530B parameters. Thus, the size of the language models has increased 563-fold within five years. Only a few institutions with massive resources can train such big models. In general, inference on low-budget hardware such as smartphones or embedded hardware becomes prohibitive with the growing size of deep networks. Even though there exist techniques to shrink the model size after training, such as quantization [38], model pruning [39], or model distillation [40] it is questionable if making models bigger is the best way to develop more intelligent systems.

[34]: Peters et al. (2018)

[35]: Brown et al. (2020)

[36]: Lambda Inc. (2021)

[37]: Smith et al. (2022)

Another major issue of deep learning systems is that they suffer from catastrophic forgetting. If a model is trained on a specific task and afterwards trained (or fine-tuned) on another task, the model suffers a "catastrophic" drop in performance over the first task. The reason for this effect is that during training on the second task, the model adjusts the parameters learned during the first task and therefore "forgets" the learned mapping functions. Mixing all data sets or learning all tasks in parallel in a multi-task setting [41] does not seem feasible to achieve general intelligence as general intelligence might require training on many unrelated tasks. Therefore, models should not forget previously learned knowledge. Catastrophic forgetting is also caused by the fact that learning is mostly done offline[7]. Online learning [42] and lifelong learning [43] are hot research topics. However, these methods still need to be established. Moreover, such models do not solve catastrophic forgetting but can adapt better to changing conditions.

7: offline in this context means that the model parameters are not adapted after training during inference time

Furthermore, some problems may not be solved with the current principles of deep learning. First of all, it is questionable if deep learning models can achieve *real* generalization[8]. With enough data, DL can achieve generalization in the sense that the model can interpolate within the known data distribution. However, deep learning models fail to extrapolate. For example, convolutional neural networks (CNNs) do not generalize to different viewpoints unless they are added to the training data [44].

8: generalization refers to the ability of the model to adapt appropriately to previously unseen data from the same distribution

[44]: Madan et al. (2022)

Second, deep learning cannot learn abstract relationships in a few trials but requires many samples and is thus data-hungry. Gary Marcus [45] showcased this problem with an example: He defines the new word "schmister" as a sister over the age of 10 but under the age of 21. He found that humans can immediately infer whether they or their best friends have any "schmister". However, modern DL systems lack a mechanism for learning abstractions through explicit, verbal definitions and require thousands or even more training samples.

[45]: Marcus (2018)

Third, no DL model has been able to demonstrate causal reasoning generically. Deep learning models find correlations between input and output data, but not causation. Other AI approaches, such as hierarchical Bayesian computing or probabilistic graphical models are better at causal reasoning but cannot be well combined with deep learning architectures.

Lastly, deep learning models are, to some extent, too isolated since they have no embodiment and cannot interact with the world. For example, the human body provides needs, goals, emotions, and gut feeling[9]. In current deep learning systems, emotions are absent, and the goals are set externally. Deep reinforcement learning is a first step toward dissolving this isolation as they interact with a virtual environment. However, AI systems interacting with the real world have not worked well so far. Moravec's paradox from 1995 [46] states that "it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility". This statement still seems true almost 30 years later.

9: one could argue that the body is, therefore, even a co-processor of the brain

[46]: Moravec (1995)

2.4 Biological Learning

The human brain comprises many interconnected areas processing much information in parallel. For example, Figure 2.3 illustrates the connections between different organisational units in the cerebral cortex responsible for vision. These areas are connected in a rather complex structure. On

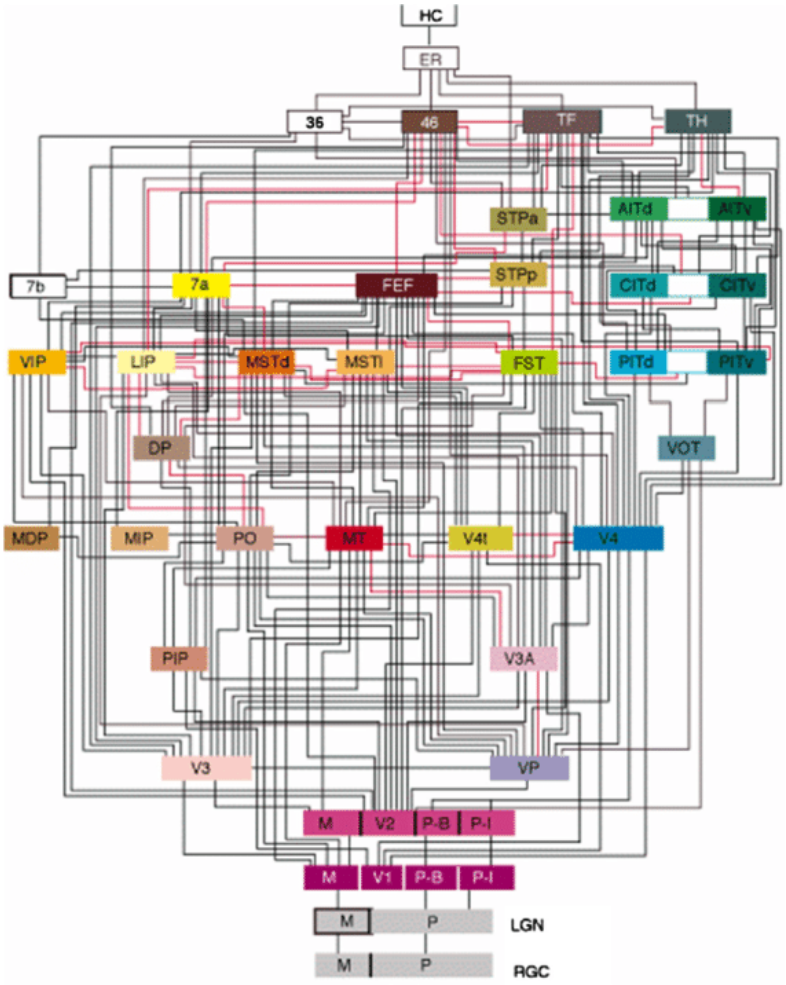

Figure 2.3: The organisation of the visual system in the cerebral cortex. The image is from Felleman et al. [47].

the other hand, deep learning architectures are primarily sequential, and the signal flows forward or backwards from layer to layer[10]. These architectures are not divided into many different sections. Instead, deep learning architectures tend to be monoliths, i.e. one large unit of layers rather than several small units with clearly assigned responsibilities. However, the choice of architecture influences how the model can learn the mapping function from input to output. It could be that the complex structure of our brain comprises an inductive bias that is needed for intelligent image processing and has been learned over time through evolution.

10: except for recurrent connections, skip connections, or residual connections

An artificial learning system requires a mechanism that tells the system if something goes well or wrong so that it can learn from it. This is called the *credit assignment problem*. Backpropagation of error (c.f. Section **??**) is the state-of-the-art algorithm that solves this problem by propagating the error signals back through the network. However, information flows in the brain only in one direction, from presynaptic to postsynaptic

neurons. Therefore, backpropagation of error is not biologically plausible. Researching alternative learning algorithms for artificial neural networks is a hot research topic summarised in Section 3.3.

Not only the structure of the network and the way how the feedback is calculated is different between biological learning and deep learning. Also, the neurons themselves are different. While the artificial neuron does not have any dynamics (c.f. equation (**??**)), biological neurons are highly dynamic: Biological neurons adapt their firing rate to constant inputs, and they may continue firing after an input disappears and can even fire when no input is active (c.f. Section 4.1).

Lastly, the neurons in the brain are self-organising. Self-organisation is when a group of elementary units, such as neurons or a group of neurons, follow similar rules of behaviour on a subset of the available information. Such a system does not have a central supervision that orchestrates these units. Each unit applies similar deterministic functions to the information received. Two important principles of such systems are: (i) localised learning, which means that each unit adapts its behaviour to the information they receive, and (ii) emergence, which means that there is no explicit global loss function that tells the system what to do.

2.5 Neurocomputing

Neurocomputing is considered a subfield of neuroscience and focuses on implementing more biologically plausible learning algorithms. Therefore, the abovementioned discrepancy is mainly investigated, and alternative learning algorithms are developed.

2.5.1 Hebbian Learning

Donald Hebb describes how the connections between cells in the nervous system adapt: "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" [48]. This statement is often simplified to the well-known phrase "Neurons that fire together wire together".

[48]: Hebb (1949)

He proposed Hebbian learning that is based on this principle. The weight w_{ij} from neuron i to neuron j changes based on the pre-synaptic activity a_i of neuron i and post-synaptic activity a_j of neuron j[11]

11: the pre-synaptic and post-synaptic activity corresponds to the output of the activation function $f(\cdot)$ of a neuron in the previous resp. subsequent layer

$$\Delta w_{ij} = \eta a_i a_j \tag{2.16}$$

where η is the learning rate. Thus, the weights between frequently co-activated neurons increase called Hebbian plasticity.

In its original form, Hebbian learning had the problem that the connections can only become stronger but not weaker. Therefore, this formula is often extended based on the covariance of the activity between neurons. The covariance is positive if two neurons fire often together and negative

if they do not often fire together. The following equation changes the weight relative to the covariance:

$$\Delta w_{ij} = \eta(a_i - \psi_i) \cdot (a_j - \psi_j) \tag{2.17}$$

where ψ_i and ψ_j are estimates of the expected pre- and post-synaptic activity[12]. However, the formulation above lacks boundaries, i.e. the weights could grow to infinite. A simple solution is to enforce hard boundaries $w_{min} \leq w_{ij} \leq w_{max}$. An alternative to enforce boundaries is to normalize the length of the weight vector [49].

12: the expected activity can, for example, be estimated through a moving average function

[49]: Oja (1982)

$$\Delta w_{ij} = \eta(a_i a_j - \alpha r_j^2 w_{ij}) \tag{2.18}$$

where $\alpha r_j^2 w_{ij}$ is the regularization term, and the parameter α is a parameter that determines the size of the norm of the weight vector. By using this regularization term, the weight vector $\boldsymbol{w}$ convergences to

$$||\boldsymbol{w}||^2 = \frac{1}{\alpha} \tag{2.19}$$

Other measures also exist that limit the size of the vector $\boldsymbol{w}$, for example, by using rate-based threshold adaption [50, 51]. Rate-based threshold adaption uses a sliding threshold for long-term potentiation (LTP) or long-term depression (LTD) induction. When a pre-synaptic neuron fires and the post-synaptic neuron is in a lower activity state than the sliding threshold, it undergoes an LTD (i.e. the connection is weakened); otherwise, an LTP is applied, and the connection is strengthened.

[50]: Bienenstock et al. (1982)

[51]: Intrator et al. (1992)

Furthermore, by training a layer of multiple neurons with linear activation, the neurons converge to the first principle component of the input data [49]. As all neurons only learn the first principle component, a network of multiple layers in this setting seems useless. This implies that neurons within a network trained with Hebbian learning are rather uniform. However, independent neurons can encode more information and work better than dependent neurons [52]. Thus, a mechanism to encourage differentiation between neurons is needed. Differentiation between neurons can be achieved with several methods. Two well-known approaches are the winner-take-all competition (i.e. only the neuron with the most similar activity is selected for learning)[13] and a recurrent circuit that provides a competitive signal (i.e. the neurons compete with their neighbours to become active to learn). Another approach is to add a penalty to the learning rule. For example, anti-Hebbian learning is a method that adds a penalty for similarly active neurons and thus minimizes the linear dependency between neurons [53]. Others adapt the activation function of the neurons to enforce a specific activity distribution and to stabilize Hebbian learning in multilayer neural networks [54, 55]. Many other improvements to Hebbian learning are out of scope to be summarised in detail in this thesis.

[52]: Simoncelli et al. (2001)

13: in practice is k-winner-take-all often preferred where k instead of one neuron learns

[53]: Vogels et al. (2011)

[54]: Joshi et al. (2009)

[55]: Teichmann et al. (2015)

Similar to large parts of the brain, Hebbian learning is unsupervised and learns based on local information (i.e. neurons in close proximity). However, the brain is also largely recurrent and could guide neighbouring or preceding units. This assumption inspired supervised Hebbian learning. In supervised Hebbian learning, a subset of inputs which should evoke

post-synaptic activity can be selected. Supervised Hebbian learning can be extended to top-down and bottom-up learning [56], which leads to a combination of supervised and unsupervised Hebbian learning.

[56]: Grossberg (1988)

2.5.2 Hopfield Networks

Hopfield networks serve as associative (i.e. content-addressable) memory systems [57]. Such systems are particularly useful for retrieving representations based on degraded or partial inputs. Auto-associative memories return for a given input the most similar previously seen sample. A classical implementation of an auto-associative memory is the nearest neighbour algorithm [58]. This algorithm compares a given sample with the previously seen training data with a distance metric and returns the most similar sample[14]. Memory networks [59] implement an auto-associative memory with the deep learning framework. Such memory networks convert an input $\boldsymbol{x}$ to an internal feature representation $E(\boldsymbol{x})$. Afterwards, the memory $\boldsymbol{M}$ is updated by a memory-update function $G(\cdot)$ given the internal feature representation of the input $\boldsymbol{x}$.

[57]: Hopfield (1982)

[58]: Fix et al. (1989)

14: or the k most similar samples in the case of the k nearest neighbour (k-NN) algorithm

[59]: Weston et al. (2015)

$$\boldsymbol{M} = G(\boldsymbol{M}, E(\boldsymbol{x})) \tag{2.20}$$

Finally, the output $\hat{\boldsymbol{y}}$ is calculated with an output function $O(\cdot)$ based on the updated memory and the latent representation of $\boldsymbol{x}$:

$$\hat{\boldsymbol{y}}o = O(\boldsymbol{M}, E(\boldsymbol{x})) \tag{2.21}$$

This process is applied during the training and inference phase. The only difference is that the parameters for the functions $E(\cdot)$, $G(\cdot)$, and $O(\cdot)$ are only updated during training.

In a Hopfield network, on the other hand, no sequential layers exist, but all neurons are connected without self-connections, i.e. $w_{ii} = 0$. Furthermore, the weights are symmetrical $w_{ij} = w_{ji}$. A Hopfield network in its original form works only with binary units. For consistency, these networks are called binary Hopfield networks in the following. The output of a neuron in a binary Hopfield network depends on the output of the other neurons within the network:

$$a_i = \sum_{i \neq j} w_{ij} y_j + b \tag{2.22}$$

$$y_i = \begin{cases} 1, & \text{if } a_i > 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.23}$$

Hopfield networks have their own dynamics, and the output evolves over time. If a binary Hopfield network's initial value y_i has a different sign than a_i, the output will flip (i.e. change its sign). A flipping output influences all other neurons and may encourage them to flip as well. Since the signals are either positive or negative (i.e. binary), The term $y_i \cdot a_i$ is negative if y_i is not equal to a_i, otherwise, it is positive. Since the neuron flips if the term $y_i \cdot a_i$ is negative or stays the same if this term is positive, the change of this term can only be positive:

$$\Delta\left(y_i \cdot a_i\right) \geq 0 \tag{2.24}$$

The negative sum of the term $y_i \cdot a_i$ for the entire network is called the energy function $I(\boldsymbol{y})$[15] of the network:

$$I(\boldsymbol{y}) = -\sum_i y_i \left(\sum_{j>i} w_{ji} y_j + b \right) \tag{2.25}$$

As shown in equation (**??**), the network energy function $I(\boldsymbol{y})$ of the network can only decrease. Moreover, the energy function has a lower bound, and thus the network reaches a stable state after a finite number of iterations. A stable network pattern (i.e. no neurons flip their sign) is a local minimum of the energy function and is called a point attractor. An input $\boldsymbol{x}$ is fed into the network by initialising $\boldsymbol{y}$ accordingly. Afterwards, the binary neurons flip until a point attractor is reached. Thereby, a pattern is attracted to the closest stable pattern. Thus, the network can be used as an associative memory, as an input pattern is converted to the nearest stable pattern.

A binary Hopfield network with N neurons has a capacity[16] of $C = 0.138N$[60]. However, so far, it has only been discussed how patterns can be retrieved for a given input. Hebbian learning (c.f. Section 2.5.1) can be used to *store* specific patterns in a Hopfield network. To store a pattern, the weights $\boldsymbol{w}$ must be chosen in a way so that P desired patterns $(\boldsymbol{y}^1, \ldots, \boldsymbol{y}^P)$ are local minima of the energy function. By combining Hebbian learning with some smart mathematical transformations, it can be shown that the weights can be directly learned with only one iteration over the P training patterns[17]:

$$\boldsymbol{w} = \frac{1}{p} \sum_{k=1}^{P} \boldsymbol{y}^k \times (\boldsymbol{y}^k)^T - \boldsymbol{I} \tag{2.26}$$

where $\boldsymbol{I}$ is the identity matrix. Later, the weight update of the binary Hopfield network was extended so that the network can either learn patterns during an awake cycle or forget patterns during a sleep cycle [61].

One of the limiting factors of binary Hopfield networks is the capacity of $C = 0.138N$. The problem comes from the fact that the energy function is a quadratic function. More than three decades after the introduction of the binary Hopfield networks, Krotov and Hopfield [62] reformulated the energy function as a polynomial function to get polynomial capacity $C \approx N^{a-1}$ where a is the order of the polynomial function. Later, the energy function was reformulated as an exponential function [63] and thus modern Hopfield networks have an exponential capacity of $C \approx 2^{\frac{N}{2}}$.

The second limiting factor of binary Hopfield networks is that only binary patterns can be stored. Recently, Hopfield networks have been extended to continuous patterns by reformulating the energy function and the corresponding update rule [64]. Continuous Hopfield networks

15: typically, the energy function is called $E(\cdot)$ and not $I(\cdot)$; however, $E(\cdot)$ is already used as encoder function in this thesis and therefore $I(\cdot)$ is used as energy function.

16: the number of patterns that can be stored
[60]: McEliece et al. (1987)

17: for the derivation of this equation refer to [57]

[61]: Hopfield et al. (1983)

[62]: Krotov et al. (2016)

[63]: Demircigil et al. (2017)

[64]: Ramsauer et al. (2021)

can retrieve continuous patterns or a combination of several similar continuous patterns. The authors claim that continuous Hopfield networks can replace fully-connected layers, attention layers [65], LSTM layers [66], support vector machines (SVM) [67], and k nearest neighbour (k-NN) [68].

2.5.3 Spiking Neural Networks

Biological neurons emit time-dependent spikes (c.f. Section 2.1). To transmit information, especially the firing rate (i.e. the number of spikes per second in Hz) and precise timing of the spikes are relevant. The amplitude and duration of the spike only matter a little. So-called Spiking neural networks (SNNs) incorporate the concept of time into a computational model. SNNs do not transmit information in each forward pass but rather transmit a signal when the membrane potential reaches a threshold value[18]. The neuron fires as soon as the threshold is reached, thereby influencing other neurons' potential. The most prominent model of a spiking neuron is the leaky integrate-and-fire (LIF) neuron [69]. The LIF neuron models the membrane potential with a differential equation. Incoming spikes can either increase or decrease the membrane potential. The membrane potential either decays over time or is reset to a lower value if the threshold value is reached and the neuron has fired. There exist different integrate-and-fire (IF) neurons models such as the Izhikevich quadratic IF [70] or the adaptive exponential IF [71]. While each model has different mathematical properties, the concept remains the same: Each model of a neuron has a membrane potential that is increased or decreased through spikes from other neurons and decays over time or is reset by emitting a spike on its own.

18: the membrane potential is related to the electrical charge of the membrane of a biological neuron

[69]: Abbott (1999)

[70]: Izhikevich (2003)

[71]: Brette et al. (2005)

Biological neurons have different dynamics. Some neurons regularly fire if they receive an input current, while others slow down the firing rate over time or emit bursts of spikes. Modern models of spiking neurons can recreate this behaviour of biological neurons [72].

[72]: Paugam-Moisy (2006)

The synaptic plasticity can be modelled with an adapted version of Hebbian learning (c.f. Section 2.5.1). The spike-timing dependent (STDP) plasticity rule [73] distinguishes the firing behaviour of pre-synaptic and post-synaptic neurons. If the pre-synaptic neurons fire before the post-synaptic neuron, the connection is strengthened; otherwise, it is weakened.

[73]: Bi et al. (2001)

For a long time, SNN only worked for very shallow networks. In 2018, Kheradpisheh et al. [74] proposed an SNN based on the idea of CNNs called a deep spiking convolutional network. This network uses convolutional and pooling layers with IF neurons instead of classical artificial neurones and is trained with STDP. First, the image is fed into cells with a difference-of-Gaussian (DoG) function. DoG is a feature enhancement algorithm that subtracts a Gaussian blurred version of an image from the original image. Thereby, positive or negative contrast is detected in the input image. The higher the contrast, the stronger a cell is activated and the earlier it emits a spike. Thus, the order of the spikes depends on the order of the contrast. These spikes are forwarded to a convolutional layer. Deep spiking convolutional networks use two types of LIF neurons: On-center neurons fire when a darker area surrounds a bright area, and

[74]: Kheradpisheh et al. (2018)

off-centre neurons do the opposite. Convolutional neurons emit a spike as soon as they detect their preferred visual feature[19]. Neurons that fire early perform the STDP update with a winner-takes-all mechanism. Thus, the neurons within a layer compete with each other and those which fire earlier learn the input pattern. This mechanism prevents other neurons from firing and guarantees a sparse connection. Later convolutional layers detect more complex features by integrating input spikes from the previous layer. The features from the last convolutional layer are flattened, and a support vector machine is used to classify the features.

19: the feature's location is irrelevant as convolution layers are translation invariant

2.5.4 Reservoir Computing

As described in Section 2.4, biological neurons are highly dynamic while artificial neurons are not. Another type of model that introduces such dynamics is based on reservoir computing. Reservoir computing is an umbrella term for networks based on the concepts of echo-state networks (ESN) [75] and liquid state machines (LSM) [76]. A reservoir is a fixed non-linear system that maps an input vector $\boldsymbol{x}$ to a higher dimensional computation space. After the input vector is mapped into the computation space, a simple readout mechanism is trained to return the desired output based on the reservoir state. In principle, the system should be capable of any computation if it has a high enough complexity [77]. However, only some systems are suited as a reservoir. A good reservoir system distributes different inputs into different regions of the computation space [77].

[75]: Jaeger (2001)

[76]: Maass et al. (2002)

[77]: Konkoli (2018)

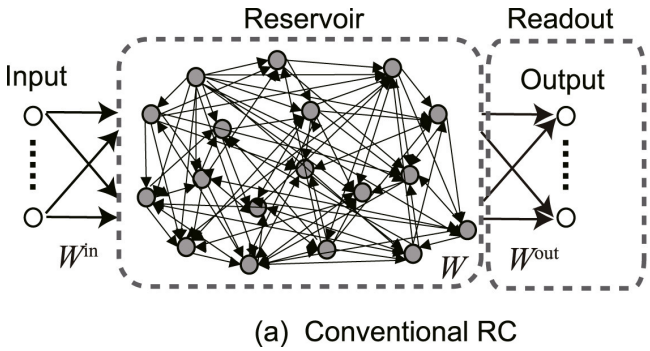

Figure 2.4: Structure of an echo Ssate network. The image is from Tanaka et al. [78].

An ESN is a set of sparsely connected recurrent neurons as visualized in Figure 2.4. The reservoir consists of N nodes which are connected according to an Erdős–Rényi graph model[20] [79]. This graph model is represented by an adjacency matrix $\boldsymbol{W}$ of size $N \times N$. The time varying input signal $\boldsymbol{x(t)}$ is mapped to a sub-set of N/M graph nodes by multiplying it with $\boldsymbol{W}_{\text{in}} \in \mathbb{R}^{N \times M}$ and the output by multiplying the reservoir state with $\boldsymbol{W}_{\text{out}} \in \mathbb{R}^{M \times N}$. Interested readers are referred to [80] to learn more about the mathematical properties and how the network is updated in detail.

20: the Erdős–Rényi model is a model for generating random graphs where all graphs on a fixed set of vertices and edges are equally likely

[79]: Erdös et al. (1959)

[80]: Lukoševičius (2012)

In the original form of ESN, only the readout weights are learned, and the rest is chosen randomly. The input $\boldsymbol{x(t)}$ brings the recurrent units in an initial state. The recurrent connections inside the reservoir create different dynamics in the network. The readout neurons linearly transform the recurrent dynamics into temporal outputs. The readout weights $\boldsymbol{W}_{\text{out}}$ are trained to reproduce a target function $\boldsymbol{y(t)}$.

Liquid State machines use a spiking neural network instead of a graph of recurrent units as reservoirs. The nodes of the spiking neural network are randomly connected. Thus, every node receives time-varying inputs from the inputs and other nodes. The recurrent connections turn the varying input into a spatiotemporal pattern. Similar to ESN, the spatiotemporal activation patterns are read out by a linear layer.

In general, reservoirs are universal approximators and can approximate any non-linear function, given that there are enough neurons in the reservoir. They generalize better and faster than equivalent MLP. Current systems' main drawback is that they cannot deal well with high-dimensional inputs such as images.

Related Work 3

TODO: Teile auf in related work (zu Beginn) + optionales Kapitel mit weiteren relevanten Themen zu Natural Intelligence, die aber nicht wirklich related work sind sondern eher ein Survey

3.1 Natural Intelligence

This thesis is inspired by the work "A Theory of Natural Intelligence" from von der Malsburg et al. [2]. Therefore, summarize their work in detail in the following.

[2]: von der Malsburg et al. (2022)

According [2], the process of learning is influenced by "nature", "nurture", and "emergence"[1]. They point out that human genome (as of nature) only contain 1GB of information [81] and humans only absorb a few GB into permanent memory over a lifetime (as of nurture) but it requires about 1PB to describe the connectivity in human brain. Therefore, it is important to distinguish the amount of information to describe a structure from the amount of information needed to generate it. Similar, nature and nurture only require a few GB to construct, respectively instruct the entire human brain. Therefore, they argue that the human brain must be highly structured (i.e. nature and nurture "generate" the human brain by selecting from a set pre-structured patterns). The authors call the process of generating the highly structured network in the human brain the "Kolmogorov [82] Algorithm of the Brain"[2]. Network self-organization is the only mechanism that has not yet been disproved by experiments as the brains Kolmogorov algorithm [83, 84]. This mechanism loops between activity and connectivity, with activity acting back on connectivity through synaptic plasticity until a steady state, called an attractor network, is reached. The consistency property of an attractor network means that a network has many alternative signal pathways between pairs of neurons [85]. Thus, the brain develops as an overlay of attractor networks called net-fragments [86]. Net-fragments consist of small sets of neurons, whereby each neuron can be part of several net fragments. The network self-organization has to start from an already established coarse global structure which is improved in a coarse-to-fine manner to avoid being caught in a local optima.

1: nature refers to the influence of genes and evolution, nurture to the influence of experience and education

[81]: McPherson et al. (2001)

[82]: Kolmogorov (1998)

2: as the Kolmogorov complexity describes the number of bits required by the shortest algorithm that can generate the structure

[83]: Willshaw et al. (1976)

[84]: Willshaw et al. (1979)

[85]: Malsburg et al. (1987)

[86]: Malsburg (2018)

Also, von der Malsburg et al. [2] discuss scene representation (i.e. how a scene is represented in the brain) even tough they point out that this is a contested concept [87]. Scene representation is a organization framework to put abstract interpretation of scene layouts, elements, potential actions, and emotional responses in relation. The details are not rendered as in photographic images but the framework supports the detailed reconstructions of narrow sectors of the scene. The basic goal if learning is to integrate a behavioral schema into the flow of scene representations. They propose the hypothesis that the network structure resulting from self-organization together with the neural activation in

[87]: Freeman III et al. (1990)

the framework of scene representation are the inductive bias that tunes the brain to the natural environment.

Finally, they discuss how net fragments can be used to implement such structures and processes using vision as an example. They point out that a neuron is grouped in one or multiple net fragments through network self-organization. The net fragments can be considered as filters that detect previously seen patterns in the visual input signal. An object is represented by multiple net fragments, where each fragment responds to the surface of that object and has shared neurons and connections with other net fragments representing that object. Thus, net fragments render the topological structure of the surfaces that dominate the environment. Von der Malsburg et al. [2] propose that net fragments represent shape primitives which can adapt to the shape of actual objects[3]. Shifter circuits are one possible implementation of networks that enable invariant responses to the position- and shape-variant representations [88, 89]. They are composed of net-fragments that can be formed by network self-organization [90]. Ref. [2] also argue that net fragments are the compositional data structure used by the brain. A hierarchy of features may be represented by nested net fragments of different size. Complex objects, such as mental constructs, can thus be seen as larger net fragments composed as mergers of pre-existing smaller net fragments.

3: adapt in spite of metric deformations, depth rotation, and position

[88]: Arathorn (2002)

[89]: Olshausen et al. (1995)

[90]: Fernandes et al. (2015)

Von der Malsburg et al. [2] do not address how their interesting theoretical concepts can be implemented in a mathematical model. However, a concrete implementation was done by Claude Lehmann in the form of a Master's thesis [91]. He proposes a new layer called the laterally connected layer (LCL). The LCL layer extends convolutional layers by forming lateral intra-layer connections based on the Hebbian learning rule (c.f. Section Section 2.5.1). Similar to a convolutional filter, the convolutional feature map is calculated. Afterwards, the convolutional feature maps are compared and the lateral impact between feature maps is calculated (i.e. the covariance between the feature maps). When two feature maps are similar in the same pixel locations, their connection strength is increased. By using Hebbian learning, lateral connections are formed between the feature maps with a high lateral impact. Thus, new filters are formed based on existing filters with a high covariance. Lehmann found that LCL layers increase robustness for object recognition. He shows on the MNIST dataset [18] that for a small reduction in accuracy of 1%, the performance on corrupted images increases by up to 21% and that it works especially well for noisy types of corruptions. However, this layers improved performance only on very small networks.

[91]: Lehmann (2022)

In this thesis, the ideas from the work of von der Malsburg et al. [2] are incorporated into a deep learning model. Thus, this work is considered to be *the* inspiration for this thesis. However, this thesis is based on other concepts than the thesis by Lehmann [91] and therefore has little in common with his work.

3.2 Self-Organization

Self-organization is the process by which systems consisting of many units spontaneously acquire their structure or function without interference

from a external agent or system. They organize their global behavior by local interactions amongst themselves. The absence of a central control unit allow self-organizing systems to quickly adjust to new environmental conditions. Additionally, such systems have in-built redundancy with a high degree of robustness as they are made of many simpler individual units. These individual units can even fail without the overall system breaking down.

In nature, groups of millions units that solve complex tasks by using only local interactions can be observed. For example, ants can navigate difficult terrain with a local pheromone-based communication and thus form a collective type of intelligence. Such observations inspired researchers to build algorithms which are based on local communication and self-organization, for example ant colony optimization algorithms [92]. DeepSwarm [93] is a neural architecture search method that uses this algorithm to search for the best neural architecture. This methods achieves competitive performance on rather small datasets such as MNIST [18], Fashion-MNIST [94], and CIFAR-10 [95].

[92]: Dorigo et al. (1997)

[93]: Byla et al. (2020)

Cellular Automata mimic developmental processes in multi-cell organisms. They contain a grid of similar cells with an internal state which is updated periodically. The transition from a given state to a subsequent state is defined by some update rules. During an update, cells are only allowed to communicate with the neighbouring cells. Thus, self-organization is enforced by the definition of the update rules. Such automata can be used to study biological pattern formations [96] or physical systems [97]. Neural Cellular Automata [98] use neural networks to learn the update rule. The input in such a neural network is the state of a given cell and its neighbours, the output the subsequent cell state. Usually, the same network is applied to all cells. In this case, a fully connected neural network which is applied to each cell and its local neighbours can be reformulated as a CNN[99]. NCAs can be trained efficiently with gradient descent to grow given 2D patterns such as images[100, 101]. These images are grown through self-organisation (i.e. the pixels pick a color based on the color of neighboring pixels) and are surprisingly resistant to damage. For example, large parts of the images can be removed and the system is able to rebuild these pixels[4]. However, the aforementioned approaches can only grow the pattern they were trained on. A recent method called Variational Neural Cellular Automata [103] use an NCA as decoder of a Variational Autoencoder [104]. This probabilistic generative model can grow a large variety of images from a given input encoded in a vector format. However, there is still a big gap in performance compared to state-of-the-art generative models. Besides growing 2D patterns, NCAs can also create 3D patterns such as buildings in the popular video game Minecraft by utilizing 3D CNNs [105] or generate structures with specific function such as simulated robots able to locomote[106]. Moreover, self-assembling approaches based on NCAs are not restricted to grid-structures. NCAs can be generalized to graph neural networks [107]. Graph cellular automata (GCA) use graph neural networks [108] instead of CNNs to learn the state transition rules and can thus deal with more complex pattern structures than just 2D and 3D grids. The process of growing images from cells of an NCA can also be inverted. Randazzo et al. [109] propose to use NCA to classify given structures such as images. They apply the same network to each pixel

[96]: Wolfram (1984)

[97]: Vichniac (1984)

[98]: Wulff et al. (1992)

[99]: Gilpin (2019)

[100]: Mordvintsev et al. (2020)

[101]: Mordvintsev et al. (2022)

4: a demo of this regeneration process is available at [102]

[103]: Palm et al. (2022)

[104]: Kingma et al. (2014)

[105]: Sudhakaran et al. (2021)

[106]: Horibe et al. (2021)

[107]: Grattarola et al. (2021)

[108]: Zhou et al. (2021)

[109]: Randazzo et al. (2020)

and its neighbours of an image. In an iterative process based on local communication, the image fragments then agree on which object they represent. NCAs can even be used to control reinforcement learning (RL) agents. Variengien et al. [110] use the observations of the environment as state of the NCA, the subsequent state predicted by the NCA are used as Q-value estimates of a deep Q-learning algorithm [111].

[110]: Variengien et al. (2021)

[111]: Mnih et al. (2013)

Self-organization can not only be used to generate structures but also to optimize the weights of a neural networks over the agents lifetime. For example, a Hebbian learning rule for meta-learning can be used to self-organize the weights of a RL agent over his lifetime[112]. This means that across multiple episodes the weights of a Hebbian based model are learned. The weights of the agents policy are reset in every episode and the Hebbian based model is used to update them. This allows the agent to adapt better to the changed conditions within the environment.

[112]: Najarro et al. (2020)

Besides optimizing the weights, self-organization has also been used to change the learning rule itself. The method "Evolve and Merge" [113] uses the so called "ABCD" Hebbian learning rule which updates the weights as follows:

[113]: Pedersen et al. (2021)

$$\Delta w_{ij} = \alpha(Ao_io_j + Bo_i + Co_j + D) \tag{3.1}$$

α is the learning rate, o_i and o_j are the activity levels of connected neurons and A, B, C, and D are learned constants. For each connection in the network is one learning rule initialized and the constants are learned. After a pre-defined number of epochs, the learning rules are clustered and the ones with similar constants are merged. By repeating this process, the number of parameters can be reduced and robustness increases according to the authors.

Alternatively, it is also possible to initialize the network with shared parameters instead of starting with many rules and merging them over time. Kirsch and Schmidhuber [114] use multiple tiny recurrent neural networks (RNNs) that have the same weight parameters but different internal states[5]. By using self-organization and Hebbian learning, they show that it is possible to learn powerful learning algorithms such as backpropagation while running the network in forward-mode mode only. However, it works only for small-scale problems as it can get stuck in local optima. In general seem self-organizing systems to be hard to optimize and only to work for small datasets or simple problems so far.

[114]: Kirsch et al. (2021)

5: Intuitively, these tiny RNNs can be interpreted as more complex neurons.

Risi [115] describes why self-organizing systems are hard to train; First, the system is hard to control because there is no central entity in charge but the system must still be nudges into the right direction. Second, self-organizing systems are unpredictable (i.e. there exist no mathematical model that tells the outcome of the self-organizing process).

[115]: Risi (2021)

3.2.1 Growing Networks

Unsupervised learning techniques usually map high dimensional input data to a lower-dimensional representation. One approach to do so are self-organizing maps (SOM) [116, 117]. They map the input data to a discretized representation of the input space of the training samples,

[116]: Kohonen (1982)

[117]: Kohonen (1989)

called a map. In opposite to ANNs, they use competitive learning instead of error correction learning (i.e. back-propagation with gradient descent). A weight vector is used to map the data to a node in the mapping field. The datapoints "compete" for the weight vectors. The weight vector of a node in the map that best matches a datapoint is moved closer to that input, as are nodes that are in the neighbourhood. By doing so, samples that are close in the input space are also closed in the resulting maps.

However, SOM have two major limitations; First, the network structure must be pre-defined which constraints the result mapping accuracy. Second, the capacity of the map is predefined through the number of nodes. Growing networks are able to overcome this limitations. Growing networks add nodes or whole layers of nodes into the network structure at the positions of the map where the error is highest. Many growing networks [118–120] add such units after a fixed number of iterations in which the error is accumulated. After adding a unit, it takes several iterations to accumulate the error again until the next node can be added.

[118]: Fritzke (1994)

[119]: Reilly et al. (1982)

[120]: Fritzke (1994)

Grow When Required (GWR) networks [121] use a different criterion to add nodes. Instead of adding nodes to support the node with the highest error, nodes are added when a given input samples cannot be matched with the current nodes by some pre-defined accuracy. This allows the network to adapt the growing process rather fast; The networks stops growing when the input space is matched b the network with some accuracy and the networks starts growing again if the input distribution changes.

[121]: Marsland et al. (2002)

Such GWR networks can be used to build self-organizing architectures. For example, Mici et al. [122] build a self-organizing architecture based on GWR to learn human-object interactions from videos. They use two GWR in parallel, one to process feature representations of body postures and another to process manipulated objects. A third GWR is used to combine these two streams and to create action–object mappings in a self-organized manner. By doing so, they are able to learn human-object interactions and exhibit a model which is more robust to unseen samples than comparable deep learning approaches.

[122]: Mici et al. (2018)

3.2.2 Self-Organization in Spiking Neural Networks

Spiking neural networks (SNNs) (c.f. Section Section 2.5.3) communicate through binary signals known as spikes and are very efficient on special event-based hardware[123]. There exist several methods to self-organize such architectures. For the sake of completeness, two well-known approaches are described in the following. However, since this thesis focuses on self-organization in deep learning systems, these approaches are only roughly described and for detailed explanations please refer to the respective literature.

[123]: Davies et al. (2018)

Similar to deep learning, there exists a multitude of different network architectures; Shallow [124, 125] and deep networks [126, 127] structures, fully connected [128] and convolutional layers [129, 130], as well as based on different learning rules such as supervised [131, 132], unsupervised [128, 133] and reinforcement learning based [134].

[124]: Masquelier et al. (2007)

[125]: Yu et al. (2013)

[126]: Kheradpisheh et al. (2018)

[127]: Mozafari et al. (2019)

[128]: Diehl et al. (2015)

[129]: Cao et al. (2015)

[135]: Raghavan et al. (2020)

A representable method for self-organization in SNNs is proposed by Raghavan et al. [135]. They introduce a stackable tool-kit to assemble multi-layer neural networks. This tool-kit is a dynamical system that encapsulates the dynamics of spiking neurons, their interactions as well as the plasticity rules that control the flow of information between layers. Based on the input, spatio-temporal waves are generated that travel across multiple layers. A dynamic learning rule tunes the connectivity between layers based on the properties of the waves tiling the layers[6].

6: for more information please refer to [135]

[136]: Raghavan et al. (2019)

[70]: Izhikevich (2003)

An alternative method proposed by Raghavan and Thomson [136] grows a neural network. They start with a single computational "cell" and use a wiring algorithm to generate a pooling architecture in a self-organizing fashion. The pooling architecture emerges through two processes; First, a layered neural network is grown. Second, self-organization of its inter-layer connections is used to form defined "pools" or receptive fields. They us the Izikhevich neuron model [70] in the first layer to generate spatio-temporal waves. The units in the second layer learn the "underlying" pattern of activity generated in the first layer. Based on the learned patterns, the inter-layer connections are modified to generate a pooling architecture[7].

7: for more information please refer to [136]

[137]: Vaila et al. (2019)

[74]: Kheradpisheh et al. (2018)

8: DoG filter can thus be used to reduce noise and to detect edges

In general, SNNs have to "convert" static input data such as images to a dynamic signal. For example, images are often converted to such signals by using Difference of Gaussian (DoG) convolution filters [74, 137]. Such filters subtract one Gaussian blurred version of an original image from another, less blurred version[8]. This subtraction results in spikes for each pixel. To encode the filter output into a temporal signal, bigger spikes are forwarded earlier in time than smaller spikes. However, such approaches lose a lot of information about the input. For example, in the process described above are all information about color and thin structures lost. To the author of this thesis, this seems to be the reason why these SNNs can't match the performance of deep learning algorithms so far and often only work well for small gray-scale image-datasets such as MNIST [18].

3.2.3 Relevance

[2]: von der Malsburg et al. (2022)

Self-organization is, according to von der Malsburg et al. [2], one of the key elements for natural intelligence. Therefore, this concept of local optimization and interaction plays a very important role in this thesis. Specifically, two types of self-organization are proposed in this thesis; Vertical self-organisation is based on layer-wise learning. In this approach, the layers of a neural network are considered separate units and are updated independently. The communication between these units takes place by feeding data sequentially through the layers. The second type of self-organization is vertical self-organisation. In this approach, data is divided into patches and then analysed by independent networks. Thus, small networks form the self-organising units and communicate with each other by adding the output of one network to the input of another network. These two types of self-organization are described in more detail in Chapter **??**.

3.3 Alternative Training Algorithms

Neural networks, especially deep neural networks, are usually optimised by backpropagation of errors (c.f. Section Section 2.2). Soon after backpropagation was published, it was question whether this algorithm is suitable to explain the learning in the brain [138, 139]. Besides the fact that backpropagation seems no biologically plausible and is responsible for many of the limitations described in Section Section 2.3, it also has technical shortcomings. First, gradients might vanish or explode when propagated through too many layers [140]. Second, it turns the networks into "black boxes" and prevents the users from getting useful insights from trained models[9]. Training end-to-end does not allow to control the effect of the loss function on a hidden layer because of the non-linearity in the network [141]. Finally, the loss landscapes are non-optimal which might lead to slow convergence and the possibility that the reached local minimum is suboptimal [17].

[138]: Crick (1989)

[139]: Grossberg (1987)

[140]: Zhang et al. (2020)

9: training a model a layer-wise allows modularization, i.e. divide an conquer

[141]: Lee et al. (2015)

[17]: Ioffe et al. (2015)

Hebbian learning (c.f. Section Section 2.5.1), on the other hand, is widely accepted in the fields of neurocomputing, psychology, neurology, and neurobilogy [142]. Hebbian mechanisms are for example sufficient for topographic mappings[10] [117, 143], neuroplasticity[11] [144], or recognizing non-linear patterns [145], but also have limitations [146]. Hebb's learning rule is insufficient as general rule and is limited temporally as it requires (almost) synchronous stimuli [147][146]. However, many signals such as motor control problems, are sequential and Hebbian learning has therefore to be combined with additional memory mechanisms [148].

[142]: Widrow et al. (2019)

10: mapping input data to a discretized representation

[117]: Kohonen (1989)

[143]: Grajski et al. (1990)

11: the networks ability to reorganize itself by forming new connections

[144]: Montague et al. (1991)

[145]: Mel (1992)

[146]: Anderson (1998)

[147]: Rumelhart et al. (1987)

[148]: Grossberg et al. (1989)

There exists many alternative learning functions to these two well known algorithms that might overcome the aforementioned limitations. A group of algorithms that do not require end-to-end forward or backward pass is called *Proxy Objective*. These algorithms use a proxy objective function for each layer. While backpropagation calculates a global loss after the forward pass, this strategy uses a proxy function for each layer. The loss function of the first Layer L_1 (i.e. the proxy objective of the first layer) influence only the trainable parameters of the first layer θ_1 but not the parameters of the second layer θ_2. This allows to decouple the layers. The different instances of *Proxy Objective* methods mainly differ in the proxy objective function. However, the idea behind the proxy objective function is always the same; namely to characterize the separability of the hidden representations. Minimizing L_1 encourages the first layer to improve the separability of its output representations, making the classification problem simpler for the subsequent layer. The separability between hidden representation vectors can be measured by distance/similarity metrics as done in [149, 150]. Wang et al. [151] propose an auxiliary neural network to map the hidden representation to another feature space before computing their distance/similarity. This allows to choose the dimensionality of the feature space in which the proxy objective function is computed regardless of the given network layer dimension. Others [152–154] propose an additional network with either fixed or trainable weights as proxy function and used this auxiliary network's accuracy to quantify data separability. Nokland and Eidnes [155] combine the two aforementioned approaches by using two auxiliary networks; one network is trained with a "similarity matching loss" and the other with a cross-entropy loss (i.e. to quantify separability with an auxiliary classifier

[149]: Duan et al. (2022)

[150]: Duan et al. (2020)

[151]: Wang et al. (2021)

[152]: Belilovsky et al. (2019)

[153]: Mostafa et al. (2018)

[154]: Marquez et al. (2018)

[155]: Nokland et al. (2019)

accuracy). A major issue of the first *Proxy Objective* methods is that the models are trained layer-wise[12]. The resulting computational complexity can be reduced with synchronous or asynchronous training. In the synchronous setting is first a forward pass through all layers performed and afterwards are all layers trained simultaneously by minimizing their own proxy objective function [152]. In the asynchronous settings are all layers trained simultaneously by using a replay buffer for each layer that stores the layers output [156]. Each layer receives its input from such a buffer instead of the previous layer what eliminates the need for an end-to-end forward pass.

12: when layer k is trained, the layers $1, \ldots, k-1$ are frozen and the layers $k+1, \ldots, n$ are not updated

[156]: Belilovsky et al. (2020)

[157]: Hinton (2022)

Recently, Hinton [157] introduced the *Forward-Forward (FF) Algorithm* that uses a proxy objective functions over multiple time-steps. This algorithm uses two forward passes, one with positive (i.e. real) data and one with negative data. Each layer has its own objective function which is to have a high "goodness" for positive data and a low "goodness" for negative data. The goodness is measured by the sum of the squared neural activities. The goal of the learning process is to make the goodness be above some threshold value for real data and below that threshold for negative data. More specifically, the aim is to correctly classify input vectors as positive data or negative data. The probability that an input vector is positive $p(positive)$ is given by applying the logistic function σ to the goodness minus the threshold θ

$$p(positive) = \sigma\left(\sum_j y_j^2 - \theta\right) \tag{3.2}$$

where y_j is the output of a hidden unit. To perform a classification task, the label is included in the input. Positive data consists of an input vector (e.g. an image) and the correct label, negative data consists of an input vector with the wrong label. The goal is then to have a high goodness in each layer for the positive data and a low goodness for the negative data. The only difference between positive and negative data is the label and thus the *FF* algorithm only learns features that correlate with the label. During inference, the data is fed with each label through the network and the one with the highest goodness is kept as prediction. One major weakness is that one layer at a time is learned and later layers cannot affect what is learned in earlier layers. This limitation can be overcome by treating the static input vector as a sequence (i.e. use the same input for multiple steps) and by using a multi-layer recurrent network [158]. The algorithm still runs forward in time but the activity vector at each layer is determined by the activity vector of the previous layer and the activity vector of the subsequent layer at the previous time-step.

[158]: Hinton (2021)

Target Propagation methods are a group of algorithms that require an end-to-end forward pass but not an end-to-end backward pass. These algorithms compute targets[13] rather than gradients at each layer. Similar to gradients, the targets are propagated backward through the network. While the output layer obviously uses the true label as its target (i.e. the goal is to predict the label), the targets of previous layers are found sequentially (from the output layer to the input layer). Good targets are those that minimize the loss in the output layer if they are realized in the forward pass. The simplest solution would be to apply the inverse function of each layer to propagate the target backward. If h_l are the

13: a target can be interpreted as the value that should be predicted by a layer

activations and θ_l the parameters of layer l, we can define the forward pass as:

$$h_l = f(h_{l-1}; \theta_l) \tag{3.3}$$

The inverse function that yields the target activation $\hat{h}_l$ would then be:

$$\hat{h}_l = f^{-1}(\hat{h}_{l+1}; \theta_{l+1}) \tag{3.4}$$

As mentioned before, the target of the final layer $\hat{h}_L$ is the loss-optimal output activation such as the correct label distribution. However, advanced neural networks are usually not invertible and approximate inverse transformations have to be learned with decoders

$$g(h_{l+1}; \lambda_{l+1}) \approx f^{-1}(h_{l+1}; \theta_{l+1}) \tag{3.5}$$

where λ_l are the trainable parameters of the decoder at layer l. Thus, the target activation can be approximated by an encoder:

$$\hat{h}_l \approx g(\hat{h}_{l+1}; \lambda_{l+1}) \tag{3.6}$$

Different instances of *Target Propagation* methods mainly differ in the way the the targets are generated and/or the loss function of the decoder L_g. Vanilla Target Propagation [159] directly derives the target from the decoder $\hat{h}_l = g(\hat{h}_{l+1}; \lambda_{l+1})$ and trains the decoder $g()$ by minimizing

[159]: Bengio (2014)

$$L_g = \|g(h_{l+1} + \epsilon; \lambda_{l+1}) - (h_l + \epsilon)\|_2^2 \tag{3.7}$$

where ϵ is some added Gaussian noise to enhance generalization. It is obvious that this loss functions encourages the decoder $g()$ to learn the inverse of $f()$ by minimizing the difference between the activations from the forward-pass h_l of layer l and the decoder's output (based on the activations of the subsequent layer h_{l+1}). The targets are predicted by feeding the *target activations* $\hat{h}_{l+1}$ through the decoder[14].

14: note that these are two different kind of activations: The inverse function is learned based on the activation of the forward pass h_l, the target activation is predicted based on the loss-optimal output activation $\hat{h}_L$

Later, a major improvement was to extend the decoder $g()$ of Vanilla *Target Propagation* with a correction term that enables a more robust optimality guarantee for bigger networks [160]. This method is known as *Difference Target Propagation* and extends Equation (**??**) as follows:

[160]: Lee et al. (2015)

$$\hat{h}_l = g(\hat{h}_{l+1}; \lambda_{l+1}) + [h_l - g(h_{l+1}; \lambda_{l+1})] \tag{3.8}$$

The extra term $[h_l - g(h_{l+1}; \lambda_{l+1})]$ is to correct errors of the decoder in estimating the inverse. *Difference Target Propagation* has been further improved by novel loss functions L_g for the decoder. These methods are known as *Difference Target Propagation with Difference Reconstruction Loss* and *Direct Difference Target Propagation* [161]. However, since the concepts are roughly the same as for *Difference Target Propagation* (despite adding some terms to the loss function), I do not summarize these methods in more detail here. An advantage of calculating targets is that only layer-wise gradients are required what allows models to have

[161]: Meulemans et al. (2020)

non-differentiable operations [160]. A major drawback is that auxiliary models have to be trained to calculate the hidden targets $\hat{h}_l$ what might overweight the savings achieved by eliminating a full backward pass.

[162]: Jaderberg et al. (2017)

[163]: Czarnecki et al. (2017)

Another group of alternative learning algorithms are known as *Synthetic Gradients* [162, 163]. *Synthetic Gradients* methods replace the gradients used in the backward pass by approximating local gradients with auxiliary models and utilize gradient-based optimization algorithms such as SGD to update the weights locally. The auxiliary models are usually fully-connected networks that are trained to regress a layer's gradients when given the layer's activations. End-to-end backwards passes are only performed occasionally to acquire real gradients that can be used to trained the gradient approximation models. Thus, the frequency of end-to-end backward passes is reduced. When auxiliary input models are used to predicts the layer's input, the frequency of the forward pass can be reduced as well in the same manner as for the backward pass [162]. It is also possible to get rid of the backward pass completely by training the auxiliary gradient models with local information only [164]. However, this works significantly worse than when real gradients are used. An advantage of *Synthetic Gradients* methods is that they can be used to approximate backpropagation through time (BTT) for an unlimited number of steps [162]. It has been shown that this allows more efficient training for learning long-range dependencies compared to BTT.

[164]: Lansdell et al. (2020)

[170]: Bartunov et al. (2018)

[166]: Lillicrap et al. (2016)

[169]: Liao et al. (2016)

Many other methods are motivated purely by biological plausibility. Some examples are [165–169]. However, these biological plausible methods have been significantly outperformed by backpropagation of error on meaningful benchmark datasets [170]. In the following *Feedback Alignment* methods are discussed as they seem to be the most popular group of biological plausible alternatives to backpropagation. From a biological point of view, one of the major criticisms of backpropagation is the "weight transportation problem" [166]; it is believed that the human brain doesn't have a backward pass where an error signal is passed to previous layers. *Feedback Alignment* algorithms use fixed, random weights during the backward pass [166]. Thus the symmetry between the weights used during the forward pass and the backward pass is broken. Later, the algorithm was improved by using fixed, random weights that share the signs with the actual weights of the network [169].

Lastly, *Auxiliary Variables* are another group of learning algorithms [171–174]. These methods use the idea of variable splitting, i.e. transform a complicated problem into a simpler one by introducing additional trainable variables. Introducing auxiliary variables may pose scalability issues, and these methods require special, usually tailor-made solvers. Therefore, such methods are not reviewed in more detail in this thesis.

[175]: Duan et al. (2022)

[170]: Bartunov et al. (2018)

According to [175] and [170], only proxy objective functions have achieved competitive performance with large models on large datasets such as ImageNet [176]. The other approaches work only on smaller datasets such as MNIST [18] or CIFAR-10 [95] so far.

[2]: von der Malsburg et al. (2022)

End-to-end backpropagation is not compatible with local self-organisation which is one of the key concepts of natural intelligence according to von der Malsburg et al. [2]. With end-to-end backpropagation, systems are trained as a single unit and consequently have no independent local units that can organise themselves. Therefore, alternative learning algorithms

is very relevant to this thesis. In fact, the approach based on vertical self-organisation uses proxy objective functions to train the layer locally as independent units (c.f. Chapter **??**). Proxy objective functions are used as these are the only alternative learning algorithm known that scale well on large neural networks and large datasets.

3.4 Visual Representation Learning

One of the earliest methods of learning visual representations are *Autoencoders*. *Autoencoders* have been introduced 1985 [30] and learn an encoder and a decoder function [177]. The encoder A maps the input from a high-dimensional space to a lower dimensional embedding space $A : \mathbb{R}^n \rightarrow \mathbb{R}^e$ and the decoder B reverses this mapping $B : \mathbb{R}^e \rightarrow \mathbb{R}^n$. Typically, neural networks are used to learn the encoder function A and decoder function B by minimizing a reconstruction loss [178]. In order that the functions A and B are not just the identity operators, some regularization is needed. One of the simplest regularization methods is to introduce a bottleneck, i.e. to compress the representation with the encoder while still being able to re-create the original input as good as possible with the decoder. With a bottleneck layer, the number of neurons is limited. An alternative (or a supplement) is to limit the number of activations by enforcing sparsity. Autoencoders with such a sparsity constraint are also known as *Sparse Autoencoders*. A sparsity constraint can be imposed with L_1 regularization or a kullback-leibler (KL) divergence between expected average neuron activation and an ideal distribution [179]. Other popular versions of *Autoencoders* are *Denoising Autoencoders* [180], *Contractive Autoencoders* [181], and *Variational Autoencoders (VAE)* [104]. In *Denoising Autoencoders*, the input is disrupted by noise (e.g. by adding Gaussian noise or removing some pixels by using Dropout) and fed into the encoder A. The goal of the decoding function B is to reconstruct the clean version of the input without the added noise. By doing so, the *Denoising Autoencoders* learns to create more robust representation of the input or can be used for error correction. *Contractive Autoencoders*, on the other hand, try to make the feature extraction less sensitive to small perturbations. By adding the squared Jacobian norm to the reconstruction loss, the latent representations of the input tend to be more similar to each other. This diminishes latent representations that are not important for the reconstruction and only important variations between the inputs are kept. The encoder of *Variational Autoencoders* map the input to a probabilistic distribution; Instead of mapping the input to an encoding vector, the input is mapped to a vector representing the means μ and another vector representing the standard deviations σ. This done by adding two fully connected layers after the encoder A, one layer to predict μ and the other layer to predict σ. Afterwards, a variable is sampled from this distribution $z \sim \mathcal{N}(\mu, \sigma^2)$[15] and fed into the decoder B. By doing so, the latent space becomes by design continuous and allows random sampling and interpolation of data.

[30]: Rumelhart et al. (1985)

[177]: Baldi (2012)

[178]: Ranzato et al. (2007)

[179]: Le et al. (2012)

[180]: Vincent et al. (2008)

[181]: Rifai et al. (2011)

[104]: Kingma et al. (2014)

15: this process is also called the reparameterization trick

Other approaches for *self-supervised learning* typically augment the visual scene and either predict the augmentation parameters, reconstruct the original version of the image, or learn consistent representations among augmented views of the image. For example, some models use masking

to learn visual representation; A part of the input image is removed and the model predicts the missing part (image inpainting) [182]. This not only allows to generate part of images but also to learn visual representations of the image [183–185]. Other approaches rotate images and predict the rotation angle to generate representations [186, 187]. There exist many other methods that augment images and learn good visual representations based on it. A comprehensive overview is given by [188].

[183]: Pathak et al. (2016)

[184]: He et al. (2022)

[185]: Shi et al. (2022)

[186]: Komodakis et al. (2018)

[187]: Zhai et al. (2019)

[188]: Chen et al. (2022)

Another popular *self-supervised learning* paradigm is *contrastive learning*. The idea of *contrastive learning* is that similar images should yield similar representations. Typically, *contrastive learning* models are trained without labels. In this case, two augmented views of the same image are created, fed through an encoder and the representation of this two views are pushed together in the embedding space by maximizing their similarity [189–191]. Subsequently, the encoder can be used to generate image representations that are used for other downstream tasks such as image classification. However, *contrastive learning* can also leverage information from annotations. For example, Khosla et al. [192] use labels in a contrastive setting to pull the representations of images of the same class together in the embedding space, while simultaneously pushing apart clusters of samples from different classes.

[189]: Chen et al. (2020)

[190]: He et al. (2020)

[191]: Caron et al. (2020)

[192]: Khosla et al. (2020)

In this thesis, representations of images are learned. Two different principles are applied; vertical self-organisation uses a type of contrastive learning by forcing samples from the same class to have similar representations and samples from different classes to have high diversity. Horizontal self-organisation uses independently trained variational autoencoders to obtain good representations of input patches.

3.5 Meta-Learning

TODO: NOT SURE IF THIS CHAPTER IS STILL NEEDED -> ADD REFERENCE TO THIS CHAPTER

Meta-learning is the process of distilling experience from multiple learning cycles and using the accumulated experience to improve the future learning process [193]. Therefore, meta-learning is also referred to as "learning-to-learn" [194] and improves learning on a lifetime (i.e. single agent) and evolutionary timescale (i.e. population of agents) [195]. Typically, meta-learning includes *base learning* (also referred to as *inner* or *lower* learning) and *meta-learning* (also referred to as *outer* or *upper* learning). During *base learning*, the model learns a typical task such as image classification based on a dataset and a objective function. During *meta-learning*, an algorithm updates the *base learning* algorithm based on a meta objective function[16]. Thus, meta-learning usually iterates between learning a task and improving the learning algorithm that is used to learn the task.

[193]: Hospedales et al. (2021)

[194]: Thrun et al. (1998)

[195]: Schrier (1984)

16: Typical meta objective function aim to improve generalization or learning speed

In the following, I use the categorisation according to Hospedales et al. [193] to describe the different aspects of meta-learning algorithm.

Meta-Representation What meta-knowledge shall be learned by the outer *meta-learning* algorithm.

Meta-Optimizer How the outer *meta-learning* algorithm learns, i.e. the from of the learning algorithm.

Meta-Objective What the goal of the *meta-learning* algorithm is.

The first dimension of the meta-learning landscape is the **meta-representation**. Meta-representations describe which part of the learning strategy should be learned. One possibility is to learn good *initial parameters* that can be used by the model during *base learning*. Good initial parameters are only a few gradient steps away from a set of parameters that can solve a task $\mathcal{T}$ drawn from a set of tasks $p(\mathcal{T})$. A popular algorithm to learn initial parameters is called MAML [196, 197] and works well on smaller networks. However, a major challenge is that the outer *meta-learning* algorithm has to find a solution for as many parameters as the inner *base learning* needs. Therefore, many approaches focus on isolating a subset of parameters to meta-learn [198–200]. *Black-box models*, on the other hand, use *meta-learning* to directly provide the parameters required to classify data (i.e. the *base learning* algorithm maps a sample directly to class without iterative parameter optimization) [201–203]. A special case of the black-box models are the approaches based on *metric learning*[17]. The outer *meta-learning* process optimizes a model to transform inputs into representations that can be used for recognition by similarity comparison (e.g. by using cosine similarity or euclidean distance) [199, 204, 205].

[196]: Finn et al. (2017)

[197]: Finn et al. (2018)

17: an explanation why this approach can be considered black-box learning is provided by Hospedales et al. [193]

Other approaches learn the *optimizer* of the inner *base learning* algorithm [206–208]. Typically, such approaches generate each optimization step of the *base learning* algorithm based on the models parameter and a given base objective function. The trainable component can for example be simple hyper-parameters such as the learning rate [208] or more sophisticated such as pre-conditioned matrices [209]. Even the learning algorithm itself can be learned [114].

[114]: Kirsch et al. (2021)

The outer *meta-learning* algorithm can also be used to learn the *inner objective function* (while the outer objective function is fixed). Such loss-learning approaches output a scalar value that is treated as loss by the inner optimizer based on relevant quantities such as prediction/ground truth pairs or model parameters. Common goals of this approach are to obtain a loss that has less local minima [210, 211], provides better generalization [212–214], leads to more robust models [215], or to learn from unlabelled data [216, 217].

Other approaches use the outer *meta-learning* loop to learn *network architectures* for the inner *base learning* cycle. Some approaches use reinforcement learning in the outer loop to learn CNN architectures [218], while other approaches use evolutionary algorithms to learn the topology of LSTM cells [219] or to model the network by a graph of network-blocks [220].

The outer *meta-learning* loop can also meta-learn hyper-parameters such as regularization-strength [221, 222], task-relatedness for multi-task learning [223], or sparsity-strength [223]. Furthermore, it can meta learn suitable data augmentation methods [224, 225], or improve the selection process for the samples of a mini-batch [226, 227]. Even the dataset itself can be learned with dataset distillation; A rather large datasets can be summarized a smaller dataset with only a few support images [228, 229] that still allow good generalization on real test images. In sim2real

learning [230], also the graphics engine [231, 232] can be trained in an outer loop so that the performance on real-world data is maximized.

The second dimension of the meta-learning landscape is the **meta-optimizer**. The meta-optimizer describes how the algorithm in the outer *meta-learning* loop is learned. Many methods use backpropagation of error and gradient descent (c.f. Section Section 2.2) to optimize the meta parameters [196, 206, 215, 221–223, 229]. However, this algorithm has some well-known downsides (c.f. Section XXXXXXXXXX) and requires differentiability. When the inner *base learning* algorithm includes non-differentiable steps [224] or the outer meta objective function is non-differentiable [233], many methods utilize the RL paradigm to optimize the outer objective [234]. However, using RL to alleviate requirement for differentiability is usually computationally very costly. Finally, evolutionary algorithms can be used for learning the outer *meta-learning* function [235–237]. These algorithms have no differentiability constraints, do not suffer from gradient degradation issues, are highly parallelizable, and can often avoid local minima better than gradient-based methods [237]. The downside of evolutionary algorithms is that often a large population size is required (especially if many parameters have to be learned) and the performance is generally inferior to gradient-based methods for large models. These three learning methods are also used in conventional machine learning. However, meta-learning comparatively uses RL and evolutionary algorithms more frequently as some components are often non-differentiable in the meta-learning setup.

The third dimension of the meta-learning landscape is the **meta-objective**. The meta-objective describes what the goal of the meta-learning algorithm is. Typically, the performance of a meta-learning algorithm is evaluated with a metric on the inner loop or a meta-metric on the outer loop. However, there are several design options within the meta-learning framework. First, the inner base-learning episodes can either take few [196, 206] or many [223, 238] samples and thus the goal can be to either improve few- or -many-shot performance. Second, the validation loss of the inner base-learning algorithm can either be calculated at the end of a learning episode to encourage a better *final* performance of the base task or as sum of the loss calculated after each update step to encourage *faster* learning [239]. Third, the goal of the meta-learning can either be to better solve any task drawn from a set of (often related) tasks (i.e. multi-task setting) [196, 204, 215], or to solve one specific task better than when only the inner base learning algorithm is used (i.e. single task setting) [223]. Finally, the meta-optimization can either be done offline (i.e. the inner and outer loop alternate) [196] or online (i.e. the meta-learning takes place within a base learning episode) [215].

4 Neuroscientific Concepts

A large part of the contribution of this thesis consists of identifying appropriate findings from neuroscience and adapting them to a deep learning setting. The identified concepts and the link between the two disciplines is described in this chapter. It should be noted that this section presents only one possible interpretation for the implementation of neuroscientific concepts in the context of deep learning and that alternative interpretations might also be promising. Concrete implementations of the concepts presented in this chapter can be found in Chapter 5 and Chapter 6.

4.1 The Discrepancy

Deep learning is inspired by the human brain and consequently many essential components of a deep learning model are named after their biological counterparts. But this linkage is deceptive: both systems work fundamentally differently and many concepts cannot be transferred directly. This experience is one of the author's most important but also most painful findings, as it was only made towards the end of the thesis (and after hundreds of failed attempts that are not written down in this document). This confusion of terminology between deep learning concepts and insights from neuroscience hinders and in some cases prevents the development and implementation of novel concepts[1].

1: a lesson that probably has to be learnt by anyone researching along the ridge of these two fields

In the biological context, neurons are often discussed as the main component of the nervous tissue and thus the nervous system. A neuron is a clearly defined cell consisting of components such as the cell body, dendrites, and an axon that communicates with other neurons via synapses. Consequently, in the neuroscience context, it is clearly defined what a neuron is. In fact, many of the theories in neuroscience are based on how neurons behave in various circumstances (see following chapters).

In the field of deep learning, this definition is much more vague. It could even be argued that neural networks contain (almost) no neurons. In fact, deep learning systems model and optimise weight parameters. Admittedly, multiplying a weight matrix $\boldsymbol{w}$ with input data $\boldsymbol{x}$ and adding a bias $\boldsymbol{b}$, i.e. $\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}$, can be interpreted as a kind of modelling of a neuron[2] (c.f. Section Section 2.2). Today, no modern network architecture is based purely on this folly-connected modelling pattern. Modern deep learning architectures for computer vision are usually based on CNNs (c.f. Section 2.2.1) or vision transformer [13]. The convolutional and pooling operations of CNNs cannot properly be modeled with neurons. The same applies to the attention mechanism of vision transformers. Furthermore, both architectures use various other concepts that have shown to improve results and/or computation efficiency such as skip-connection, normalization layers, etc. As a result, findings from neuroscience are often based on the investigation of neurons or neuron structures and consequently,

2: however, also in this case, weights and not neurons are modelled explicitly

[13]: Dosovitskiy et al. (2021)

the theories and findings derived from them cannot be transferred to deep learning settings in a simple way as an artificial neuron is something fundamentally different.

Another fundamental difference is the interaction between the neurons. Biological neurons fire electrical impulses when they are sufficiently excited. The synaptic signals generated can be either excitatory or inhibitory. If the biological network is frozen in a certain state, it could be argued that the neurons are binary (excited or not excited). In reality, however, this process is much more complex: neurons communicate in temporal patterns of spikes. Artificial neurons, on the other hand (or, more precisely, the result of the matrix operation, often referred to as a neuron), can take on any floating point number as a "state" and thus represent an infinite number of states in the frozen condition. On the other hand, artificial neurons have no temporal dynamics. Thus, a large part of the intelligence of biological networks lies in the temporals pattern of the spikes, which are not taken into account by artificial networks (and cannot be modelled efficiently due to the clock-gated processing of modern compute infrastructure). These facts make it extremely difficult to link biological and artificial networks: If a frozen state of both networks is discussed, then a fundamental building block of biological neurons is missing, i.e. the temporal aspect. If, on the other hand, time dynamics are taken into account, then deep learning networks are not suitable and spiking networks have to be used.

Although biological and artificial neuronal networks are fundamentally different in many aspects, there are also many similarities and biological findings can serve as a source of inspiration. However, this section should make clear that these sources of inspiration are to be interpreted rather abstractly and cannot be implemented in the same form as in the human brain. In fact, many failed experiments that are not documented in this thesis have failed because their implementation was too strongly oriented towards the biological concept. Therefore, in the following sections, various concepts are discussed and then an intuition is given by the author on how this can be interpreted in a deep learning setting without time dynamics but with complex neuron states.

4.2 Self-Organisation

[240]: Kelso (1995)

[2]: von der Malsburg et al. (2022)

[241]: Dresp (2020)

It is known that large parts of the human brain are self-organizing [240]. Self-organization is the process by which systems consisting of many units acquire their function through local interaction and without interference from a external supervisory system. Recently, renowned scientists [2] put forward the hypothesis that this process of self-organization is *the* key mechanism of natural intelligent systems such as the human brain. Dresp [241] describes seven clearly identified properties of self-organization in the human brain: (i) modular connectivity, (ii) unsupervised learning, (iii) adaptive ability, (iv) functional resiliency, (v) functional plasticity, (vi) from-local-to-global functional organization, and (vii) dynamic system growth. However, it is not obvious how these insights from neuroscience can be integrated into a deep learning framework.

Deep learning networks are usually optimized with end-to-end backpropagation of error. Thus, the entire network is optimized for a specific target. This is considered a violation of the self-organisation principle, as a global update algorithm (i.e. the optimizer) adjusts all network weights to minimise a global target function.

Claim 1 *End-to-end backpropagation of error violates the principle of self-organization. Self-organisation in neural networks requires dividing the network into smaller units that are optimised independently of each other.*

In fact, the plausibility of backpropagation of error for explaining how the brain works was questioned soon after it was published [138, 139]. Since then, many alternative and biologically more plausible algorithms have been proposed such as the feedback alignment (FA) algorithm [242], generalized recirculation [243], as well as target propagation (TP) [244] (c.f. Section 3.3). However, Bartunov et al. [170] have demonstrated that these algorithms do not scale to large vision datasets such as ImageNet [245] and only work for smaller datasets such as MNIST [246] and CIFAR-10 [247]. The only algorithm that seems to scale well is using a proxy objective functions (c.f. Section 3.3).

[138]: Crick (1989)

[139]: Grossberg (1987)

[242]: Lillicrap et al. (2014)

[243]: O'Reilly (1996)

[244]: Le Cun (1986)

[170]: Bartunov et al. (2018)

The biologically most plausible learning algorithm is Hebbian learning (c.f. Section Section 2.5.1) and its variants such as contrastive Hebbian learning [248]. However, even tough I obtain some promising results in preliminary experiments with Hebbian learning (c.f. Appendix Chapter A), this algorithm doesn't seem to be well suited to learn good image representation if a network is trained from scratch.

[248]: Movellan (1991)

Thus, the use of proxy objective functions[3] seems promising; the updates of weights are done in separate local units, yet the power of current deep learning training algorithms can be exploited and systems can be created that can solve complex problems and scale to large datasets by interacting with each other.

3: proxy objective functions are loss functions that are only applied to local units of a system

Implementation 1.1 *Instead of using end-to-end backpropagation of error to optimise the whole system according to a single global objective function, proxy objective functions are applied to local units. Thus, self-organisation takes place through the optimisation of local units instead of an overall system.*

The next ambiguity is what local units are in a deep learning setting. In deep learning models, typically not neurons are modelled but the trainable parameters[4]. One of the strengths of deep learning systems is that matrix multiplications allow to calculate the layer outputs in one step. Calculating each neuron activity separately, on the other hand, would be very inefficient. Therefore, the smallest meaningful unit for local updates seems to be a layer and not a single neuron. If a neural network is visualised layer-wise from left to right, then the self-organising units line up horizontally. This is why layer-wise self-organisation is also referred to as "horizontal self-organisation" in this thesis (c.f. Figure 4.1).

4: the weights $\boldsymbol{w}$ and the bias $\boldsymbol{b}$ are modelled, so that the output y can be calculated as $\boldsymbol{y} = \boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}$ for given data $\boldsymbol{x}$

Implementation 1.2 *Self-organisation takes place within local units. A local unit can be a part of a model such as layer. In this thesis, this type of self-organisation is called horizontal self-organisation.*

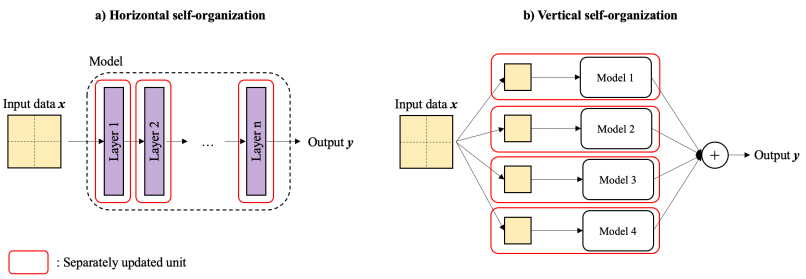

Figure 4.1: Two different ways of building self-organizing units. Self-organization can either take place horizontally (i.e. layer-wise) within a model (a) or vertically by splitting the data into patches and processing them with independent units (b). The independent units are marked with a red frame.

A second type of self-organisation is not to split the model into separate units but to split the data. The input data can be divided into smaller patches and then be processed by independent models. It is important that one model does not process the entire set of existing patches, as is the case with the vision transformer [13]. If only one model is used, there would again be an end-to-end backpropagation of error on a single unit. But if the patches are processed by a graph of independent models, then each model can be considered as a self-organising unit. In this thesis we call this kind of self-organisation the "vertical self-organisation" (c.f. Figure 4.1).

[13]: Dosovitskiy et al. (2021)

Implementation 1.3 *A second type of self-organising unit can be a model that processes a subset of input data that is not shared with other models. In this thesis, this type of self-organisation is called vertical self-organisation.*

4.3 Net-Fragments

Another very important principle according to von der Malsburg et al. [2] is that neurons form net-fragments (a.k.a. sub-networks) that represent features of objects (c.f. section Section 3.1). For example, some net-fragments may represent shapes and structures while a multitude of such net-fragments together represent objects such as persons or entire scenes. Net fragments are a compositional data structure, meaning that some low-level features can be composed to a higher-level feature and multiple higher-level features are composed to an object. To some extend, neural networks do this as well; data is fed into the network, the first layer extract some low-level patterns, and subsequent layer combine these patterns in higher-level features in a hierarchical manner. However, there is one big difference: The features that are used to fulfil a specific task such as classification are extracted from the latent space of *one* single layer. Net-fragments in the human brain, on the other hand, are not considered to be present at a specific point in time but to be built up over a short period of time.

Claim 2 *Net fragments are represented by groups of neurons and their activity over multiple time-dependent spikes. In addition, several net fragments can be composed into a higher-level fragments. Thus, objects are not represented by a single neuron but by all neurons that were activated due to this object.*

Deep learning models are not based on time-dependent spikes of neurons that compose features to more complex features. However, a time-step could be interpreted as a forward-step from one layer to the next within a network of layers. Thus, net-fragments would be the activation of

multiple layers. This interpretation is also in line with the compositional property of net-fragments. Low-level features that are detect within the first layers activate higher-level features in subsequent layers. An object, however, is not represented in the last layer of a neural network but through all activations within the network. Thus, representations of objects cannot be extracted from a single layer but from multiple layers. Intuitively, this seems promising. For example, auto-regressive models applied on speech capture different information at different layers of the network [249]. While the first layers contain more information to distinguish speakers, representations in later layers provide more phonetic content. Thus, extracting information from several layers could lead to representations containing more information.

[249]: Chung et al. (2019)

Implementation 2.1 *Net-fragments cannot be extracted from one single layer. Therefore, the representations of an image are extracted from multiple layers.*

In the human brain, distinct groups of neurons represent specific net fragments. This means that neurons are only active when the corresponding feature is present in the input and are inactive otherwise. Moreover, the features represented by the neurons should be meaningful[5] and consequently not active for every existing object. This inevitably leads to the sparse and diverse activations. The activations are sparse and diverse because, the object in the image consists of only a small sub-set of all learned features. Thus, only a small sub-set of the neurons should be active for a given object. When the input changes and a different object is shown to the model, also the set of active neurons should change.

5: interpretable in the sense that a neuron is active if a specific feature is present and inactive otherwise

Claim 3 *The activation patterns of neurons that represent net-fragments must be sparse and diverse to obtain meaningful activation patterns.*

The extent to which neuronal networks contain or can produce net fragments with meaningful activation patterns is investigated in detail in a preliminary study. The methodology used as well as an in-depth evaluation is described in the appendix in the Chapter Chapter B. In summary, it was found that neural networks do not contain net fragments with meaningful activation patterns by default. Typically, there are neurons that are always active regardless of the input data and encode most of the input information in their activation strength. Other neurons, however, are never active and are therefore not needed by the network. If, on the other hand, a sparsity and diversity constraint is applied, then layers are obtained with neurons that appear to be suitable for net fragments.

Implementation 3.1 *Sparse and diverse activation patterns can be obtained by imposing sparsity and diversity constraints to the objective function of the model.*

However, sparsity and diversity constraints are mainly necessary to obtain meaningful and more robust activation patterns. They don't seem to be necessary, for example, if the goal is to obtain good classification performance. An alternative that lead also to robust and meaningful activations is to model the net-fragments as probability distribution. For example, variational autoencoders (c.f. Section 3.4) model their latent space as a multivariate Gaussian distribution.

4.3.1 Sparsity

Sparistiy seems to be an important principle in biological networks. Presumably, this is because the neurons in the brain can be inhibitory or excitatory by firing spikes at different time intervals. An artificial neuron, on the other hand, uses a floating point number as its state and can thus represent an infinite number of different states, in contrast to the biological neuron. Thus, an artificial neuron is able to represent many different things while the biological neuron is responsible for specific features. Therefore, the activations of biological neurons *must be* sparse, while the activations of artificial neurons *do not have to be* sparse (but being sparse has also advantages for artificial neurons, see below).

Sparsity is deeply embedded in the biological learning process. For example, in the visual cortex of mice are more than 75% of the neurons active before the first opening of the eyes, 36% after the opening of the eyes and only 12% in adulthood [250]. Thus, a sparsification of neuronal activations takes place through visual experience. In the field of deep learning, sparsity is often interpreted in two different forms; sparse weight matrices and sparse activation matrices. Sparse weight matrices are often chosen to make models smaller or to increase inference speed [251, 252]. From a biological point of view, this process of first creating a large network and then shrinking it is obviously not plausible[6]. Sparse activations, on the other hand, can increase robustness [253]. Intuitively, sparse activations enforce that only the most relevant information is passed to the subsequent layer. Furthermore, combining sparsity with diversity can help to obtain more interpretable activations (c.f. Appendix Chapter B).

[250]: Rochefort et al. (2009)

[251]: Louizos et al. (2018)

[252]: Hoefler et al. (2021)

6: otherwise we would have a large brain at the beginning, which becomes smaller by factors in the course of time

4.4 Lateral Connections

In addition to forward connections, lateral connections are also located in visual cortex [254]. Thus, the biological neurons are not only connected to the neurons in the subsequent layer but also within the same layer. Von der Malsburg et al. [2] describe that lateral connections help active neurons to support each other in order to remain active: Initially, many neurons are active, but relatively quickly a part of the neuron becomes inactive and only those neurons that support themselves remain active. In this way, lateral connections can lead to the emergence of high-level features from initial activations.

[254]: Gilbert et al. (1990)

Claim 4 *Lateral connections allow active neurons to support each other and to remain active.*

Neural networks lack this time dynamic and neurons do not suddenly become inactive during a forward pass. In addition, initial experiments have shown that performance does not improve when the activation maps of neural networks are sparsified over several steps. One assumption is that all information is already contained in the data at the beginning and the same sparsified activations can also be generated in one initial step.

However, the lateral support between neurons seems promising if the input changes slightly: An activation map can be calculated for a given

input image. In a next step, the input can slightly be changed through data augmentation to btain another activation map from the same image. Thus, the network receives different views of the same image and generates different activation maps of the same image. In every step, the model can decide whether the activation map from the previous step was correct or if the activation map should be updated.

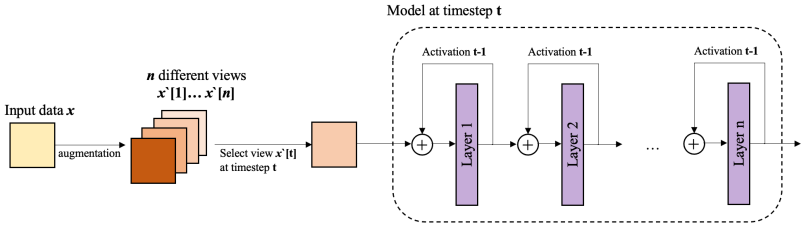

Figure 4.2: An illustrative network architecture of a model with lateral connections: An in sample x is augmented n times to obtain n different views $x'[1], ..., x'[n]$ of the same sample. At every timestep t, the sample $x'[t]$ is fed through the model. Each layer receives the activation map of the previous layer as input as well as its own activation maps from the previous time-step $t-1$.

This could be implemented by interpreting the lateral connection as a recurrent connection. An illustrative architecture of such a model is shown in Figure Figure 4.2. A data sample x can be augmented with data augmentation methods n times to obtain n different views $x'[1], ..., x'[n]$ of the input sample. Afterwards, the n views are iteratively fed into the model. At every time-step t, the model calculates an activation map $a_l[t]$ for each layer l. If $t > 1$, the input of the layer l is not only the activation map of the layer $l-1$ but also the activation map of the previous time-step $a_l[t-1]$. The activation map of the previous time-step is stored $a_l[t-1]$ by the recurrent (lateral) connection. Thus, the activation map $a_l[t]$ is not only calculated based on the activation map of the previous layer $a_{l-1}[t]$ but also based on the activation map of the previous time-step $a_l[t-1]$. This allows each layer to preserve the activation map $a_l[t-1]$ from the previous time-step or to correct it based on a slightly different input $a_{l-1}[t]$ from the previous layer. The lateral connections thus support the activations within a layer; a layer can look at several inputs and decide which features are present over several time-steps (support these features over several time-steps).

Implementation 4.1 *A lateral connection can be implemented as a recurrent connection that "stores" the layer's activation of a previous time-step. If the same input is present for several time-steps in a slightly augmented version, the layers can keep their previous activation maps (lateral support) or correct them.*

With vertical self-organisation, the lateral connections can be implemented not only as recurrent connections within layers, but also as connections between the sub-models. In this case, each sub-model extracts a distinct low-level feature. Based on this feature, a guess can be made to which higher-level feature or object the extracted feature belongs to. Through communication with neighbouring sub-models, this guess can either be supported or rejected. Thus, the lateral connections are used as a "support channel" between sub-models.

Implementation 4.2 *In the case of vertical self-organisation, a lateral connection can be implemented as connection between sub-models to vote for which high-level feature or object is represented in the image. Either the prediction of a model can be supported by neighbouring models or rejected.*

4.5 Other Principles

Other important principles that are considered promising are a *continuous input* signal and an *embodiment* of the agent. However, implementing such an interaction between an agent and an outside world is out of scope for this thesis but might be interesting for future work.

The visual cortex receives a continuous input signal. This allows the tracking of moving objects and enables an object to be perceived from different angles. Since the change of the object between the captured frames is small, it can be determined that it is always the same object instance and consequently mapped to the same mental object prototype of a world model.

An ANN, on the other hand, is typically trained on samples that have little relation to each other. When the system is trained on images, each frame is different; with videos, each sequence of frames is different. A continuous input might help to get better representation of objects through self-supervised learning. If an input is continuous and shows the same object from different angles or in different transformations (e.g. stretching) and it can be inferred that it is the same object then the object representations derived from this continuous stream can be homogenized. These principles are already applied to some extent by self-supervised learning systems for computer vision. In contrastive learning, a popular form of self-supervised learning, two different views are derived from one image by data augmentation, and their representations are then pushed closed together in the feature space [189, 191, 255]. However, this paradigm is still quite limited since only two views of the same scene and not the continuous transformation of an object are presented to the learning system.

[189]: Chen et al. (2020)

[255]: Chen et al. (2020)

[191]: Caron et al. (2020)

Claim 5 *A continuous input stream can help to build a better representation of objects, especially if the objects are slightly transformed between captured frames or if the point of view changes continuously and smoothly.*

Furthermore, efference copies of motor signals in the form of neuronal activities are directly sent to the brain's sensory system if animals are moving [256]. Such efference copies can be useful to better understand objects and how they behave when undergoing object transformations. To do so, the agent must be allowed to perform actions and interact with its world. This gives the agent more information about the objects but also about the physical properties of a (virtual or real) world. For example, he can perform different actions on different objects: He can rotate, squeeze, stretch or move objects. By doing so, he receives different visual and sensory feedback for identical actions on different objects. The agent can map the captured feedback signals to the object representations of an internal world model.

[256]: Keller et al. (2012)

Claim 6 *Allowing an agent to interact with the world can help to learn better object representations and to build a better world model.*

Furthermore, the agent can learn to understand objects better by improving his internal object representations on its own. For example, if the agent examines an object and his current internal object representation does not describe how the object looks from the side, then the agent can

rotate the object accordingly and complete or correct the representation. Such a behaviour could be implemented within the agent itself, for example, by optimising an entropy-based loss in order that the agent explores the world and tries to learn unknown things [257, 258].

[257]: Storck et al. (1995)

[258]: Klapper-Rybicka et al. (2001)

5 Vertical Self-Organisation

In this thesis, two different concepts of self-organisation are implemented. This chapter presents a method based on vertical self-organisation (c.f. Section 4.2). Vertical self-organisation is based on training small units within a network independent from each other. The first section presents the methodology (i.e. how vertical self-organization was implemented) and the second section describes the obtained results.

5.1 Methods

The first choice for networks based on vertical self-organisation is what the independent units are, i.e. which network parameters are trained independently. In this thesis, a linear layer was selected as a self-contained unit. This is a small unit (e.g. compared to a model part that contains many layers) but can still be trained efficiently. Smaller units would be neurons or parts of a layer, but training them separately would be very inefficient, since the efficiency of matrix operations on GPUs could no longer be fully exploited[1].

1: the layer output can be efficiently calculated by a single matrix multiplications and addition, i.e. $z = w \cdot x + b$ (c.f. Section 2.2)

Each layer optimises its own proxy objective function. Proxy objective functions are used because this type of local optimisation allows very good performance even on larger data sets, in contrast to other target functions such as target propagation, synthetic gradients or feedback alignment (c.f. Section 3.3). Figure Figure 5.1 visualizes the updates based on a proxy objective function. A objective function is calculated for each layer and the optimization algorithm only updates the parameters within this layer. Thus, the gradients do not flow backwards to preceding layers.

A combination of diversity and sparsity constraints is used as the loss function, which leads to representations that are easy to interpret and suitable for net fragments (c.f. Section 4.3) and also increases robustness (c.f. Section 4.3.1). Identical to the preliminary experiments in Appendix Chapter B, the sparsity is achieved by using the kullback-leibler (KL) divergence [179]. The model consists of several linear layers l with a relu activation function $(z)^+ = max(0, z)$. A layer consists of $m^{(l)}$ neurons and the activations $\boldsymbol{z}^{(l)}$ of a linear layer l are calculated for a given input $\boldsymbol{z}^{(l-1)}$ as

[179]: Le et al. (2012)

$$\boldsymbol{z}^{(l)} = z_1^{(l)}, ..., z_m^{(l)} = \boldsymbol{w}^{(l)} \cdot \boldsymbol{z}^{(l-1)} + \boldsymbol{b}^{(l)} \tag{5.1}$$

whereby $\boldsymbol{w}^{(l)}$ is the weight and $\boldsymbol{b}^{(l)}$ the bias of layer l.

The activation probability can be calculated for each neuron. If a mini-batch contains n samples, the activation probability $\hat{\rho}^{(i)}$ of a neuron $z^{(i)}$ can be calculated as:

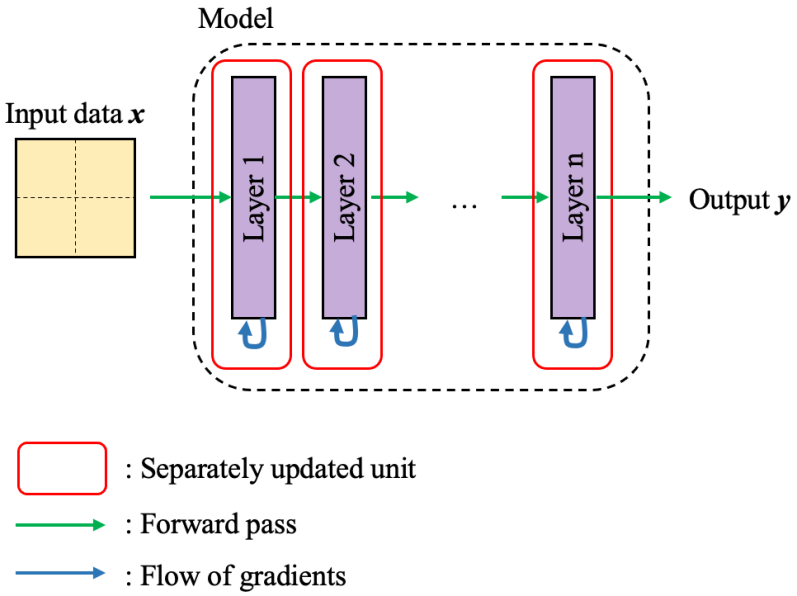

Figure 5.1: The flow of gradients within the network based on vertical self-organisation: The data is fed from layer to layer during the forward pass. The layers, are trained independently and the gradients do not flow from one layer to the previous one.

$$\hat{\rho}_i^{(l)} = \frac{1}{n} \sum_i^n \frac{1}{1 + e^{-z_i^{(l)}}} \tag{5.2}$$

where $\frac{1}{1+e^{-z^{(i)}}}$ is the sigmoid function that squeezes the activation in the range between 0 and 1. With the KL divergence, the divergence of the current activation probability $\hat{\rho}^{(i)}$ and a desired activation probability $\rho = 0.05$ can be calculated:

$$KL(\rho||\hat{\rho}_i^{(l)}) = \rho \cdot \log \frac{\rho}{\hat{\rho}_i^{(l)}} + (1 - \rho) \cdot \log \frac{1 - \rho}{1 - \hat{\rho}_i^{(l)}} \tag{5.3}$$

The sparsity loss L_s is the sum of the divergence between all $\hat{\rho}^{(i)}$ and ρ:

$$L_s(\rho, \hat{\rho}) = \sum_{i=1}^{m} KL(\rho||\hat{\rho}_i^{(l)}) \tag{5.4}$$

The second constraint is a diversity constraint. The goal is that the activations of *different* objects are diverse. For this purpose, the activations $z^{(l)}$ are made as identical as possible (i.e. pushed together in feature space) if they stem from the same class and as different as possible if they stem from different classes. The cosine similarity is used to calculate the similarity between two activations $z_i^{(l)}$ and $z_j^{(l)}$:

$$\cos(z_i^{(l)}, z_j^{(l)}) = \frac{z_i^{(l)} \cdot z_j^{(l)}}{\max(||z_i^{(l)}||_2, ||z_j^{(l)}||_2)} \tag{5.5}$$

In order to make representations of different objects different, resp. to make representations of identical objects identical, the information of image labels y_i is needed. Thus, $y_i \in C$ where C is the set of classes. The diversity loss L_d minimises the similarity $\cos(z_i^{(l)}, z_j^{(l)})$ if the two

activations stem from different classes $y_i \neq y_j$, or maximises the similarity if they stem from the same class $y_i = y_j$.

$$L_d(z^{(l)}) = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} k \cdot \cos(z_i^{(l)}, z_j^{(l)}) \tag{5.6}$$

whereby k changes sign depending on the class:

$$k = \begin{cases} +1, & \text{if } y_i \neq y_j \\ -1, & \text{otherwise} \end{cases} \tag{5.7}$$

However, it was found that the loss is more stable if the similarity is not calculated between two activations but between one activation and the average activation of a class. The average activation of a class c can be calculated over n_c samples from this class as:

$$z^{(l)}[c] = \frac{1}{n} \sum_{i=1}^{n_c} z_i^{(l)}, \text{ for } y_i = c \tag{5.8}$$

Another problem of is loss is if all activations are identical, then $\cos(z_i^{(l)}, z_j^{(l)}) = 1$ and the loss gets $L_d = 0$, since the similarities of the same and different classes neutralise each other. Therefore, a margin between the similarities is enforced, as done for the triplet-margin-loss [259].

[259]: Balntas et al. (2016)

$$L_d(z^{(l)}) = \frac{1}{n} \sum_{i=1}^{n} \max(\cos(z_i^{(l)}, z^{(l)}[v]) - \cos(z_i^{(l)}, z^{(l)}[y_i]) + \text{margin}, 1) \tag{5.9}$$

where v is a random class drawn from the set $v \{C$
$y_i\}$ and margin is a hpyer-parameter that was set to 1. Thus, the triplet-margin-loss is calculated but the $L2$-norm is replaced by the cosine-similarity as distance measure and the positive, resp. negative anchor is replaced with the average class activation from the same resp. from a randomly selected different class.

The loss used is the sum of sparsity loss and diversity loss, with the diversity loss weighted by $\lambda = 0.1$:

$$L = L_s(\rho, \hat{\rho}) + \lambda \cdot L_d(z^{(l)}) \tag{5.10}$$

One problem with proxy objective functions is that this type of training often requires layer-wise training. First, layer 1 is trained completely, then the weights are frozen, then layer 2 is trained and so on. This is inefficient because it requires more forward-passes than if the model is trained with end-to-end backpropagation. It was found that this loss functions allows to train all layers simultaneously. With a forward-pass, all activations are calculated, followed by a layer-wise backward-pass where the gradients from one layer do not propagate back into the previous layer. Since the gradients are only needed locally, no large graph of gradients has to be calculated, which reduces the memory utilization on GPUs and allows

larger mini-batch sizes. In addition, this type of architecture is very easy to parallelise, since theoretically each layer (i.e. each independently trained unit) or a group of layers can be assigned to a different GPU. In contrast to end-to-end backpropagation of error architectures, the gradients only flow backwards locally on a GPU and do not have to be passed on to other GPUs.

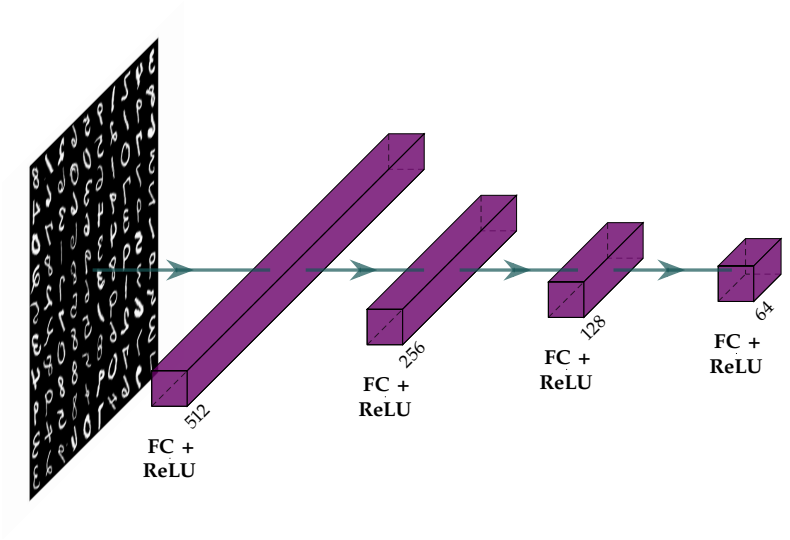

Figure 5.2: The network architecture of the fully connected model for vertical self-organisation with fully connected layers.

The model used consists of 4 fully connected layers with relu activation. The first layer has 512 neurons, the second 256 neurons, the third 128 neurons and the fourth 64 neurons. The model is illustrated in Figure Figure 5.2. Each layer is trained separately by minimising the loss of equation equation (**??**) with the Adam optimizer[260] and a learning rate of $\eta = 1 \cdot 10^{-3}$. The mini-batch size is 60, 000.

[260]: Kingma et al. (2017)

5.1.1 Extraction of Representations

In accordance with net-fragments as in Section 4.3, the representations are not extracted at a specific point (i.e. a pre-defined layer) but all representations from all layers are taken into account to fulfil a task. In the following, this is demonstrated based on a classification task, but other tasks are also conceivable in the future. After training, the average activation $z^{(l)}[c]$ for each class $c \in C$ in each layer l is determined, as done in Equation equation (**??**). These averages from the training set represent prototypes of each class object in each layer and can be considered as reference representation per class. Thus, the representations needed for this task are computed *after* training and are not part of the training as for example in the case of a classification loss based on cross-entropy[2]. When a new sample $\boldsymbol{x}_s$ is classified, the cosine similarity between the activations $\boldsymbol{z}_s^{(l)}$ of this sample and the class prototypes $z^{(l)}[c]$ is calculated in each layer.

2: but in the case of classification this information is implicit in the loss function

$$\cos_s^{(l)}[c] = \cos(\boldsymbol{z}_s^{(l)}, z^{(l)}[c]) = \frac{\boldsymbol{z}_s^{(l)} \cdot z^{(l)}[c]}{\max(||\boldsymbol{z}_s^{(l)}||_2, ||z^{(l)}[c]||_2)}, \text{ for } c \in C \quad (5.11)$$

Thus, the cosine similarity $\cos^{(l)}[c]$ for each class $c \in C$ and each of the for layers $l \in 1, ..., 4$ is calculated. Afterwards, the average class c with the highest average cosine similarity between the sample activations $z_s^{(l)}$ and the class prototypes $z^{(l)}[c]$ is used as prediction.

$$\arg\max_{c \in C} \frac{1}{4} \sum_{l=1}^{4} \cos_s^{(l)}[c] \tag{5.12}$$

This results in a weighted voting; if a layer is very sure that the sample belongs to a specific class, then the sample has a high cosine similarity with one class prototype and a low similarity with all other class prototypes. Accordingly, this layer influences the prediction more than a layer that cannot clearly assign the sample to one class and calculates a similarly high cosine similarity between the sample and all prototypes.

5.1.2 Lateral Connections

As described in section Section 4.4, lateral connections serve neurons to support their activations among each other. In this chapter it is also described in detail that this can be implemented through recurrent connections. There are several ways to implement recurrent connections. A very simple possibility is to concatenate the layer input $z_t^{(l-1)}$ at time t with the layer output $z_{t-1}^{(l)}$ at time $t-1$. This is visualized in Figure Figure 5.3. Of course, $z_{t-1}^{(l)}$ is undefined at $t = 0$. In this case, $z_{t-1}^{(l)}$ is initialized with zeros.

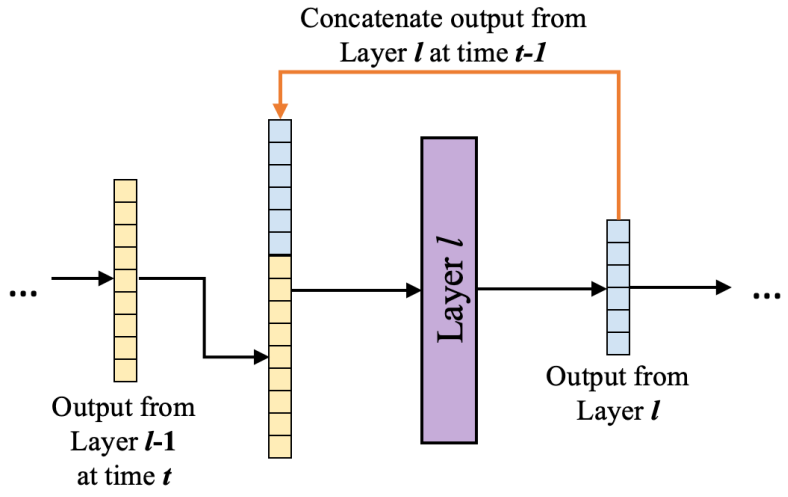

Figure 5.3: The lateral connections can be implemented by concatenating the layer's output at the previous time-step with the layer's input at the current time-step.

A second option is to use a second weight matrix as it is usually done in recurrent layers and to expand the equation equation (**??**) for the activation function as follows:

$$z_t^{(l)} = w_x^{(l)} \cdot z_t^{(l-1)} + b^{(l)} + \overbrace{w_h^{(l)} \cdot z_{t-1}^{(l)}}^{\text{lateral connection}} \tag{5.13}$$

whereby $w_x^{(l)}$ is the weight multiplied with the layer input and $w_h^{(l)}$ the weight multiplied with the previous layer output. In both cases, the

layer receives information about the activations at the previous time-step. However, this only seems helpful if the model input is not static. Therefore, the model input is available over several time-steps and is augmented after each time-step. Thus, the model receives different views of the same image and can adjust its activations from the previous time-step if necessary. The following image augmentation techniques are applied, each with a probability of $p = 0.8$:

- **Color Jitter**: Randomly change brightness, contrast, and saturation of the image.
- **Gaussian Blur**: Blur the image with randomly chosen Gaussian blur.
- **Random Rotation**: Randomly rotate the image with an angle in the range $[-15, \ldots, 15]$
- **Adjust Sharpness**: Randomly adjust the sharpness of the image.

Figure Figure 5.4 visualises how this augmentation affects samples of the MNIST data set [18]. The original image is shown on the left and 9 augmented versions of it are shown on the right. This allows the model to perceive the same image from different views.

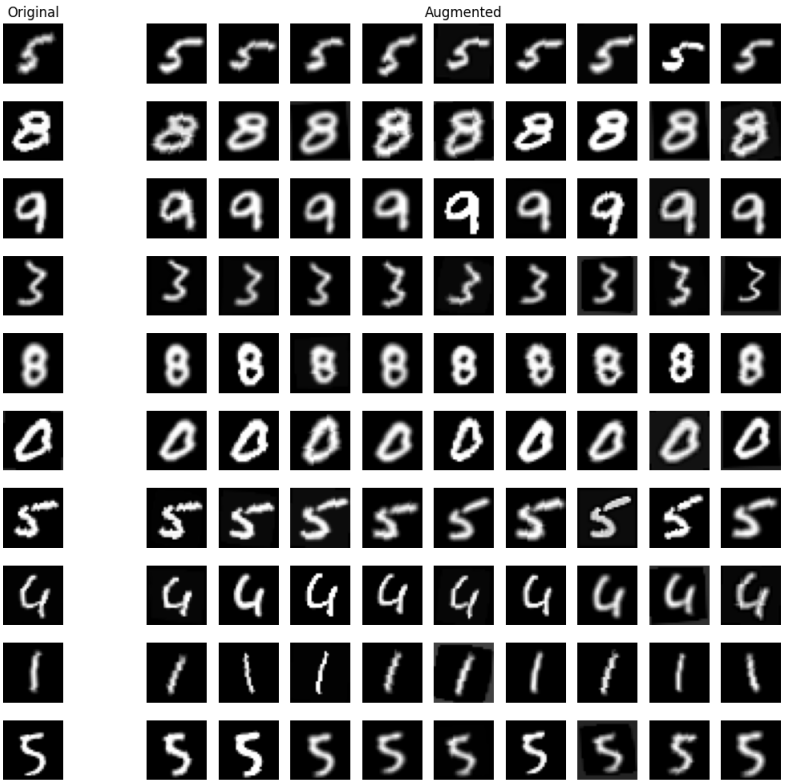

Figure 5.4: Data augmentation applied on 10 samples of the MNIST data set. The original samples as it is in the data set is shown on the left, 9 augmented versions of the same samples are shown on the right.

Recurrent connections, which in this context represent lateral connections, are typically used to process sequential data or text. In this case, the data is collected cumulatively before an output is generated. For example, in text processing, all word tokens of a sentence are typically read before the model classifies the sentence as heat-speech or not. This is necessary because all sentence information is required and classification cannot be done on the basis of a single token. Such models are typically trained

with backpropagation through time (BTT). Thereby, the gradients flow backwards over several time steps. This leads to well-known problems such as vanishing and exploding gradients.

In this thesis, BTT is not used, which means that a prediction is made after each time-step, but the prediction potentially improves with more time-steps. This leads to desirable properties: (i) Problems with vanishing or exploiding gradients do not exist, regardless of how many time-steps the model requires. (ii) After each time-step, representations can be extracted according to Section Section 5.1.1. If a task can be solved correctly with a high probability on the basis of these representations (e.g. the object representation can clearly be assigned to one class label), the sequential analysis of the image can be aborted. If this is not the case, further time-steps can be carried out until the model has a sufficiently high confidence in its prediction. Thus, the number of time-steps can be sample-dependent.

5.1.3 Hierarchical Features

A criticism of the proposed model is that it does not learn hierarchical features, even tough hierarchical features are one of the main reasons for the good performance of deep learning systems. The diversity loss (c.f. Equation equation (**??**)) forces enforces in each layer that the latent representations of objects of the same class are similar and that the representations of different classes are different. This violates the concept of hierarchical feautres; the first layers should learn general features that are helpful for all classes, but cannot necessarily be assigned to a specific class. Only later layers build class representations that are specific to a class. In the current setting, however, already the first layers generate class specific representations.

In the following, three possible measures are described to counteract this problem: (i) The fully connected layers are replaced by convolutional layers, so that the first layers have a smaller field of view and can only recognise local features. (ii) An adapted version of diversity loss forces a separation of features by class only in the last layers. (iii) While the very specific image information flows into the network from one side, general information in the form of class labels is fed into the network from the other side and these two types of information are fused. These three measures can be applied either individually or in combination with each other.

Convolutional Architecture

corresponding layer has a sufficiently large field-of-view[3]. In a CNN, the field-of-view in the first layers is restricted by design, but not in fully connected layers. Consequently, it might help to use a CNN architecture instead of a model based on fully connected layers only. This means that the network is no longer able to separate the representations based on the class in the first layers as the field-of-view is too small, but can only do this in the later layers, which have a larger field-of-view.

3: the field-of-view are all pixels of the input image that can influence a neuron's activity

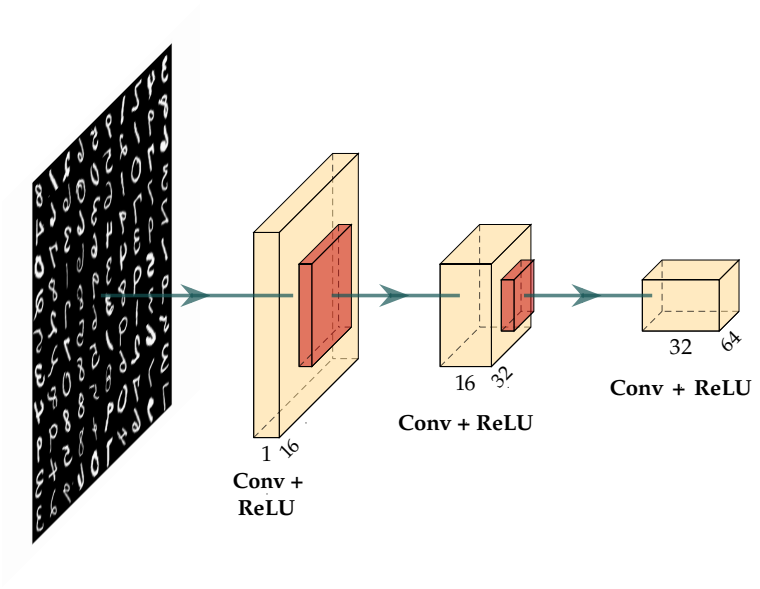

Figure 5.5: The network architecture of the CNN for vertical self-organisation with fully connected layers.

The CNN architecture used in this thesis is shown in Figure Figure 5.5. Three convolutional layers with ReLU activation function and with 16, 32, and 64 channels are used, and max-pooling layers are applied between each convolutional layer. For training, the same hyperparameters and loss functions are used as for the model with fully connected layers.

Hierarchical Diversity Loss

A second measure can be to adapt the diversity constraint of the loss function. In the current version, it forces the latent representations of objects of the same class to be similar and those of different classes to be different. This is useful in the last layers, where high-level features (i.e. high-level net-fragments) or object representations should be detected and be separated from each other. In the first layers, on the other hand, the separation should not depend on the class label. Nevertheless, the activations should also be diverse if low-level features are detected (c.f. Section Section 4.3).

Therefore, the sparsity constraint is split into two parts: One part ensures that the activations within a (large) mini-batch are diverse and thus enforces that different features are captured and represented by different neurons. The second part ensures, as before, that the activations are diverse for different classes. Thus, one part of the constraint ensures *diversity within the mini-batch* and the other part ensures *diversity between different classes*.

These two parts are weighted linearly from the first to the last layer. Since the first layer should have a high diversity within the mini-batch, it has a high weight on the first part of the diversity constraint and a low weight on the second part. The last layer has an inverse weighting and pays more attention to the second part of the diversity constraint than to the first part.

The first part of the diversity constraints ensures diversity within a mini-batch. This is achieved by ensuring that each neuron within a mini-batch

should be active.

TODO: At the moment, various experiments are still ongoing to find out how this can be implemented (therefore, not yet explained in more detail).

Two-Way Information Flow

Hinton [157] introduced with the forward-forward (FF) algorithm (c.f. Section Section 3.3) a promising idea for models based on proxy objective functions; The image is fed into the input layer of the network and the corresponding label is fed into the output layer of the network. The image remains static for several time-steps and the network is trained to push the layer's activations above a certain threshold. In a second stage, the image is fed into the network together with the wrong label and the network is trained to push the activations below a certain threshold. He found that when low-level features (i.e. images) are fed into the network from one end and high-level features (i.e. class labels) are fed into the network at the other end, a feature hierarchy is created. However, this approach has one major disadvantage: During inference, each possible sample-label combination has to be fed into the model and the label that caused the highest neural activity is used as the model's prediction.

[157]: Hinton (2022)

In this thesis, this is implemented in a different way, which does not have this disadvantage. Identical to the FF algorithm, the image is fed into the input layer of the network for multiple time-steps. The label, on the other hand, is not fed directly into the output layer but is made available to the last layer within the loss function. Each layer maximizes the mutual information (MI) between its own activations and the activations of the previous and subsequent layers. Thus, the first layer maximizes the MI between the input image and its activations, the last layer maximises the MI between its activations and a label vector. This creates a feature pyramid in which a smooth transition from concrete sample to abstract class label is learned. During inference, the last label then directly predicts latent representations that can be assigned to a class.

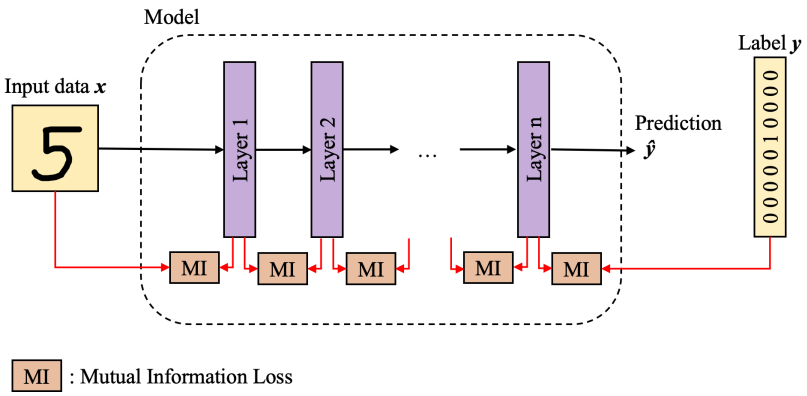

Figure 5.6: Vertical self-organization with mutual information loss: Each layer maximizes the mutual information (MI) between its activations and the activations of the previous and subsequent layer. The first layer maximizes the MI between its activations and the input image (instead of the previous layer) and the last layer maximizes the MI between its activations and the image label (instead of the subsequent layer).

This process is visualized in Figure Figure 5.6.

TODO: At the moment, various experiments are still ongoing to find out how this can be implemented (therefore, not yet explained in more detail).

5.2 Results

The results obtained by this model are presented below. It is important to note that the goal of these models is not to achieve the best possible results on a benchmark. In large-scale comparisons, it has already been found many times that end-to-end backpropagation of error is far superior to other methods, especially in the area of image classification [170]. Rather, the aim here is to examine interesting concepts that are more biologically plausible and can serve as inspiration for future work.

[170]: Bartunov et al. (2018)

5.2.1 Data set

The model was trained and evaluated on the MNIST [18] data set. It is a data set of handwritten digits with 60'000 training samples and 10'000 test samples. In this thesis the predefined training and test split is used. The samples are images of a digit with the resolution 28×28 pixel in grayscale, i.e. has one colour channel.

In addition, the models are evaluated on MNIST-C [261], a corrupted MNIST benchmark for testing out-of-distribution robustness of image classification models. This data set consists of the MNIST data with 15 types of corruptions applied to the samples.

[261]: Mu et al. (2019)

5.2.2 Baseline Model

The base model consists of 4 fully connected layers with 512, 256, 128 and 64 neurons as shown in Figure 5.2. No lateral connections or communication across multiple time-steps are used for the baseline. In addition, the measures described in Section 5.1.3 are not used. This model achieves an accuracy on MNIST of 93.7%. This accuracy is remarkably good considering that the model has never been explicitly trained for classification with, for example, a cross-entropy loss. Furthermore, the model is trained extremely fast because these local updates are suitable for batch-gradient descent and not only mini-batch gradient descent. Each layer also has a different accuracy, typically in the range of $90.1\% - 92.9\%$. However, due to the implicit voting using the sum over the cosine similarities, the prediction of the entire model is better than that of a single layer.

5.2.3 Lateral Connections

TODO: How do lateral connections improve the result?

5.2.4 Hierarchical Features

TODO: How do hierarchical features improve the result?

6 Horizontal Self-Organisation

This chapter presents a method based on horizontal self-organisation (c.f. Section 4.2). The idea of horizontal self-organisation is that the input is analysed by several smaller models instead of one big model. This means that each network sees only a patch of the input data and cannot decide on its own what is represented in the input image, but must agree on a representation with neighbouring models. It is important that each model is independent of the other models and that the parameters are not shared between the models. Otherwise the architecture would be comparable to vision transformer [13] and the input patches would no longer be analysed independently.

[13]: Dosovitskiy et al. (2021)

In the following, first the methodology (i.e. how horizontal self-organization is implemented) is presented and afterwards the obtained results are discussed in detail.

6.1 Methods

A crucial design-decision for horizontal self-organisation is the architecture of the models. In this thesis are variational auto-encoders [104] (c.f. Section 3.4) used to process patches of the input image as they have a continuous latent space and are thus more robust and better interpretable (c.f. Section 4.3).

[104]: Kingma et al. (2014)

Typical auto-encoders fed an input image $\boldsymbol{x}$ through an encoder to map the input data to a latent space and through a decoder to recreate the image $\hat{\boldsymbol{x}}$. Thereby, the latent space is limited in size, forcing the model to compress the image with the encoder and to de-compress the image using the decoder.

A variational auto-encoder models the latent space as a multivariate Gaussian distribution. The input image $\boldsymbol{x}$ is also fed through a encoder and a latent representation $\boldsymbol{z}$ is obtained. Afterwards, $\boldsymbol{z}$ is fed through two independent parallel fully connected layers to obtain the μ and σ vectors of the multivariate Gaussian distribution. Afterwards, the re-parametrization trick is used to sample a variable $\boldsymbol{z}'$ from this distribution:

$$\boldsymbol{z}' \sim \mathcal{N}(\mu,\ \sigma^2 \cdot \epsilon) \tag{6.1}$$

ϵ is a random variable sampled from $\mathcal{N}(0, 1)$ and determines the magnitude of the variance. The sampled latent representation is then fed through the decoder and the image $\hat{\boldsymbol{x}}$ is recreated. The model is trained to minimize a distance measure between the input image $\boldsymbol{x}$ and the reconstructed image $\hat{\boldsymbol{x}}$. In this thesis, the mean square error is used. For n images $\boldsymbol{X} = \boldsymbol{x}_1, ..., \boldsymbol{x}_n$, the loss is defined as

$$L_{\text{reconstruction}} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i)^2 \tag{6.2}$$

However, the latent space does not form a multivariate Gaussian distribution on the basis of this reconstruction loss and the network architecture. To accomplish this, a second loss constraint is necessary: Let $P(\boldsymbol{X})$ be the probability distribution of the data $\boldsymbol{X}$, $P(\boldsymbol{z})$ the probability distribution of the latent variable $\boldsymbol{z}$ and $P(\boldsymbol{X}|\boldsymbol{z})$ the probability distribution of generating $\boldsymbol{X}$ for a given $\boldsymbol{z}$. The objective of the VAE is to infer $P(\boldsymbol{z})$ from $P(\boldsymbol{z}|\boldsymbol{X})$ which is the probability distribution that maps $\boldsymbol{X}$ into latent space. Simply put: we want to know what the latent variable $\boldsymbol{z}$ of the input data $\boldsymbol{X}$ is. However, $P(\boldsymbol{z}|\boldsymbol{X})$ is unknown and we have to estimate it from a simpler distribution $Q(\boldsymbol{z}|\boldsymbol{X})$. This simpler distribution $Q(\boldsymbol{z}|\boldsymbol{X})$ is learned by the encoder and should be as close as possible to the real distribution $P(\boldsymbol{z}|\boldsymbol{X})$. This is accomplished by minimizing the KL divergence[1] between these two probability distributions.

1: the Kullback-Leibler (KL) divergence can "measure" the difference between two probability distributions

$$KL\left[Q(\boldsymbol{z}|\boldsymbol{X})||P(\boldsymbol{z}|\boldsymbol{X})\right] = E\left[\log Q(\boldsymbol{z}|\boldsymbol{X}) - \log P(\boldsymbol{z}|\boldsymbol{X})\right] \tag{6.3}$$

In case of two Gaussian distributions, the KL divergence is defined as:

$$KL\left[\mathcal{N}_Q(\boldsymbol{\mu}_Q, \boldsymbol{\sigma}_Q)||\mathcal{N}_P(\boldsymbol{\mu}_P, \boldsymbol{\sigma}_P)\right] = \log\frac{\boldsymbol{\sigma}_P}{\boldsymbol{\sigma}_Q} + \frac{\boldsymbol{\sigma}_Q^2 + (\boldsymbol{\mu}_Q - \boldsymbol{\mu}_P)^2}{2\boldsymbol{\sigma}_P^2} - \frac{1}{2} \tag{6.4}$$

If the target distribution $P(\boldsymbol{z}|\boldsymbol{X})$ is a multivariate Gaussian distribution with $\boldsymbol{\mu}_P = 0$ and $\boldsymbol{\sigma}_P = 1$ (i.e. $\mathcal{N}(0,1)$) and the encoder predicts $\boldsymbol{\mu}_Q$ and $\boldsymbol{\sigma}_Q$ to model $Q(\boldsymbol{z}|\boldsymbol{X})$ as $\mathcal{N}(\boldsymbol{\mu}_Q, \boldsymbol{\sigma}_Q)$, then the KL divergence simplifies to

$$\begin{aligned} L_{KLD} = KL\left[\mathcal{N}_Q(\boldsymbol{\mu}_Q, \boldsymbol{\sigma}_Q)||\mathcal{N}(0,1)\right] &= -\log\boldsymbol{\sigma}_Q + \frac{\boldsymbol{\sigma}_Q^2\boldsymbol{\mu}_Q^2}{2} - \frac{1}{2} \\ &= \frac{1}{2}\left(\boldsymbol{\sigma}_Q^2\boldsymbol{\mu}_Q^2 - 1 - 2\log\boldsymbol{\sigma}_Q\right) \end{aligned}$$

Interested reads may find a formal proof of the formulas above and a more detailed derivation in the original paper [104]. Thus, the loss for the variational auto-encoder is defined as:

$$\begin{aligned} L_{\text{VAE}} &= L_{\text{reconstruction}} + \beta \cdot KL\left[\mathcal{N}_Q(\boldsymbol{\mu}_Q, \boldsymbol{\sigma}_Q)||\mathcal{N}(0,1)\right] \\ &= \underbrace{\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i)^2}_{\text{reconstruction loss}} + \lambda \cdot \overbrace{\frac{1}{2}(\boldsymbol{\sigma}_Q^2\boldsymbol{\mu}_Q^2 - 1 - 2\log\boldsymbol{\sigma}_Q)}^{\text{KL divergence}} \end{aligned}$$

where β is a factor to weight the KL divergence. Training an variational auto-encoder with this loss functions can lead to very good reconstructions. In the case of horizontal self-organisation, however, a well-formed

latent space is of more interest than a excellent image reconstruction: Latent representations are extracted from the latent space and made available to neighbouring models. Based on these latent representations, multiple variational auto-encoders should agree on a suitable image representation. Variational auto-encoders have the notorious problem that the KL divergence term becomes vanishingly small during training [262]. This issue is known as the KL vanishing problem. This problem can be alleviated by applying annealing schedules to the KL term (i.e. by changing β over time). In this thesis, monotonic annealing is used as proposed by Ref. [262] since this seems sufficient. However, for a slight increase in computational costs, cyclic annealing strategies might lead to even better results [263]. These annealing strategies are shown in Figure Figure 6.1. When β is small, the model is forced to focus on reconstructing the input rather than minimizing the KL loss. When β is increased, the model gradually improves the shape of the data distribution in the latent space.

[262]: Bowman et al. (2016)

[263]: Fu et al. (2019)

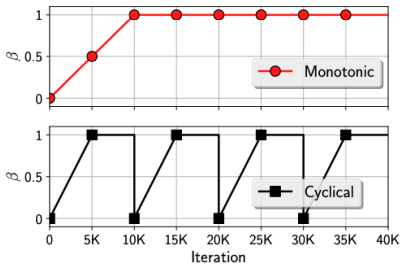

Figure 6.1: Two annealing strategies for the β term that weights the KL divergence in the loss function of variational auto-encoders. The upper graph shows a monotonic increase of β, the lower graph a cyclical annealing strategy. The picture is from [263].

In this thesis, 4 variational auto-encoders are used. Each VAE consists of an encoder, two fully-connected layers to calculate μ and σ, and a decoder. The encoder consists of 3 convolutional layers with 32, 64, and 128 channels. Each convolutional layer has a stride of 2, halving the size of the input in each layer. The decoder has the inverse structure of the encoder, i.e. 3 transposed convolutional layers with 128, 64, and 32 channels. The fully connected layer for predicting μ and σ have 32 neurons. The network architecture is shown in Figure Figure 6.2.

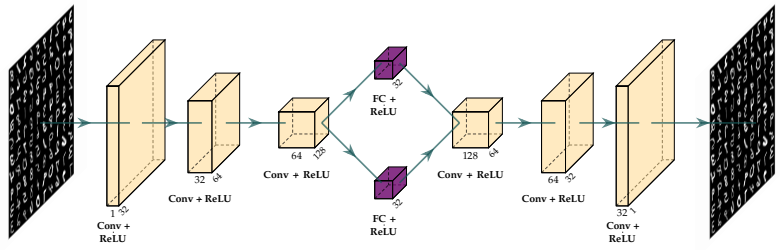

Figure 6.2: Architecture of the variational auto-encoder used for horizontal self-organisation.

Each VAE is trained independently with the goal to minimize the loss function as described in equation (1). Adam [260] is used as optimizer with a learning rate of $1 \cdot 10^{-3}$ and the mini-batch size is 32.

[260]: Kingma et al. (2017)

6.1.1 Predicting bigger Patches

Applying a monotonic annealing to the KL divergence weight term β is the first measure to improve the latent space distribution. Another measure is to predict bigger patches. The VAEs have a very limited field

of view and cannot distinguish some of the digits on their own (c.f. Section 6.1.2). However, by predicting bigger patches, the VAEs can be encouraged to better distinguish similar looking patches. For example, the patches for the digits 4, 5, and 6 look very similar for the first VAE that receives the patch extracted from the top left corner of the samples (c.f. Figure 6.5). Since the target prediction of these two digits is similar, they are located closely together in the latent space. When bigger patches or the entire image is predicted by a VAE, the target predictions of the digits 4, 5, and 6 become different. Thus, while similar latent representations are sufficient to predict only the top-left patch of these digits is sufficient, different latent representations are needed to predict bigger patches or the entire image. As a consequence, they are not mapped that closely together in the latent space anymore. Thus, predicting bigger patches helps to push apart latent representations of objects that have a high similarity patch-wise but a low similarity image-wise.

In this thesis, an additional layer is used in the decoder to predict larger patches. In fact, the last layer with 32 channels and a stride of 2 is used twice, so the encoder still consists of 3 convolutoinal layers, but the decoder has 4 transposed convolutional layers with 128, 64, 32, and 32 channels. This doubles the size of the decoder output. This architecture is shown in Figure Figure 6.3.

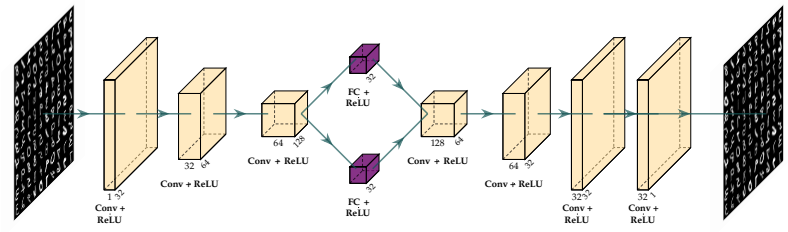

Figure 6.3: Architecture of the variational auto-encoder used for horizontal self-organisation with bigger field-of-view up-sampling.

When using 4 VAEs, the input is divided into 4 patches. But if the decoder makes the output twice as large as the input by up-sampling, then the entire image is predicted. Thus, in the case of this architecture with 4 VAEs, the task is to predict the entire image based on a patch that is only a quarter of the image. This is visualized in Figure 6.4.

6.1.2 Communication

The VAEs receive patches of the input image and thus have a very limited field-of-view on the image. Figure Figure 6.5 visualizes the patches that are fed into the VAEs when the MNIST data set [18] is used. The first row shows the average over all images of the same class. Thus, this images are rather representative for the data set. However, none of the models receives the entire image as input. Instead, the patches shown in 2nd to the 5th row are fed into the VAEs. The patches, which only depict a quarter of the image, look rather similar for some classes. For example, the top-left patches of the classes 0 and 9, as well as the classes 4, 5, and 6 look very similar. This means that these classes are placed at the same location in the latent space and thus are not separable by *one* single VAE on its own. Therefore, communication with neighbouring models is necessary to agree on an image representation.

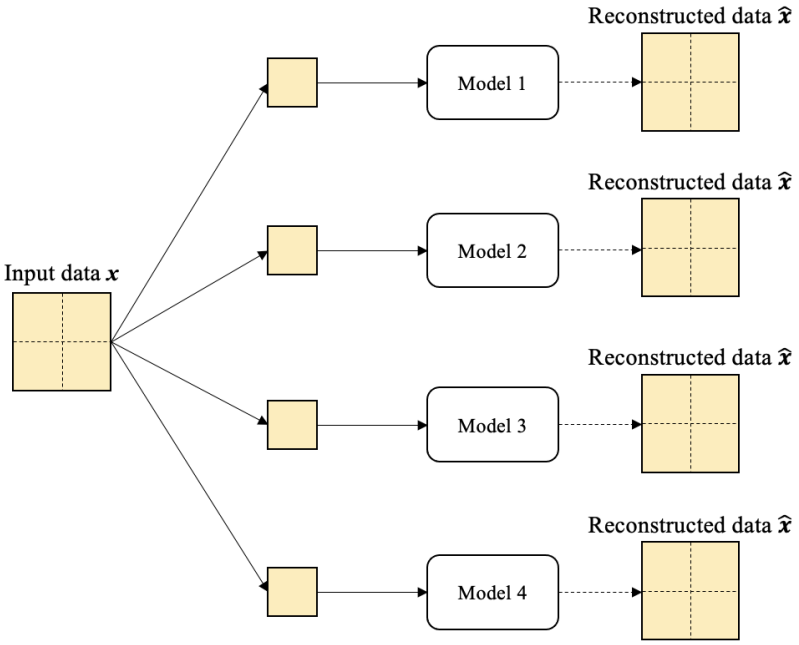

Figure 6.4: Four models receive a quarter of the image as input patch to predict the entire image.

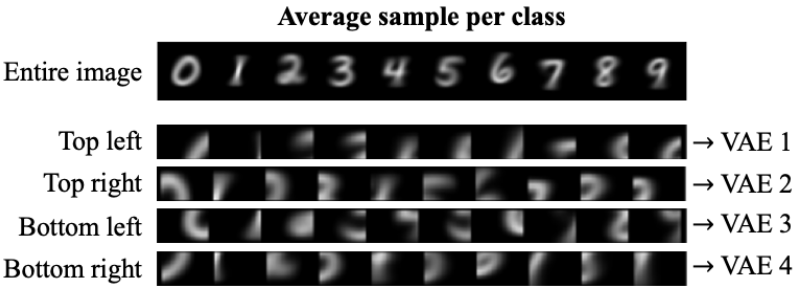

Figure 6.5: The average of all samples in the MNIST data set for each class. The first row shows the average of all samples, the 2nd to 5th row the average per patch that is fed into the models.

This connection maps well to the biological model: The single VAEs can be seen as neurons or neuron groups. At the beginning, they have many suggestions as to which classes the patch they read could belong to. Through communication with neighbouring VAEs, however, certain suggestions are constantly ruled out because they are not supported by neighbouring VAEs. Thus, out of many suggestions, only the most valid ones are retained, resulting in an image representation. The biological model behaves similarly: At first, many neurons are active because they are excited by the captured image. However, this strong activation is quickly becoming sparse as only the neurons that support each other remain active. This leads to the emergence of higher-level features (i.e. net fragments) that are representative for the captured input.

TODO: add source from Christoph

Thus, the communication between VAEs is a lateral support of representations. This corresponds to the lateral connections introduced in section Section **??**: Each VAE (which can be interpreted as a neuron) makes several suggestions as to what the patch could represent. Only the representations that are laterally supported are retained and the other representations are discarded. To learn representations in an unsupervised manner (and to keep one of the strengths of auto-encoders), no probability of predefined labels should not be communicated as this requires label information. Instead, different types of communication are

proposed that work without labels: communication based on the latent representations, communication based on the reconstructed images, and communication via a dedicated communication channel.

All these types of communication take place over two or more time-steps. First, image patches are reconstructed by all VAEs. As a result, each VAE generates information in latent space. This information can be communicated to the neighbouring VAEs. Thus, in a second time-step, the VAEs have more information at available: their own prediction based on the reconstructed image patch and information from neighbouring VAEs. This additional information allows them to improve their own prediction. The process can be repeated either over a fixed number of time-steps or until the network reaches an attractor state (i.e. the latent representations do not change anymore).

Model Heads

Each VAE learns the mapping from a patch to a latent space with Gaussian distribution, i.e. obtains a $\boldsymbol{mu}$ and $\boldsymbol{sigma}$ for each sample that is used to predict a reconstructed version of the input. The arrangement of the representations within the latent space is not predetermined by an external system but is formed by the model itself in the manner of self-organisation. A consequence of this is that $\boldsymbol{mu}$ and $\boldsymbol{sigma}$ have a different meaning in the latent spaces of different VAEs. Thus, one VAE is not able to interpret the Gaussian parameters of another VAE out-of-the-box.

In this thesis, we propose to learn a linear transformation to map the Gaussian parameters of one VAE to the Gaussian parameters of another VAE. The mapping from $\boldsymbol{mu}_1$ to $\boldsymbol{mu}_2$, respectively from $\boldsymbol{sigma}_1$ to $\boldsymbol{sigma}_2$ is done by the simple transformation

$$\hat{\mu}_2 = w_{m12} \cdot \mu_1 \tag{6.5}$$

resp.

$$\hat{\sigma}_2 = w_{s12} \cdot \sigma_1 \tag{6.6}$$

The weights w_{mij} and w_{sij} between VAE i and VAE j are learned by minimising the mean square error with gradient descent between the predicted and true Gaussian parameters, i.e. between μ_j and $\hat{\mu}_j$, resp. σ_j and $\hat{\sigma}_j$. This process is illustrated for VAE 1 in Figure Figure 6.6. First, the Gaussian parameters of each model must be predicted for the same sample. Afterwards, the Gaussian parameters of one model can be used to predict the Gaussian parameters of the other models with a linear transformation.

This makes it possible to predict the Gaussian parameters of all VAEs on the basis of one patch and to reconstruct the entire image. If each VAE predicts the Gaussian parameters of the other VAEs, then each VAE receives further suggestions from neighbouring VAEs in addition to the parameters calculated by the encoder. Thus, each VAE has its own calculated Gaussian parameters as well as suggestions from neighbours

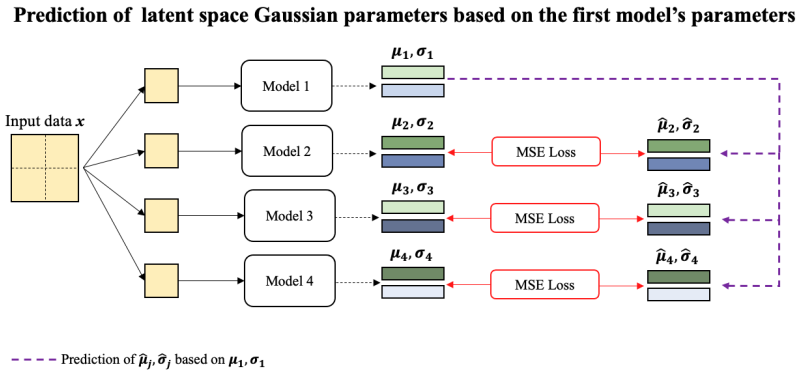

Figure 6.6: Visualization of how the first VAE can predict the Gaussian parameters of the other VAEs.

about what its Gaussian parameters might be. This allows a VAE to correct its prediction. This is particularly interesting because the suggestions made by neighbouring VAEs are based on a different patch. For example, the VAE that sees only one patch from the top left of the image cannot distinguish the digits 5 and 6. The VAE that sees the patch at the bottom left, on the other hand, is able to distinguish these digits. Thus, the VAE at the bottom left can help the VAE at the top left to choose a better latent representation.

TODO: At the moment, various experiments are still ongoing to find out how this can be implemented (therefore, not yet explained in more detail).

Reconstructed Images

As described in Section 6.1.1, the prediction of larger patches helps to better arrange the latent space. The prediction of larger patches can also be used for communication with neighbouring VAEs. Instead of extracting representations from the latent space and transmitting them to neighbouring VAEs as described in the last section, the reconstructed image can also be exchanged. This is done by predicting a part of the image in a first time-step and by adding this prediction to an additional input channel of another VAE in the next time-step. Various data can be forwarded to other VAEs:

Reconstructed Patch Each VAE reconstructs its input patch and optionally a part of the neighbourhood. It tells the other VAEs what the reconstructed version of its input patch looks like. Information about the optionally reconstructed neighbourhood is not communicated to other VAEs. Thus, a reconstructed version of the input is sent to other VAEs and added as a new channel to their input. This process is exemplified for the first VAE in Figure 6.7. The problem with this approach is that in the first time-step all VAEs reconstruct their patch and the reconstructed version is communicated to the neighbouring VAEs. As a result, in the second time-step, each VAE receives information about the entire image and thus has access to global instead of only local image information[2].

Neighbouring Patch Instead of communicating the reconstructed input patch to the neighbours, a prediction can be made of what the input patch of the neighbour looks like and this prediction can be communicated. This has the advantage that each UAE receives

2: even with a very large number of VAEs and communication restricted to the local neighbourhood, information about the entire image propagates to all VAEs over time, provided enough time-steps are executed

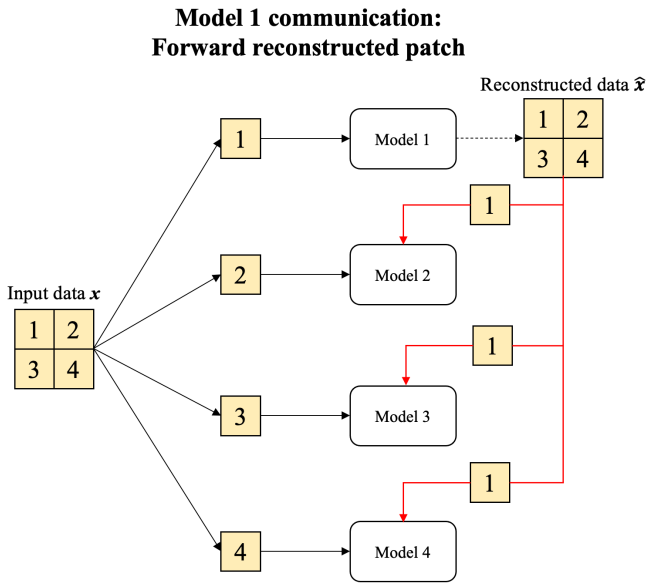

Figure 6.7: Communication between VAEs by forwarding reconstructed patch, example on the basis of the first VAE: The first VAE predicts the image and forwards the reconstructed patch (the same patch that was fed into the model) to the other VAEs.

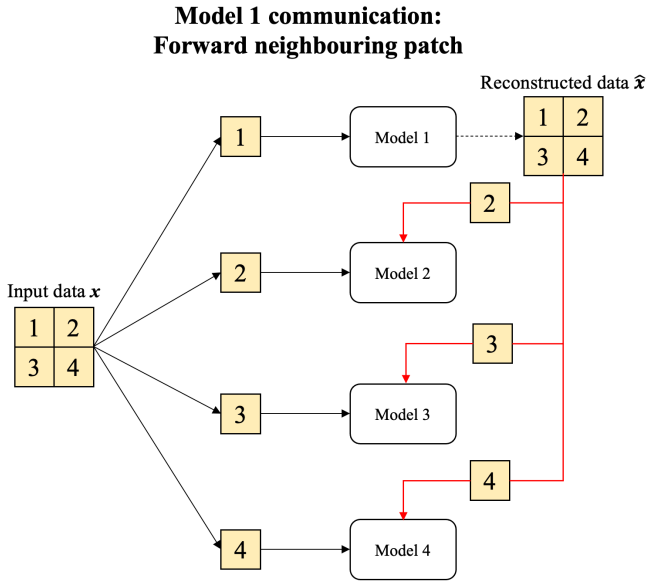

Figure 6.8: Communication between VAEs by forwarding prediction of neighbouring patches, example on the basis of the first VAE: The first VAE predicts the image and forwards a prediction of how the neighbourhood could look like to the other VAEs. Thus, each other VAE receives a patch and a prediction of the first VAE how this patch could look like as input.

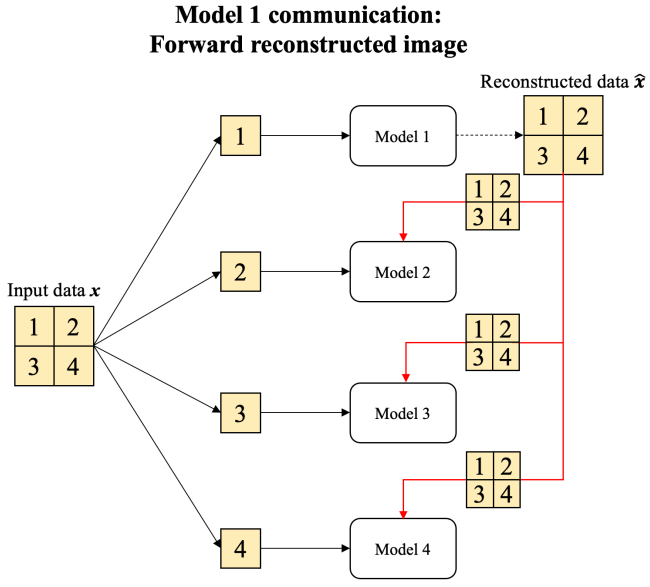

Figure 6.9: Communication between VAEs by forwarding prediction of the entire image, example on the basis of the first VAE: The first VAE predicts the image and forwards the prediction of the image to the other VAEs.

different versions of the same patch as input and thus has no global image information. This process for the first VAE is visualised in Figure 6.8. Intuitively, this can solve the following problem: If a VAE cannot decide which digit to predict, neighbouring VAEs can help it by predicting its input patch in a form that is more similar to one of the digits and thus support the VAE by making its decision.

Entire image The simplest version is when each VAE predicts the entire image and communicates this to all other VAEs. In the first time-step, only local information is available, in the second time-step, various predictions of how the image could look are additionally provided. Each VAE can then correct its own prediction based on these predictions of the other VAEs. This process is represented in Figure 6.9. Identical to "reconstructed patches", global image information is made available to each VAE. This problem can be alleviated if for each VAE i that receives the input patch $\boldsymbol{x}^{(i)}$, the part from its prediction of the whole image corresponding to $\boldsymbol{x}^{(i)}$ is masked. Otherwise, an input patch of a VAE is fed through the encoder and decoder and then communicated to the other VAEs. Then implicitly reconstructed data and not just neighbourhood predictions gets to each VAE as input.

Communication Channel

Another way to communicate is through a dedicated communication channel. For this purpose, each VAE has not only 1 input and 1 output channel but 4 input and output channels. The input patch is fed into the first channel and a reconstructed version of it is predicted in the first output channel. This corresponds to a classic VAE and the loss function remains identical. The remaining three channels are for communication: Each VAE generates a separate communication output for its 3 neighbouring VAEs on the remaining 3 channels. Each of these channels is then stacked to the input of the corresponding VAE. For example, the first VAE receives a given patch on the input channel 1 and communication data from the 2nd, 3rd, and 4th VAE on the input channels 2-4. Based on this information, it outputs a reconstruction of the patch on the output channel 1 and communication data to the 2nd, 3rd, and 4th VAE on the output channels 2-4. Thus, there is a dedicated two-way communication channel between each neighbouring VAE. This is shown in Figure 6.10.

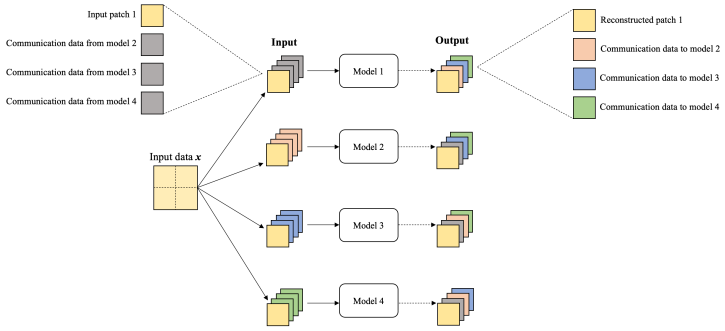

Figure 6.10: Each VAE has one and output channel to receive and reconstruct a given image patch. The other input and output channels are used for communication: Each VAE has for each other VAE a separate output channel to send messages and an additional input channel to receive messages.

The loss function remains the same for the first output channel since the goal is still the same, i.e. to reconstruct the output as good as possible. The only difference is that model can now access 4 input

channels to reconstruct the input instead of one channel. However, the communication channels must be trained in order for them to be helpful for input reconstruction. Therefore, an additional loss term is applied on the 3 output communication channels. Thereby, the idea is to optimize the output of these communication channels in a way so that the other VAEs can better reconstruct their patch.

Let $V = V_1, ..., V_n$ be the set of n VAEs. Each VAE has n output channels, one channel for the reconstructed image and $n-1$ channels for communication. Furthermore, each VAE V_i has a loss L_{VAE_i} based on the reconstruction goodness and the shape of the latent space (c.f. equation (1)). The 3 communication channels of a VAE i can influence the loss L_{VAE_j} of VAE j (i.e. help to improve the loss by providing better information). How helpful the communication channels are can therefore be determined by the average of the loss of the other VAEs.

$$L_{c_i} = \frac{1}{n-1} \sum_{n}^{j=1} L_{\text{VAE}_j}, \text{ if } i \neq j \tag{6.7}$$

Thus, the loss of a VAE with communication channel is

$$L_{\text{VAE}_{C\,i}} = L_{\text{VAE}_i} + \beta \cdot L_{\text{KLD}_i} + \beta_2 \cdot L_{c_i} \tag{6.8}$$

where β_2 is a weight factor for the communication channel. Thus, each model optimizes its own reconstruction error, the shape of the latent space, and tries to improve the latent space and reconstruction error of other VAEs.

TODO: This has not been implemented yet (therefore, not explained in more detail).

6.1.3 Class Prediction

The VAEs are examined whether they are suitable for predicting the class label. To do this, all VAEs are first trained until they can reliably reconstruct images and the latent space is well formed. After training, the average value of $\boldsymbol{\mu}$ is determined for each class $c \in C$ from the training set and each VAE v. This average value is called $\boldsymbol{\mu}_{\text{avg}_c}$ and is defined for n samples as

$$\boldsymbol{\mu}_{\text{avg}_v c} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\mu}_i \tag{6.9}$$

This average value can be considered as the "cluster centre" of a class in latent space. In addition, $\boldsymbol{\mu}_{\text{avg}_v c}$ is a world model of a class. To make a prediction, a given sample is matched against these world models per class. As with vertical self-organisation, the cosine similarity between a sample and the class prototypes is used to determine their similarity. To predict a sample, first its $\boldsymbol{\mu}_v s$ is determined. Then, this sample is compared with all classes by computing the cosine similarity:

$$\cos_{vsc} = \cos(\mu_v s, \mu_{\text{avg}_v c}) = \frac{\mu_v s \cdot \mu_{\text{avg}_v c}}{\max(||\mu_v s||_2, ||\mu_{\text{avg}_v c}||_2)} \tag{6.10}$$

Thus, the cosine similarity $\cos_{vsc}$ between each class c' prototype and the sample s is calculated for each VAE v. Afterwards, the average class c with the highest average cosine similarity between the sample activations $\mu_v s$ and the class prototypes $\mu_{\text{avg}_v c}$ is used as prediction.

$$\arg\max_{c \in C} \frac{1}{4} \sum_{v=1}^{4} \cos_{vsc} \tag{6.11}$$

Similar to vertical self-organisation, VAE's with a higher confidence have a higher cosine similarity between a class prototype and a specific class and thus have more influence on the class prediction than if the confidence is lower and the cosine similarity between the sample and all class prototypes is similarly large.

TODO: This has to be improved by also considering the standard deviation

TODO: Another improvement is to use multiple prototypes per class as digits may be written differently (depending on the person) and thus look different

VQ-VAE

In the case of classification, the latent space has to be shaped in a way that allows to assign a latent representation to a class. This can be simplified if the encoder maps the input image to a discrete variable in the latent space. Vector quantised variational auto-encoders (VQ-VAE) [264] model such a discrete latent space by using vector quantisation (VQ). Vector quantisation is a method that maps multi-dimensional vectors to a finite set of "code"-vectors. An image is fed into the encoder to obtain the encoder output z_e. Afterwards, a nearest neighbour lookup is made to find the code-vector that is most similar to z_e. The code-vector, also called the quantized vector z_q is fed into the decoder to reconstruct the image.

[264]: Oord et al. (2017)

TODO: This has not been implemented yet (therefore, not explained in more detail).

6.2 Results

As for vertical self-organisation (.c.f Section 5.2.1), the MNIST [18] and MNIST-C [261] data sets are used to evaluate the models.

[261]: Mu et al. (2019)

Different models are trained to evaluate this architecture. A single VAE is used as the baseline (i), which receives the entire image as input and reconstructs it. This VAE is obviously trained without time steps and communication. Horizontal self-organisation is done using 4 VAEs (2×2) in different settings: Four independent VAEs are trained (without time steps and communication), each VAE receives a non-overlapping image

patch as input and either reconstructs the received patch (ii) or predicts the entire image (iii). Furthermore, different types of communication are investigated: The VAEs communicate the reconstructed *input patch* to the other VAEs (according to Figure 6.7). This communication takes place over 4 time steps and the VAEs are trained by either reconstructing the input patch (iv) or predicting the entire image[3] (v). In another version, each VAE predicts the whole image and forwards the predicted *input patch of the neighbour* to each VAEs over 4 time steps (vi) (c.f. Figure 6.8). Finally, the VAEs are also used to predict the whole image over one time step and the *predicted image* is forwarded to the other VAEs (e.g. Figure 6.9) (vii). Table **??** gives an overview of the models described.

3: but despite the prediction of the entire image, only the prediction of the reconstructed patch is communicated to the other VAEs.

Table 6.1: Different models for evaluating horizontal self-organisation: The first column shows a number used as a reference id in this thesis. The second column shows the number of VAEs used, the third column whether the input field or the whole image is reconstructed, the fourth column what is communicated to the neighbouring VAEs, and the fifth column over how many time steps the communication takes place.

No.	n VAEs	Reconstruction	Communication	Time-Steps
i	1	Image	-	-
ii	4	Patch	-	-
iii	4	Image	-	-
iv	4	Patch	Input Patch	4
v	4	Image	Input Patch	4
vi	4	Image	Neighbouring Patch	4
vii	4	Image	Image	1

The number of time steps is a design decision: obviously, the models without communication do not need time steps. In the case where each VAE predicts the entire image and forwards it to the other VAEs, only one time step is used. This means that the entire image is predicted once and then forwarded to the other VAEs. It has been observed that one time step is sufficient and that the predictions do not change in further time steps. This could be due to the fact that all relevant information is already transmitted within one time-step and no further communication cycles are necessary. However, if only a patch of the image is forwarded to the other VAEs, the predictions may change over several time steps. In this thesis, 4 time steps are used when only patches are forwarded to the other VAEs. It is observed that the predictions usually do not change after 4 time steps and thus using 4 time steps seems enough. When only patches are forwarded, the model contains much more dynamics: a VAE predicts something, communicates it to its neighbours and receives data from its neighbours. In a subsequent time step, the VAE has more information and can correct its prediction, which in turn can lead to a correction of a neighbour's prediction. Thus, the communication occurs over several time steps in which the VAEs correct each other. An example of four predicted samples is shown in Figure 6.11: The left side shows the initial prediction of the model (vi) and the right side the prediction of the same image after 4 time steps. This visualisation indicates that the predictions are improved thanks to the communication. The investigation of the reconstruction error confirms the efficiency of the communication: The initial prediction without communication has an average reconstruction error of 0.23, and the prediction after 4 time step has an error of 0.16.

Of particular interest in this evaluation is not how well the samples are reconstructed but how well the latent space is shaped. Visualising the μ values is a simple way to investigate the latent space. Therefore, the μ vectors with a length of $n = 32$ are reduced to 2 dimensions using t-SNE [265] and visualised. Since 25 VAEs are trained in this evaluation, visualising all t-SNE plots is infeasible. Instead, Figure 6.12 only shows

[265]: van der Maaten et al. (2008)

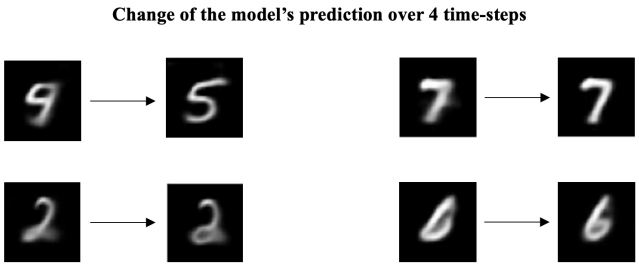

Figure 6.11: Four random predictions of model (vi) over 4 time-steps: The initial prediction is shown on the left and the prediction after 4 communication cycles on the right.

the plot of the baseline, the plot of the model (ii) as an example of an architecture without communication, and the plot of the model (vi) as a representative example of an architecture with communication.

As expected, the latent space of model (i) is relatively well organised and the classes are distinguishable within the latent space. In this model, only one VAE is used, which receives the entire image as input which simplifies the organisation of the latent space. Model (ii) consists of 4 VAEs without communication. Its latent space is less well organised: the classes are no longer clearly identifiable as clusters. However, it is interesting to note that each VAE can separate certain classes better than others because of the different patch it receives as input. Considering all 4 VAEs, each class can be relatively well identified by at least one VAE, while no VAE can identify all classes. This indicates the benefit of a communication between the VAEs. Applying the proposed measures (i.e. communication and predicting the whole image instead of reconstructing the input patch), the arrangement of the latent space for each VAE improves significantly: almost every VAE is able to separate the classes. This supports the usefulness of the proposed measures.

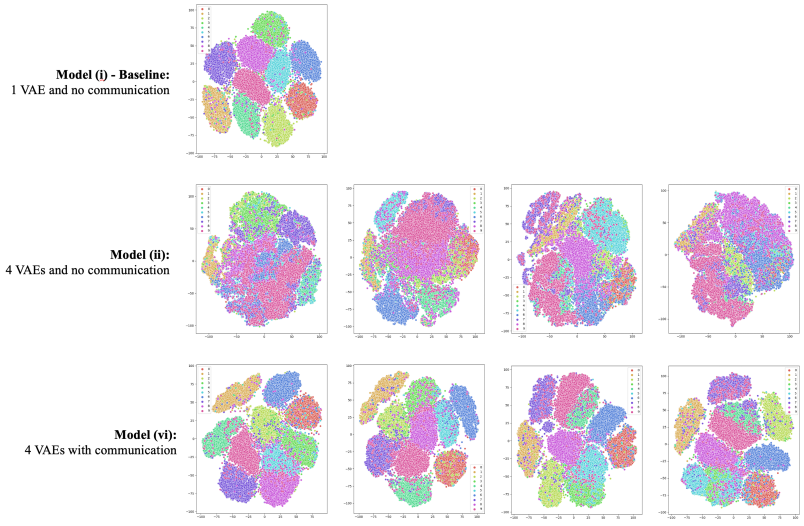

Figure 6.12: t-SNE plot of the μ values of four different models: The first row shows the baseline model (1 VAE), the second row the model (ii) that consists of 4 VAEs without communication, and the third row the model (vi) that uses 4 VAEs with communication channels. Each point in the plot corresponds to a sample, the samples are coloured according to their class.

Another way to evaluate the latent space is to measure its suitability for classification. This can be done by applying a k-means clustering and then calculating the normalised mutual information (NMI) score between the cluster centres and the labels. This metric allows the comparison between cluster assignments and labels by calculating the consistency. This means that it measures how clearly a cluster can be assigned to a label. Table **??** shows the NMI scores of the models. The NMI score is difficult to interpret as a number, and it is unclear exactly what, for example, a score of 0.6 means for model (i). However, a relative comparison of the

scores is of interest for this evaluation. The model (i) is a single VAE that receives the entire image as input, not just a patch of it. Therefore, this model is assumed to have a well-formed latent space, and the value of 0.6 is close to an upper bound for a good latent space in this setting.

It is evident that the latent space is significantly better shaped when the whole image is predicted and not only the input patch is reconstructed. For example, models (ii) and (iii) as well as (iv) and (v) are the same models but one of the models reconstructs the input patch and the other one predicts the entire image. The models that predict the whole image are significantly better: the NMI value increases on average from 0.36 to 0.53 (model (ii) → model (iii)) and from 0.37 to 0.59 (model (iv) → model (v)) respectively.

A slight improvement through communication can also be observed. Model (iii) consists of 4 VAEs without communication and predicts the whole picture. It achieves an NMI value of 0.53. The models with communication are all better, reaching NMI values of 0.59 (model (v)), 0.54 (model (vi)) and 0.54 (model (vii)). However, the improvement is not as clear as for the prediction of the whole image. This is probably due to the fact that the image patches are too large and more VAEs should be used. For example, model (iii) has a very high NMI score compared to the baseline, indicating that this model is able to classify and therefore the image patches contain too much image information. This makes communication less important and the results are not significantly improved by adding communication.

Table 6.2: The NMI score between the μ cluster centres and the ground-truth label. The clusters are calculated by using the k-means algorithm. Most models consist of several VAEs, therefore the average, minimum and maximum NMI values are reported for all VAEs per model.

No.	Avg. NMI	Min. NMI	Max. NMI
i	0.60	0.60	0.60
ii	0.36	0.29	0.39
iii	0.53	0.49	0.56
iv	0.37	0.33	0.41
v	0.59	0.58	0.61
vi	0.54	0.49	0.60
vii	0.54	0.49	0.59

Finally, the proposed classification method is evaluated. For this purpose, the accuracy on the MNIST and MNIST-C data sets is determined for each model. The results are shown in Table **??**. For each VAE of a model, the cosine similarity according to equation (**??**) between a sample and a class prototype is calculated. Thus, the accuracy can not only be calculated for the entire model (i.e. all 4 VAEs together), but also for each VAE independently. Table **??** shows the average accuracy per VAE in the column "Avg. VAEs". Furthermore, by calculating the average cosine-similarity a voting over all VAEs is performed and the actual model accuracy os obtained. The accuracy of this voting is shown in the column "Overall". The voting works exceptionally well: for example, model (ii) has 4 VAEs with an average accuracy of 62.4% on the MNIST dataset. The best VAE has an accuracy of 66.1%, the worst 59.7%. By voting, the overall accuracy increases to 81.7%, which is significantly better than the best VAE.

On the MNIST data set, the base model (i) is significantly outperformed by some models in terms of accuracy, but robustness (measured on MNIST-C) is not significantly improved by using multiple VAEs. As with the evaluation of the NMI score, it can be observed that models that

predict the entire image perform better than models that only reconstruct a patch of the image. Furthermore, the accuracy of the models without communication is generally very high, suggesting that the image patches are too large and more VAEs should be used.

No.	Avg. VAEs MNIST	Overall MNIST	Avg. VAEs MNIST-C	Overall MNIST-C
i	-	85.2%	-	63.4%
ii	62.4%	81.7%	42.3%	60.9%
iii	78.2%	89.9%	45.4%	64.9%
iv	63.8%	82.6%	41.8%	60.5%
v	86.1%	87.6%	62.4%	65.2%
vi	88.4%	91.1%	50.4%	59.2%
vii	78.5%	90.0%	46.8%	64.2%

Table 6.3: The accuracy of different models on MNIST and MNIST-C. For both data sets, the average accuracy per VAE and the overall accuracy is reported. The average per VAE is the average accuracy when each VAE makes a prediction on its own (i.e. choosing the highest cosine similarity per VAE according to equation (**??**)). The overall accuracy is calculated by averaging the cosine similarity according to equation (**??**).

The accuracy was calculated by comparing a sample with *one* prototype per class. However, using only one prototype seems to be not enough. For example, when looking at the t-SNE plots in Figure 6.12, the same class is divided into more than one cluster. The calculation of the NMI score also indicates that one prototype per class is insufficient: When more clusters are used for k-means clustering, the NMI scores become higher. This is also in line with the analysis of the data and the findings of vertical self-organisation (c.f. Section 7.2.1).

Future Work & Conclusion 7

Deep learning is far from being a system with human-like intelligence. Consequently, there is a massive amount of work to be done in the future to get closer to this ultimate goal. However, the implementations presented in this thesis are far from such a system. Consequently, a conclusion is drawn and next steps are proposed for the models presented in this thesis, but a long-term vision is presented as well. Specifically, some insights are give to the neuroscientific concept that seem very promising in section Section 7.1 and future work is described for vertical self-organisation in Section 7.2.1 as well as for horizontal self-organisation in Section 7.2.2. Afterwards, in Section 7.3, a new model is presented on a very abstract level, which could work better intuitively, but a concrete implementation is unclear. In Section 7.4 the usefulness of representations is discussed. In the last Section 7.5, the author gives a personal opinion about the future of AI, which is not supported by scientific arguments but might be interesting for (motivated) readers.

7.1 Neuroscientific Concepts

In Chapter 4, several concepts from neuroscience are identified that are believed to be fundamental to biological intelligence. However, since the discrepancy between the biological model and the artificial model is large, it is difficult to incorporate these biological concepts into a computer system. In this thesis, concrete suggestions are made how such concepts can be realised as a concrete implementation. By implementing these concepts in the form of horizontal and vertical self-organisation, it is demonstrated that these ideas can facilitate the learning process. The evaluation of the suitability of the proposed implementation was very iterative: many concepts were tested but discarded in the course of the work because they did not add a lot of value to existing systems. For example, a lot of time was spent on generating sparsity over several time-steps as it is done in the human brain. However, it was found that sparsification of representations over several time-steps has no advantage without additional system dynamics. This could be because the data is static and already contains all the information. Therefore, the result is the same if the sparsification is done in one time-step than if it is done over several time-steps.

The most promising concepts identified in this work for improving deep learning systems are self-organisation, net-fragments, sparsity, lateral connections, continuous input, and embodiment. For each of these mechanisms, a concept for implementation is proposed. However, these proposals are the interpretation of the author and are only one of many possible solutions. So it could be that these concepts can also be interpreted differently and an alternative implementation could lead to better results.

7.2 Implemented Models

The concepts from neuroscience are implemented in two specific ways, these are called vertical and horizontal self-organisation. It is important to note that the implementation of *neuroscientific concepts* is the main focus of this thesis and not to push scores like accuracy on benchmarks. In general, a comparison with end-to-end backpropagation of error is not appropriate in this field: Backpropagation has been optimised for over 30 years by many institutes and even more researchers, and therefore alternative learning concepts cannot be expected to outperform backpropagation of error instantly. Thus, backpropagation of error is note considered an appropriate baseline.

Rather, the proposed implementations show that this type of implementation of neuroscientific concepts can facilitate learning. However, the implementation has not yet been optimised and incorporated in big networks that might perform on the level of existing systems. Rather, these implementations are to be understood as a basis for further work, which, for example, deals with the concepts presented in Section 7.3, Section 7.4 and Section 7.5. Thus, this thesis is to be understood as a preliminary work of a larger research project to develop new concepts that improve AI in a long-term way.

7.2.1 Vertical Self-Organisation

TODO: this section is not yet complete as many experiments are still ongoing...

Currently, a sample is compared to the average activation of each class for classification. This average activation can be seen as a kind of object prototype within a world-model. Thus, it is compared to which prototype a sample fits best. The problem is that a single object prototype is most likely not sufficient. As an example, Figure 7.1 shows different versions of the digit 4 that can be found in the MNIST data set. This demonstrates that a digit can be written in different ways[1]. Consequently, the digits and thus the activations look different for samples from the same class and the use of more than one prototype per class seems helpful. Thus, instead of calculating the average activation per class, it could help to divide the activations per class into representative groups using a hierarchical clustering algorithm and to calculate the average activation (i.e. the cluster centre) per group. By doing so, multiple representative prototypes per class are obtained and the prediction could become lot more accurate.

1: this applies to all kind of data sets: Most objects can have various visual characteristics (size, color, shape) or look different from different viewpoints

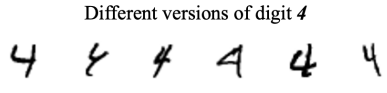

Figure 7.1: Different versions of the digit 4 in the MNIST data set.

The generated representations have so far only been examined for classification. However, since the objective function does not explicitly optimise the model for classification, it would be interesting to investigate what other information is contained in the representations. Furthermore, the representations are currently generated by a supervised approach, since the labels are used in the loss function. The use of labels is generally not

desirable because the manual creation of labels is time-consuming. An unsupervised (or self-supervised) approach, on the other hand, extracts features without using labels. Moreover, it is known that the brain also functions unsupervised to a very large extent and is thus biologically more plausible. Thus, it would be beneficial be the implementation of a loss function that does not rely on labels. Furthermore, the model should be tested on other data sets than MNIST to evaluate how well it can scale.

The proposed approach for vertical self-organisation uses a novel kind of proxy objective function. Models based on proxy objective functions are biologically more plausible and a hot research topic (c.f. Section 3.3). Recently, many algorithms have been proposed, such as the forward-forward algorithm [157], which have caused a stir in the community by challenging the classical backpropagation of error as optimisation algorithm. However, despite the fact that they are biologically more plausible, these algorithms don't provide many benefits yet. Models trained with backpropagation of error are still superior, especially on well-known tasks such as classification. One of the author's hope was that such algorithms have other advantages such as better robustness. However, the evaluation of the models presented in this work and other models such as the forward-forward algorithm have crushed this initial believes.

[157]: Hinton (2022)

7.2.2 Horizontal Self-Organisation

TODO: this section is not yet complete as many experiments are still ongoing...

TODO: Conclusion

Similar to vertical self-organisation (c.f. Section 7.2.1), the prediction accuracy could be improved by using more prototypes per class, as digits of the same class can look different. In the case of horizontal self-organisation, the μ values per VAE should be clustered hierarchically to obtain groups of similar looking digits per class. Afterwards, the average value per group can be calculated to obtain more prototypes per class.

A very important concept in the human brain is the prevention of early commitment[266] (c.f. Section 7.3), i.e. that a single entity such as a network layer not commits to something and then persists. Typical CNN architectures for classification have this problem by design, as they combine low-level features to higher-level features. With low-level features, it is already determined which feature is within the image before they are combined to higher-level features. The proposed architecture for horizontal self-organisation reduces this problem by creating representations of image patches. The patches are that small that they contain only information to build low-level features. For example, the representations from a single VAE cannot be clearly assigned to a class and thus do not contain enough information to commit to an object. Therefore, the VAEs can be considered as low-level feature extractors such as the first layer of a neural network. To create higher-level features, communication with neighbouring VAEs (i.e. when looking at the big picture) is needed. Each VAE extracts low-level features (local view) and has to communicate with other VAEs (global view) to determine what they represent. Therefore,

[266]: Marr (2010)

this architecture is less susceptible to the early commitment problem than typical end-to-end trained neural networks. Furthermore, the representations are learned unsupervised. For classification, the labels are fed into the network *after* training in order to compute average class representations. Thus, no class representation but an image representation is learned during training. This further reduces the problem of individual VAEs committing to a class too early.

In future work, this effect should even be intensified by further reducing the size of the patches and using more VAEs. It can be assumed that with 16 VAEs (4×4) the field of view is so restricted that each VAE can only produce very poor representations and these can only be assigned to a class in a random fashion. With more VAEs, interaction can also be restricted to local communication. With 2×2 VAEs, each VAE automatically receives information about the entire image when it communicates with its neighbours. With 4×4 VAEs, on the other hand, it is possible to have each VAE communicate only with its immediate neighbours. In this case, global image information must be propagated over several time-steps from one VAE to the next.

Future work should also address a key problem of this architecture: Currently, this concept only works with images in which the objects always appear at approximately the same place in the image. This can be remedied if the VAEs are applied at different places in the image, similar to a kernel in a convolutional layer. In this case, however, the same VAE sees all image data, which means that it implicitly receives not only local but global image information. This could reduce the independence of each VAE. Consequently, it should be investigated how this architecture can become translation invariant in a similar way as CNNs.

7.3 3-Staged Model

[2]: von der Malsburg et al. (2022)

[267]: Wolfrum et al. (2008)

[84]: Willshaw et al. (1979)

[268]: Wiskott et al. (1996)

[90]: Fernandes et al. (2015)

Christoph von der Malsburg, co-supervisor of this thesis and one of the main authors of the "Natural Intelligence" paper [2] which is the main inspiration of this thesis, is one of the pioneers of the theory of self-organisation of ordered fibre connections in the visual system [84, 90, 267, 268]. His theory is based on a 3-staged model. Currently, all people involved in this thesis are working on writing down this model and relating it to current deep learning architectures. This work has resulted from the collaboration between neuroscientists and computer scientists and the fact that there are often misunderstandings between these field due to a non-uniform technical vocabulary. Writing it down should enable non-neuroscientists to understand these exciting concepts better and to relate them to the context of deep learning. The resulting (hopefully publishable) paper will not be included as part of this Master's thesis, as it was not written by the author alone. Nevertheless, it explains interesting concepts that are promising for future research and highly relevant to this thesis.

Von der Malsburg's theory refers to a image perception model with three stages S1-S3, and projection fibres between these stages. The stage S1 refers to the first stage which is responsible for the extraction of features from images. In this stage, one principle is particularly important, namely

to avoid the "fallacy of early commitment". [266]. The idea is that local decisions should be only be taken if plausible in the light of high-level features while the high-level features can only be defined on the basis of low-level patterns. The brain can handle this very well, as the Gestalt psychology shows[2]. [269]. Modern deep learning networks can't deal with this and often recognise patterns in the first layers, committing to something specific that is actually something else in the light of the big picture. The core mechanism in the human brain to deal with this is, according to von der Malsburg, a mechanism based on lateral connections between neurons that are spatially close to each other. Initially, a large number of neurons are activated by the retina's signals. However, these neurons are immediately turned off if they do not have sufficient lateral support from neighbouring neurons. The lateral connections are thus there to support each other in order to remain active. This not only has the effect of "mutually confirming each other" but also helps to form higher-level features from low-level patterns. Intuitively speaking, and without neuroscientific correctness, one can imagine that one red, two brown, and one green pixel cluster are captured by the retina and active some neurons. The green pixel group receives too little lateral support and the corresponding neurons are quickly switched off again. The two brown and the red pixel groups support each other and remain active. But this is only because the two brown pixel groups are arranged in a circle at the same height and below them the red pixel group is in an elongated shape - a familiar pattern. If one looks at the big picture, one can see that this is a mouth and two eyes (locally, however, is remain unknown that this might be a face). It is important to notice that

- A global view on the whole pattern is necessary to tell what the local patterns are. For example, we don't see eyes and decide directly on eyes, but wait until we see the whole image to conclude that these brown pixel clusters could be eyes and maybe belong to a face
- Pixel constellations only support each other if they are arranged accordingly. For example, if all three pixel clusters were at the same height, we would not perceive this pattern as a typical face (the lateral support might be insufficient and the assumption that it is a face is rejected).
- The emergence of higher-level features is done in one layer through lateral connections and is not hierarchically formed across multiple layers as in deep learning.

Since these pixel groups keep each other active, neurons that represent insignificant features are continually switched off and thus implicitly higher-level features are formed. The lateral support, however, is limited to local patterns, i.e. it is not sufficient to recognise whole objects or scenes, but only parts of them. In S1, many higher-level features are precisely identified. In stage S2, however, these features are mapped to whole objects, for example faces (in terms of our example). These object representations in S2 are independent of transformation and position. The mapping of several low-level features to a concrete object is done by projection fibers. These implicitly remove transformations and position information from the features and assemble them into objects. Projection fibers can be interpreted as a kind of graph that combines features hierarchically, ignoring local distortions to create transformation

[266]: Marr (2010)

2: Gestalt psychology is about intelligent beings perceiving entire patterns, not individual components

[269]: Köhler (1929)

independent object detection. Stage S3, on the other hand, stores very abstract prototypes of such objects, and by matching S2 and S3, the captured object can finally be identified as the face of a specific person.

The model described contains very interesting concepts. Especially the prevention of early commitment and the use of dynamic fibres seem very promising. A concrete implementation that works for complex images like natural photos has not been implemented yet, despite a lot of effort in research. Maybe, there is potential in using the recently popularised graph networks as projection fibres and thus learning the projections without relying on restricted mathematical models. For example, either VAE or VQ-VAE could be used as feature extractors, as in horizontal self-organisation. The advantage of these auto-encoder types is that they can extract local patterns and describe them as vectors in a "meaningful" latent space, i.e. similar vectors lie closer together in this latent space or, in the case of VQ-VAEs, are even discrete values. This facilitates the statistical learning and mapping of similar embedding vectors together. Horizontal self-organisation (c.f. Section 7.2.2) is considered an important preliminary work to implement such a system. In contrast to horizontal self-organisation, auto-encoders could be applied at any image position to obtain continuous pixel representations from the same latent space. The auto-encoders can thus be used as feature extractors to extract good representations of small image patches. Afterwards, graph neural networks (GNNs) could be used to combine the embedding vectors into higher-level features. GNNs have already been used for image classification in recent years, mainly by identifying super-pixels and connecting them by graphs to recognise object structures [270]. One problem with GNNs is that they are often flat, i.e. many nodes lie on the same plane. This does not comply to the model of projection fibres, which must be hierarchical due to the large number of combinations. A remedy to this issue could be using differential pooling [271, 272].

[270]: Long et al. (2021)

[271]: Ying et al. (2018)

[272]: Vasudevan et al. (2023)

However, the goal is not to identify objects with graph structures. Rather, the graph should be able to match objects from images with abstract prototypes. In this way, the network should learn to ignore slight transformations and rotations. Therefore, I suggest using feeding embeddings instead of super-pixels in to GNNs to obtain graph representations of images, as embeddings can be tuned to the use case and contain more information than super-pixels. Next, and much more importantly, GNNs for object recognition can be combined with graph matching networks [273, 274] and thus might be used to match objects locally and globally despite slight transformations. This matching would then correspond to projection fibers from L1 to L2 and compare objects within an image with mental prototypes. However, many questions remain, such as how objects are identified in images (i.e. how to remove the background) or how the mental prototypes get into L2 in order that they can be matched. Furthermore, one must think about an iterative matching to avoid early commitment.

[273]: Li et al. (2019)

[274]: Xu et al. (2022)

7.4 Useful Representations

In this thesis, models are proposed that extract representations of images. However, the question how useful this representations are still remains[3].

3: despite for the usual ML tasks such as image classifications

How useful a representation is often depends on the use case. For example, representations from modern deep learning systems are very useful for typical vision tasks such as classification or segmentation. In fact, it can even be argued that deep learning is often superior to humans. For example, modern deep learning systems [275] are able to identify millions of faces with more than 99% accuracy, which is very difficult for humans [4]. For such tasks, the representations of deep learning systems seem very well suited,

[275]: Jung et al. (2022)

4: at least to distinguish such a large number of people

However, what works poorly for deep learning systems is to recognise an object as the same instance, regardless of which transformations have been applied to the object. In general, it seems to be a problem that deep learning systems cannot learn a good world model and understand transformations applied to the objects of this model. One way to address this problem is to allow the model to interact with the world (i.e. perform actions). This allows it to learn how an action changes the view of an object. If the same actions are applied to different models, an object-independent transformation behaviour can be learned. This could allow a model to understand which views represent the same object in the world model and what kind of transformations have been applied to it.

It is known from the study of animals that both eye movements and the behavioural state influence the responses of neurons in the visual cortex [256]. Thus, animals integrate their action (i.e. the movement they are doing) with currently incoming sensory signals to predict future sensory inputs. The internal copy of an outflowing movement-producing signal generated by an organism's motor system is also known as efference copy. Keurti et al. [276] argue that such efference copies are useful to learn *useful* latent representations perceived by the visual system. They translated this idea into an AI-based system by allowing an agent to interact with the environment and to observe its state to build internal representations. In fact, the enforce that transformations of the real-world can also be applied on latent representations, i.e. that the representations of object and the real-world objects remain consistent when similar transformations are applied on them.

[256]: Keller et al. (2012)

[276]: Keurti et al. (2022)

Giving an agent an embodiment[5] to interact with the world to better understand it and to create better representations seems not only important from a neuroscientific perspective, but is also in line with theories from psychology. Piaget [277] argues that perceiving an object rather about understanding of how an object transforms and behaves under different interaction and not about creating a mental copy of it.

5: in this context, an embodiment can also be virtual, i.e. allow the agent to interact with objects

[277]: Piaget (1964)

Such an agent can be implemented, for example, with reinforcement learning. The training process could be explicitly modelled by predicting future states based on a given state and possible actions before the action is executed and the actual outcome is observed in the world model. This procedure corresponds to the perception-action episode that is also postulated by LeCun [278]. He divides the process in seven steps; (i) First, the perception system extracts a representation of the current state of the world $s[0] = P(x)$. (ii) The actor then proposes an initial sequence of actions $(a[0], ..., a[t], ...a[T])$ that is evaluated by the world model. (iii) The world model in turn predicts likely sequence of world state representations resulting from the proposed action sequence $(s[1], ..., s[t], ...s[T])$. (iv) A cost model estimates the total costs for each

[278]: LeCun (2022)

state sequence as a sum over time steps $F(x) = \sum_{t=1}^{T} C(s[t])$. (v) Based on the cost predictions, the actor proposes the action sequence with the lowest costs. (vi) The actor then executes one or a few actions (and not the entire action sequence) and the entire process is repeated. (vii) Additionally, every action, the states and associated costs are stored in a short-term memory that can be used to optimize the system.

7.5 Cognition and Reasoning

At the end of this thesis, I want to share my personal opinion on the future of deep learning systems. I am of the opinion that deep learning systems can extract pixel-representations from images very well. I intentionally call it "pixel"-representations and not object representations because these vectors contain information about pixel constellations which are not sufficient to describe objects but very well suited for tasks like classification or segmentation[6]. According to my definition, these pixel-representations are only a part of object representations. Object representations are, in my understanding, a mental construct that contains much more information about objects than how they look. For example, object representations should contain information about how an object behaves under different transformations. In the previous two sections (c.f. Section 7.3, Section 7.4). it is discussed how such transformations could be learned. However, there are many other parameters that make up an object, such as what abilities (e.g. can it move?) it has and how it feels. The appearance of an object (which corresponds to the pixel-representations) is thus only one dimension of a high-dimensional formula that describes an object. For example, fish and ships look completely different on a pixel-level description and have visually nothing in common. For us humans, these two objects have a clear relationship defined by their ability to swim and the typical place where they are found, namely the sea. However, many modern pattern extraction system only model the one-dimensional pixel-representations and not multi-dimensional object-representations and thus cannot build such relationships. For me personally, the extension of these one-dimensional object representations ("what does the object look like") to multi-dimensional representations is one of the key elements to be able to create a system with cognition and reasoning. This allows object hierarchies to be built on multiple dimensions: Instead of just a hierarchy of objects that look similar, we also need hierarchies of objects that behave similarly, have similar capabilities, feel similar, etc. In addition to this construction of multi-dimensional object representations, an understanding of the physics of the world is also necessary. This second type of representation summarises knowledge that relates to all objects. For example, the gravitational force of the earth, the fact that living beings cannot pass walk solid materials such as walls, etc.

6: end-to-end backpropagation of error seems to be well suited for such tasks

The question that arises now is how such a system can be implemented. It is obvious that simply showing pictures and the corresponding labels is far from sufficient to learn such complex relationships. At least three key elements are required:

The world in which the agent lives must allow interactions. This allows the agent to learn representations better, for example through

interactions, getting eference copies, and observations how the world behaves. This allows differences and similarities between objects to be identified, for example how two different objects behave when the same actions are applied to them. It also allows the agent to define what it does not yet know and to learn these things consciously (for example with an entropy-based loss function [257]). This also means that a continuous input is necessary and the model cannot be trained with a sequence of random and independent images like many current vision models.

[257]: Storck et al. (1995)

The network architecture must, to some extent, not only enable but encourage the capture of the world structure. Personally, I think that a multi-dimensional world model is helpful to store and relate the appearance, behaviour, transformation capabilities, abilities, etc. of each object as well as a second model to store general knowledge about the world.

The learning algorithm should take several things into account to learn a good world model. A key element seems to be that the learning algorithm decides itself what to learn (entropy-based on the knowledge in the world model). Furthermore, curriculum learning seems promising: for a high level of intelligence, an equally complex world must be available in order to be able to learn it. However, such worlds might be so complex that the agent is overwhelmed without a step-by-step complexation of the world[7].

7: this also applies to us humans: We have a prior knowledge through evolution, then our parents explain the world to us. In fact, we are not able to survive without proper support in the first years

Furthermore, I think we often underestimate the amount of information that such a system has to learn. It is often argued that today's models are huge and see far more data than a human. However, this does not take into account, for example, that people receive a boost through curriculum learning from many other people (e.g. parents, siblings, relatives, teachers, trainers, colleagues, etc.). Moreover, these people in turn draw on tens of thousands of generations of previous knowledge. A certain amount of this knowledge is also implicitly pre-programmed through the structure of the nervous system. According to Darwin's theory, this goes back to the first living creatures that lived billions of years ago. Why is this relevant? I see it as a very critical link to the design of the world in which an agent lives: we humans often define a task for agents such as landing a spaceship (LunarLander from Open AI [279]) and believe that the agent learns to control such a ship. We humans immediately realise that we have to activate the boost in the correct direction to balance the rocket and make it sink less quickly to the ground. But we already have this prior knowledge before we even play the game. For example, we know the physics of gravity and it seems logical to us that a spaceship sinks to the ground within the atmosphere, while mountains or stars do not. An AI agent, on the other hand, does not have this knowledge - this does not have to be bad by definition but can be seen as a knowledge gap that has to be learned. The question here is whether an understanding of this can ever be learned if the input is only a visual scene and the agent cannot experience this force itself. Or why should an agent ever understand that a spaceship looks like a spaceship and that it is being pulled towards the earth by gravity, and not just see a funny looking collection of pixels that happens to be moving downwards at a constant speed? In other words, we humans consider ourselves intelligent in the real 3D world we live in. But this intelligence is based on the fact that we

[279]: Brockman et al. (2016)

are part of a huge population that has lived in this highly complex world for generations. Personally, I think it is questionable whether an agent can learn knowledge comparable to that of humans by interacting in a minimally simple world-simulation.

Assuming such a world exists. Then there is still the question how the design of a proper learning system looks like. An intelligent system consists of several components: One very crucial component is the perception system. From today's point of view, this is probably the component that is best mastered[8]. Around this system, a world-model has to be created, which stores the understanding about the world. One question is how this world-model can be created. It seems to be important that the agent learns to learn, i.e. finds out for itself what knowledge it lacks in the world model and acquires it from the real world. As mentioned, such behaviour can be implemented by an entropy-based loss function [257]. This shifts the problem to the question of what knowledge is extracted from the world and stored in the world model (and thus implicitly defines what the agent learns). The knowledge stored in the world-model should be the multi-dimensional object information described in the first paragraph of this section, which provides a much richer understanding of the world than the current one-dimensional pixel representations. To get an actual understanding of the world, interactions with the world and prediction how the next state of the world look like seems to be very central[9]. A very simple implementation is a roll-out (i.e. predict how the world looks like after applying some actions and comparing this to the real world after these actions have been applied) [278], more advanced seem to be GFlowNetworks [280]. After such predictions are learned, I think that reversing this process as in diffusion models could help casual reasoning. In diffusion models, one learns to map from a specific state (a concrete image) to a very general state (noise). This process is reversed and over several steps, the model can build trajectories from noise to a sample through a combination of direct motion and random motion. If a similar concept is applied to the learned prediction of actions, then it can be predicted which action combinations have led to the current world state.

However, all these ideas are very abstract and have emerged in the course of this work. The concretisation of these ideas could be a thesis in itself and, unfortunately, the elaboration of such ideas is out-of-scope of this thesis. However, I hope to be able to address such questions as concrete research topics in the near future, ideally in a similar constellation as in this Master's thesis.

8: although there are undoubtedly things that need to be improved, c.f. Section 7.3

9: for example, the simple task of predicting the next token of a text has led to the enormous capabilities of large language models

[278]: LeCun (2022)

[280]: Bengio et al. (2021)

Appendix

Experiments Hebbian Learning A

I used different variants of Hebbian Learning with the goal of generating good latent representations of visual scenes. However, all preliminary experiments with different implementations were not promising. In my experiments, classical Hebbian learning (as described in Equation equation (**??**)) led to symmetric weights, which in turn led to poor representations. This symmetry could be broken by using the ABCD-Hebbian learning rule [281]. The ABCD learning rule calculates the weight update Δw_{ij} from neuron i to neuron j based on the pre-synaptic activity r_i of neuron i and post-synaptic activity r_j of neuron j as follows:

[281]: Niv et al. (2001)

$$\Delta w_{ij} = \eta_w \cdot (A_w r_i r_j + B_w r_i + C_w r_j + D_w) \tag{A.1}$$

where η_w is a weight-specific learning rate, and A_w is a correlation coefficient, B_w is a presynaptic coefficient, C_w is a postsynaptic coefficient, and D_w is a bias coefficient. The bias coefficient D_w can be interpreted as an individual inhibitory or excitatory bias of each connection in the network. Similar to Najarro and Risi [112], the coefficient were learnt through an evolution strategy [237]. However, this method does not scale for large networks with many parameters and did not achieve the desired performance in my experiments.

[112]: Najarro et al. (2020)

[237]: Salimans et al. (2017)

Of course, it cannot be concluded from these experiments that Hebbian Learning cannot be used to learn networks for extracting good representation from visual scenes. However, it was found that it is not trivial and that just applying the Hebbian learning rule is not sufficient, especially if the network has a structure of modern Deep Learning architectures with many parameters.

However, in further preliminary experiments it was found that Hebbian Learning has interesting properties for pruning. A good performing CNN was created and trained on MNIST with backpropagation. The network consists of two convolutional layers, followed by a pooling layer and three linear layers. The first convolutional layer increases the number of channels from [width × height × n_channels] to [width × height × 32]. The second convolutional layer further increases the number of channels to [width × height × 64]. The subsequent max. pooling layer decreases the size to [width/2 × height/2 × 64]. The activation map is then flatten and fed into 3 fully connected layers with an output size of 1024, 128, and 10 respectively. Between each layer, ReLU activtions are employed. The network is visualized in Figure Figure A.1.

The image classification model is trained with backpropagation to minimize the negative log likelihood (NLL) loss. Adam [260] is used as optimization algorithm with a mini-batch size of 32 a learning rate of $5 \cdot 10^{-4}$. As soon as the loss reaches a plateau, the learning rate is reduced to $1 \cdot 10^{-4}$.

[260]: Kingma et al. (2017)

After the model is trained, the first two fully connected layers within the network are pruned with Hebbian learning based methods. The

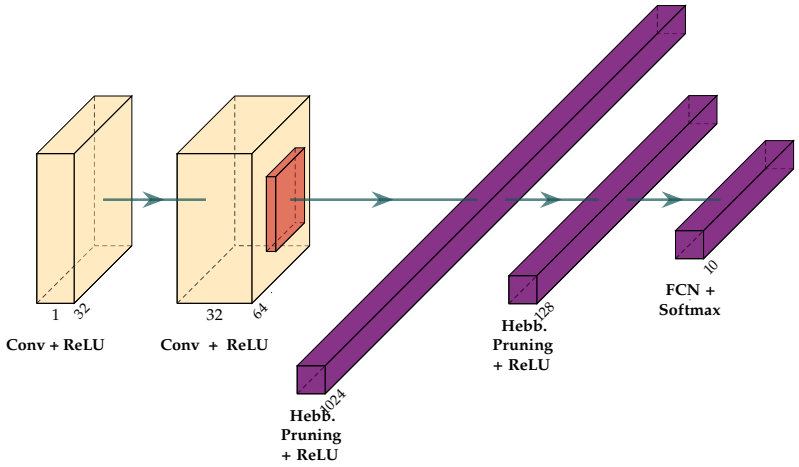

Figure A.1: The network architecture used to perform Hebbian-based weight pruning.

convolutional layers cannot be pruned because such layers employ a convolutional function that impedes calculating the correlation between an input and an output neuron. The last layer, on the other hand, cannot be pruned as it corresponds to the number of classes.

The *first* Hebbian based method proposed pushes the weights between neuron i and neuron j towards 0 if i and j have a low correlation within a mini-batch. First, a mini-batch is sampled. Afterwards, the correlation between i and j is calculated for each sample. if the correlation is below 0.02 for at least 20% of the samples (i.e. 7 samples or more for a mini-batch size of 32), the weight is updated as follows:

$$\Delta w_{ij} = -\eta_H \cdot w_{ij} \tag{A.2}$$

where η_H is the Hebbian learning rate set to $1 \cdot 10^{-5}$.

The *second* Hebbian based method proposed pushes some of the weights exactly to 0 while all other weights remain the same. During an epoch, the correlation between neurons i and j is measured for each weight w_{ij}. Afterwards, the weights with the lowest correlations are set to 0. How many weights are set to 0 is a hyper-parameter that has to be specified in advance.

For both methods, the model is pre-trained and pruned on the MNIST dataset [246] and evaluated on the MNIST as well as on the MNIST-C[1] dataset [261]. The *first* Hebbian based method can improve the results even after backpropagation as shown in Table Table **??**.

1: MNIST-C is a corrupted version of MNIST and well suited to measure model robustness

Table A.1: NLL loss of the model before and after the *first* Hebbian-based pruning method.

Dataset	NLL-Loss without Hebbian Updates	NLL-Loss with Hebbian Updates
MNIST	0.05314	0.03386
MNIST-C	0.6013	0.4766

Dataset	Accuracy without Hebbian Updates	Accuracy with Hebbian Updates
MNIST	98.94%	98.97%
MNIST-C	88.74%	89.94%

The NLL-loss on the MNIST test dataset decreases from 0.05314 to 0.03386, while the loss on MNIST-C test dataset decreases from 0.6013 to 0.4766 after applying the *first* Hebbian based method. However, in terms of accuracy the improvements are marginal. Furthermore, it has to be considered that this is only a preliminary experiment and this method certainly offers various potential for improvement. This experiments indicates that Hebbian in combination with back-propagation could lead to better or more robust representations because the Hebbian updates improved performance on both datasets.

The *second* Hebbian based method sets weights between neurons with a low correlation to 0. The network trained with back-propagation has an accuracy of 98.94% on MNIST. By setting up to 40% of the weights to 0, the accuracy remains above > 98%. Setting weights to 0 can make the network smaller and more efficient [282]. Furthermore, it leads to sparse representations that can improve robustness. Figure Figure A.2 visualizes the loss of the *second* Hebbian based method for different ratios of kept weights. Furthermore, the correlation based method is compared to randomly setting the same fraction of weights to 0.

[282]: Liang et al. (2021)

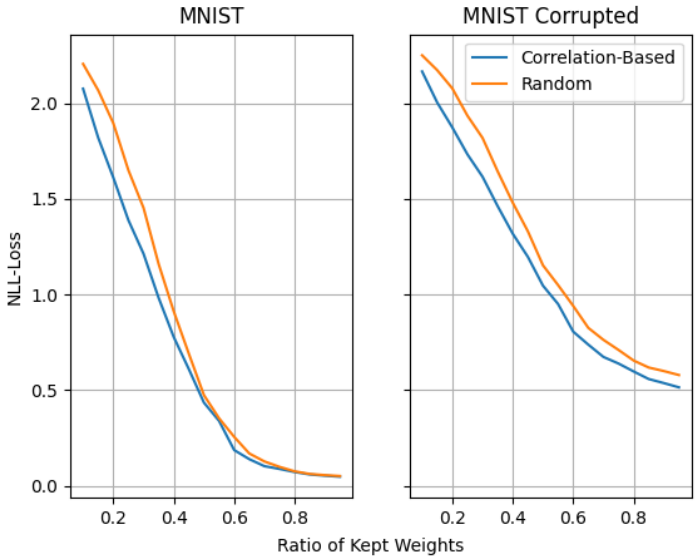

Figure A.2: The NLL loss of the *second* Hebbian based pruning method compared to removing random weights. The y axis shows the loss while the x axis shos the ratio of weights kept.

It can be observed that the performance is better when weights between neurons with low correlation are set to 0 than if random weights are set to 0.

Net-Fragments and Deep Learning

B

In this chapter, it is examined how representations of typical deep learning architectures look like. Especially net fragments are of interest, i.e. (groups of) neurons that represent certain features of objects. Net-fragments are discussed in Section 4.3. There, it is described that the interpretation of net-fragments in this thesis relies on two principles:

- The latent representations should be read out from multiple layers and not from a single layer
- To obtain meaningful activations, the activation maps should be sparse and diverse

The first principle is violated by most deep learning architectures as the latent representations are read out from a single layer and subsequently used for a downstream task. Since this first principle is violated by deep learning architectures by design, this chapter focuses mainly on the second principle. Specifically, it investigates whether specific neuron groups can be assigned to specific features (i.e. represent net-fragments). If this would be the case, specific features would be expected to trigger the activation of a group of neurons when they are present and that the same neurons would be inactive when this feature is not present. Thus, different neurons should be active for different objects. It is important to note that this mainly improves robustness and interpretability. Sparsity and diversity is not necessary, for example, to achieve high classification accuracy.

Deep Learning architectures usually consist of several layers with millions of neurons, leading to millions of activations [21–26]. Thus, the input data is processed in a complex way, which makes the analysis of activations difficult. To simplify the analysis of network activations, a novel straightforward classification dataset is proposed; The dataset consists of 10 images as shown in Figure B.1. Each image has a size of 9×9 pixels and depicts a number between 0 and 9. The images have only one channel and contain binary values (i.e. pixels are either set to 0 or 1). Small networks are sufficient to analyze these images and this dataset thus leads to less network activations what simplifies the search for net fragments.

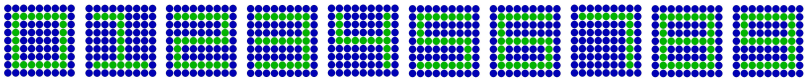

Figure B.1: A novel dataset created to investigate the properties of modern deep learning architectures. The dataset consists of images of the numbers $0-9$, each image has a size of 9×9 pixels. The blue dots represent pixels with the value 0, green dots represent pixels with the value 1.

Even in this small data set, there exist various features and consequently a multitude of network fragments. A very intuitive way for us humans to construct features on this data set is to interpret each line as a feature. For example, the dataset could be explained by 9 basic lines, that can be composed to 10 different digits. Thus, a low-level net fragment could

represent such a basic line and be composed to higher-level net fragments such as digits. Figure B.2 visualizes such an intuitive composition (basic lines on the left, digits on the right).

Thereby, each low-level fragment can be part of one or multiple digits. The composition can either take place across one or multiple layers. The visualization in Figure B.2 can be interpreted as a composition across one layer, i.e. the low-level fragments are combined to high-level fragments within one step.

The composition can also take place across multiple layers. An exemplary composition of net fragments of the number 9 across 4 subsequent layers is shown in Figure B.3. In this example, it is demonstrated that a fragment representing a digit can be composed of other fragments that also represent digits.

However, this explanatory example has to be understood as an intuitive way how humans would build net fragments. Actual net fragments do not necessarily have to be a composition of lines but can be a composition of multiple pixels without semantic. In fact, it can be expected that neural networks come up with their own features that look totally different than the example used above for explaining net fragments. Regardless of what these net fragments represent, the network should have certain characteristics; If a neuron or a group of neurons are part of a net fragment that represents a feature in the input, then these neuron(s) should be strongly active if the feature is present and not or only very weakly active if the feature is not present. This leads to a sparse activation map and net fragments should only be active for digits with specific characteristics. Thus, different neurons should be active depending on the digit.

Digits with common characteristics should have some overlapping network activities; For example, the 7 is contained in the 3, the 3 is contained in the 9 and the 9 is contained in the 8. Therefore, these digits should have some common network activations. The 1, on the other hand, has nothing in common with the 4, thus these digits should have no overlapping activations. In the following, networks are trained and their activations are examined for such properties.

Two types of models are trained on this dataset, namely a classification network (supervised) and autoencoder networks (unsupervised). The goal of the classification task is to predict the number (labels $0-9$) that is shown in the image (c.f. Section 5). This task is neither for humans nor deep learning architectures challenging. Usually, the last layer of an image classification architecture has exactly as many neurons as there are classes to predict. Each neuron corresponds to a class, and the neuron with the highest activity (i.e. the most active neuron) is eventually used to predict the image-level label. Such architectures thus have the design constraint that the last layer of neurons must represent distinct classes. Therefore, each neuron in the last layer can be interpreted as a representation of a class-object. Since the last layer of a classification network represents entire classes, such architectures seem to be well suited to investigate whether the preceding layers represent object features (i.e. net fragments) that are composed by the last layer.

A classification network relies on labels. An autoencoder, on the other hand, works unsupervised and reconstructs the input data from a lower

dimensional embedding space (c.f. section 3.4). Thus, the autoencoder generates representations that contain not only the class information but the entire image information. Consequently, the embedding activations of autoencoders are also examined in the embedding space. In addition, the autoencoder offers the possibility to constrain the embedding space so that net fragments are more strongly encouraged.

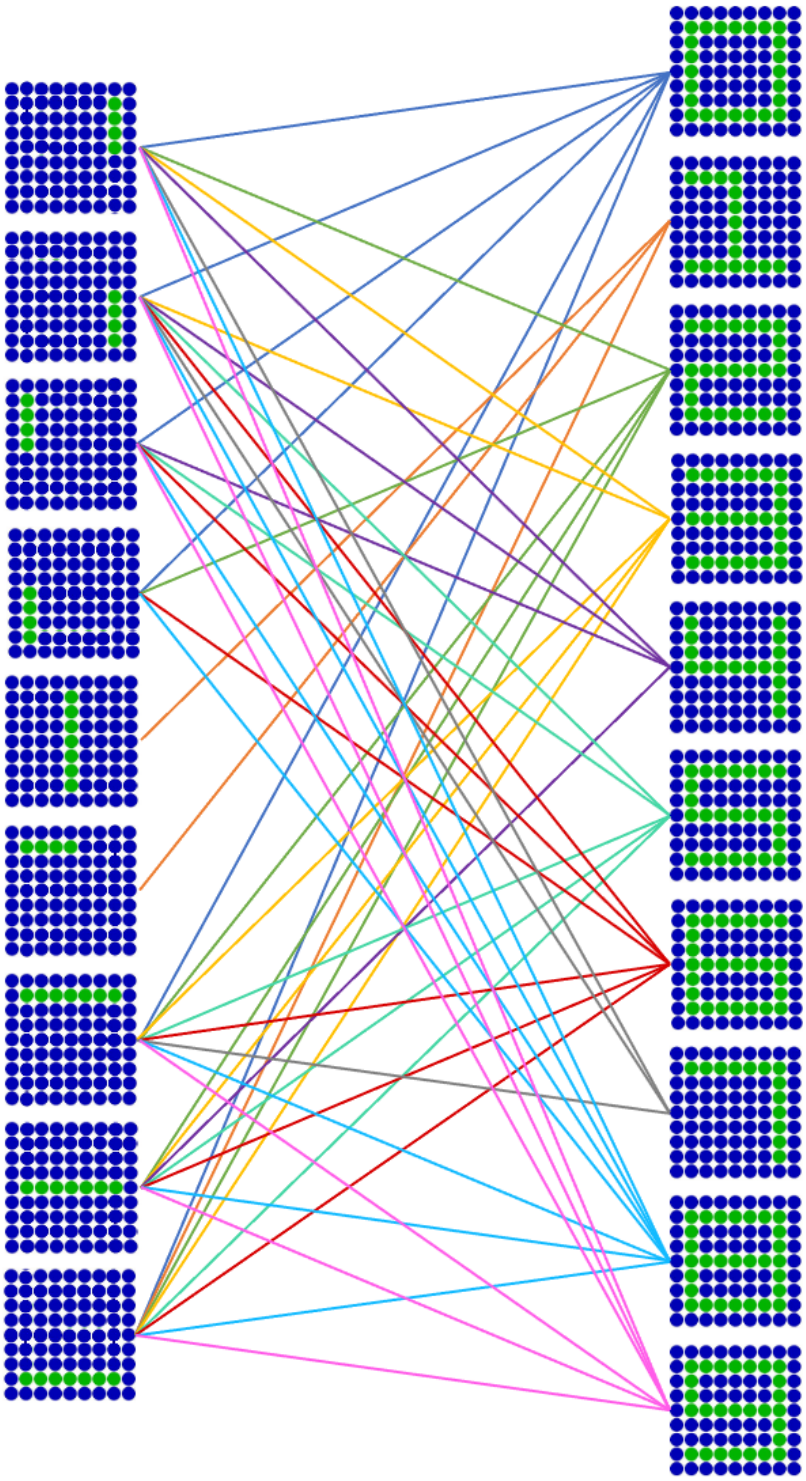

Figure B.2: A visualisation of all the lines (left) needed to compose the digits in the data set (right). The coloured lines in-between illustrate the relationship between the lines and the digits.

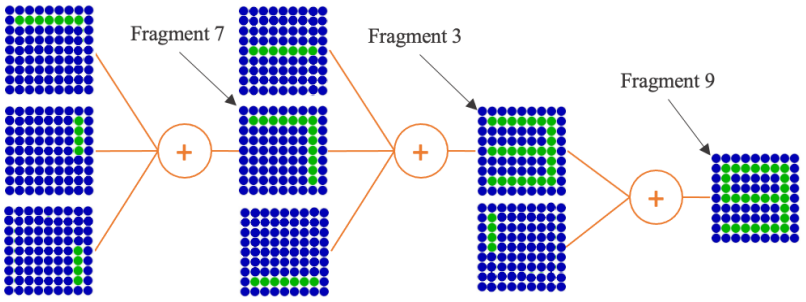

Figure B.3: A sample composition of the net fragment that could represent the digit 9.

B.1 Classification

B.1.1 Methods

To investigate the emergence of net fragments in classification networks, different architectures are trained. The model's parameters are optimised by minimising the cross-entropy loss with the Adam optimizer [10] so that the models learn to predict the corresponding class of the images shown in Figure B.1. The learning rate is $5 \cdot 10^{-4}$, the mini-batch size is 32 samples and the model is trained for a total of 10 epochs.

[10]: Kingma et al. (2015)

The models used have different feature extractors consisting of either convolutional (Conv.) layers (models no. 1-8, no. 11) or fully connected (FC) layers (models no. 10) as shown in Table **??**. Model no. 9 has no feature extractor as the input images are so simple that they can be considered as features by themself, model no. 10 uses a FC layer as feature extractor, and model no. 11 has a Conv. layer with two hand-crafted kernels to extract horizontal and vertical lines as feature extractor. After the feature extractor, all models have a similar "head" consisting of 2 fully connected layers. The feature extractor aims to extract certain features from the image, that are combined into higher-level net fragments in the first fully connected layer of the "head" and composed into predictions per class (i.e. net fragments corresponding to classes) in the last fully connected layer of the "head". The first FC layer of the "head" maps the input activations to 12 output features. The number of output features is determined empirically by training various architectures and examining the number of active neurons. It was found that there are always fewer than 12 neurons active and that this capacity is therefore sufficient. The last fully connected layer consists of 10 neurons since this corresponds to the number of classes to predict. The encoders of the models used are described in more detail in Table **??**. The "head" is identical for all models and consists of a sequence of the following layers; FC (out size=12) → ReLU → FC (out size=10) → Softmax.

Table B.1: A description of the different classification networks that are investigated for net-fragments.

No.	Encoder Description
1	Conv. Layer (kernel size=3 × 3, channels=2) → ReLU → head
2	Conv. Layer (kernel size=3 × 3, channels=4) → ReLU → head
3	Conv. Layer (kernel size=5 × 5, channels=2) → ReLU → head
4	Conv. Layer (kernel size=5 × 5, channels=4) → ReLU → head
5	Conv. Layer (kernel size=3 × 3, channels=2) → ReLU → Conv. Layer (kernel size=3 × 3, channels=4) → ReLU → head
6	Conv. Layer (kernel size=3 × 3, channels=4) → ReLU → Conv. Layer (kernel size=3 × 3, channels=8) → ReLU → head
7	Conv. Layer (kernel size=3 × 3, channels=2) → ReLU → Max Pooling → Conv. Layer (kernel size=3 × 3, channels=4) → ReLU → head
8	Conv. Layer (kernel size=3 × 3, channels=4) → ReLU → Max Pooling → Conv. Layer (kernel size=3 × 3, channels=8) → ReLU → head
9	head
10	FC (in size=9 ∗ 9, out size=12) → ReLU → head
11	Hand Crafted Conv. Layer for vertical & horizontal edge detection (kernel size=3 × 3, channels=2) → ReLU → head

After each epoch, the model's weights are stored as well as the activations of each layer. These vectors are visualized and investigated for net fragments.

Furthermore, the most relevant input features for a fully trained model are analysed. This is done by freezing the model's parameters so that they cannot change. Instead, an empty image is fed into the network and updated with backpropagation of error such that the probability for a given class is maximized. This leads to an input image that has the highest probability to be predicted by the model as a specific class (e.g. generate the image that has the highest probability to be predicted as digit 3 by the model).

B.1.2 Results

All the models learn to classify these digits perfectly within a few epochs. Interestingly, not only the accuracy reaches 100% but some models also achieve a cross-entropy loss of 0.0, meaning that they can find a global minimum. However, the goal is not to achieve high accuracy but to exhibit net fragments. It is not feasible to visualise all activations of all layers in this thesis. Therefore, only the activations of the second last FC layer (i.e. the first FC layer of the "head") after the ReLU function are shown. Since the last layer contains the net fragments that depict classes, this is the layer with the highest-level fragments. Furthermore, it is the only layer that has a global view on the input, i.e. can access all the features extracted by the encoder[1] (except for the encoder consisting of FC layers only). Some higher-level net fragments should be visible in this layer, which are subsequently composed in the last layer to net fragments corresponding to classes.

1: convolutional layers only consider a local neighbourhood by sliding a kernel over the input (c.f. Section 2.2.1)

Figure B.4 shows the activations of the first fully connected layer in the "head" of the different models. The FC layer has 12 neurons whose activation is indicated along the vertical axis (per model), and the horizontal axis depicts the activation of the same neuron for the classes 0-9. For example, the circle in the top left corner indicates the activation of the first neuron for class 0, the circle in the top right corner the activation of the first neuron for class 9, and the circle in the bottom left corner the activation of the last neuron for class 0. Blue circles show activations that are exactly 0, red circles are low activations > 0, and green circles represent strong activations[2].

2: activations < 0 do not exist because they are set to exactly 0 by the ReLU function

It can be seen in the visualization that none of the models utilises all 12 neurons and certain neurons are always inactive regardless of the class. Also, no obvious net fragments can be identified; Most neurons are always similarly strongly active regardless of the class. If net fragments are formed, however, the neurons of a fragment would have to be strongly active in the presence of certain features and weakly active otherwise. Such behaviour is not observable in these activations.

Furthermore, some digits are very similar and differ in only two pixels. Some examples are the digit pairs 5 and 6, 8 and 9, 0 and 8, or 3 and 9. However, when the activations of the corresponding classes are compared, it is obvious that almost always all activations change a little bit, and not one neuron is turned on or off depending on the presence of these two pixels.

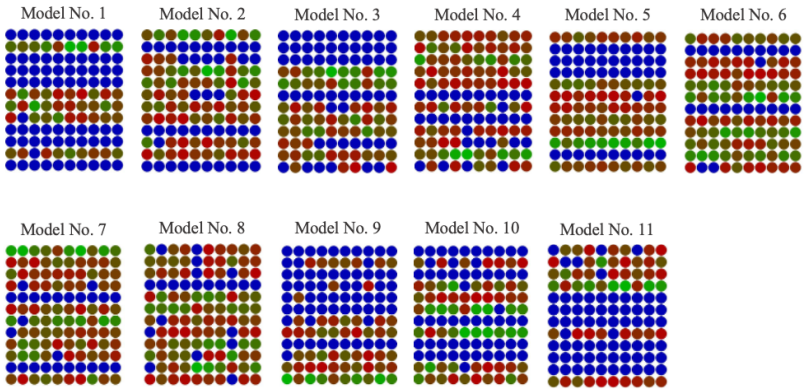

Figure B.4: The activations of the first fully connected layer in the "head" of the different networks. For each model, the activity of the 12 neurons is shown for each class (activations along the vertical axis, classes along the horizontal axis). Red means low activation, green means high activation, blue means activation is off (i.e. 0).

Figure B.5 visualizes the input that maximizes the probability for each class. It is obvious that all models focus on the wrong features and it can be assumed that these networks would not be robust to slight perturbations in the input. This also indicates that net fragments are not present in current deep learning models (or at least not to the desirable extent); if a class-level net fragment is composed of several lower-level net fragments, then some of these lower-level net fragments have to be active. Since these lower-level net fragments represent specific input features, corresponding pixel constellations should be visible in this visualisation. However, since these rather random-looking pixel combinations maximise the probability of predicting a specific class, this suggests that pixel combinations are not composed into hierarchically more complex net fragments.

Figure B.5: The inputs that maximize the probability that the different models predict a specific class. For example, the image in the top left corner maximizes the probability that model number 1 predicts that this is class 0 (digit 0).

B.2 Autoencoders

B.2.1 Methods

Similar to the experiments with classification networks, different autoencoders are also investigated. Since most architectures lead to similar results for the classification networks, the focus is less on testing many different architectures but rather on incorporating different constraints that might encourage net-fragments. Incorporating constraints into an autoencoder can be done easily by adding a loss function that regulate the activations in the bottleneck layer. For a classification network, on the other hand, it is not obvious how such constraints should be added. The two different autoencoder architectures used are shown in Table **??**.

Table B.2: A description of the different autoencoder networks that are investigated for net-fragments.

No.	Encoder Description
1	FC (in size=$9*9$ out size=12) $\rightarrow$ ReLU $\rightarrow$ FC (in size=12 out size=$9*9$)
2	FC (in size=$9*9$ out size=40) $\rightarrow$ ReLU $\rightarrow$ FC (in size=40 out size=$9*9$)

The autoencoder's parameters are optimised by minimising a reconstruction loss (i.e. a distance measurement between the input and the reconstructed output) with the Adam optimizer [10] so that the output looks similar to the input even tough the networks capacity is reduced by a bottleneck-layer in the middle. The learning rate is $5 \cdot 10^{-4}$, the mini-batch size is 32 samples and the model is trained for a total of 10 epochs.

[10]: Kingma et al. (2015)

The reconstruction loss L_{rec} used for the different experiments is either the L1 distance (mean absolute error) or the L2 distance (mean square

error). If $\boldsymbol{x}$ is the input data with N pixels and $\hat{\boldsymbol{x}}$ is the reconstructed data, then the reconstruction loss can be written as follows:

$$L_{rec}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} |x_n - \hat{x}_n|, & \text{if use L1 distance} \\ \frac{1}{N}\sum_{n=1}^{N} (x_n - \hat{x}_n)^2, & \text{if use L2 distance} \end{cases} \tag{B.1}$$

Optionally, a sparsity and a diversity loss were added to the reconstruction loss to encourage net-fragments. Since specific neurons of a net-fragment represent certain features, they should only be active if the feature is present. Good features only occur as part of a few objects and not in all of them: Thus, the neurons representing this feature should be active only sporadically and therefore the activations should be sparse and diverse. This combination of sparsity and diversity is also in line with the ideas of LeCun to obtain autonomous machine intelligence [278].

[278]: LeCun (2022)

For the sparsity loss L_s the kullback-leibler (KL) divergence is used [179]. The hidden representations $\boldsymbol{z}$ in the bottleneck layer with length m (i.e. m neurons in the bottleneck layer) are defined as $\boldsymbol{z} = z_1, ..., z_m$. Furthermore, the average activation probability of a neuron z_j over the input data $\boldsymbol{x}$ can be calculated as

[179]: Le et al. (2012)

$$\hat{\rho}_j = \frac{1}{N}\sum_{i}^{N} \frac{1}{1 + e^{-z_i^{(l)}}} \tag{B.2}$$

where $\frac{1}{1+e^{-z^{(i)}}}$ is the sigmoid function that squeezes the activation in the range between 0 and 1. Furthermore, $\rho = 0.05$ is sparsity parameter that can be interpreted as a target activation probability of each neuron. If the kullback-leibler divergence between all $\hat{\rho}_j$ and ρ is minimised, then an average activation probability per neuron in the hidden layer of ρ is enforced. The corresponding kullback-leibler divergence for each $\hat{\rho}_j$ is defined as:

$$KL(\rho||\hat{\rho}_j) = \rho \cdot \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \cdot \log \frac{1-\rho}{1-\hat{\rho}_j} \tag{B.3}$$

The sparsity loss L_s is the sum of the kullback-leibler divergence between all $\hat{\rho}_j$ and ρ and thus enforces that on average only 5% of the hidden representations $\boldsymbol{z}$ are active:

$$L_s(\rho, \hat{\rho}) = \sum_{j=1}^{m} KL(\rho||\hat{\rho}_j) \tag{B.4}$$

Thus, this constraint ensures that the activations are sparse, which is consistent with the concept of net-fragments. Another property of net-fragments is that the activations should be diverse. Therefore, a new diversity loss L_d is proposed. This loss is based on the fact that the activations in the bottleneck layers $(\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(N)})$ should be as different as possible for all samples of a mini-batch $(\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)})$. It is easy to see that the sum of the dot product between the activation of a sample $\boldsymbol{z}^{(i)}$ and all other samples $\boldsymbol{z}^{(j)}$, $i \neq j$ is small if the samples within a batch have a high diversity. Therefore, the diversity loss can be defined as:

$$L_d(\boldsymbol{z}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} \boldsymbol{z}^{(i)} \cdot \boldsymbol{z}^{(j)}, \text{ if } i \neq j \tag{B.5}$$

Minimising L_d results in the bottleneck layer activations not only being sparse but also diverse. The overall loss can be defined as the sum of these three loss components, with λ_s and λ_d beeing hyperparameters that control the influence of the diversity and sparsity loss.

$$L = L_{rec} + \lambda_s \cdot L_s + \lambda_d \cdot L_d \tag{B.6}$$

In the experiments, these hyperparameters were set to $\lambda_s = 0.02$ and $\lambda_d = 0.02$, respectively $\lambda_s = 0$ and $\lambda_d = 0$ if the sparsity and diversity loss should not be used.

Similar to the analysis of classification architectures, the model's activations of each layer are stored after every epoch. These vectors are visualized and investigated for net fragments.

B.2.2 Results

The activations of the autoencoders trained with reconstruction loss only and without diversity or sparsity constraint (i.e. $\lambda_s = 0$ and $\lambda_d = 0$) look very similar to those of the classification network; about half of the neurons are always active regardless of the object depicted in the image and the other half of neurons are always inactive. Thus, this version of the autoencoder has the same problem as the classification networks. Adding a sparsity constraint (i.e. $\lambda_s = 0.02$ and $\lambda_d = 0$) improves this issue slightly. The activations become sparse, but the same neurons are always active independent of the objects in the image. Only in a few cases can a set of neurons be identified that represent specific objects or features.

However, if all three loss components are used, the activations look much better. In this case, specific neuron combinations represent object-dependent features and are only active for specific objects. There are no more neurons that are constantly active. It is even possible to infer image labels based on a binary activity state (i.e. which neuron is active or inactive) and without knowing the strength of the activation, even though labels were not used during training. Thus, an autoencoder with additional sparsity and diversity constraint can represent net-fragments to some extent[3].

3: except that constraint 1 which says that net-fragments span several layers by design is violated (c.f. Introduction in Section **??**)

This can be observed in Figure B.6 that shows the activations of the "model 1". On the left are the activations without constraint, in the middle the activations with sparsity constraint and on the right the activations with sparsity and diversity constraint. For each of these three loss functions, the activations in the bottleneck layer are shown per class (activations along the vertical axis, classes along the horizontal axis). It is clearly visible how adding the sparsity constraint and the diversity constraint gives more meaning to the individual neurons. While without both constraints the same neurons are always active independent of the class, the activations are significantly more varied when both constraints

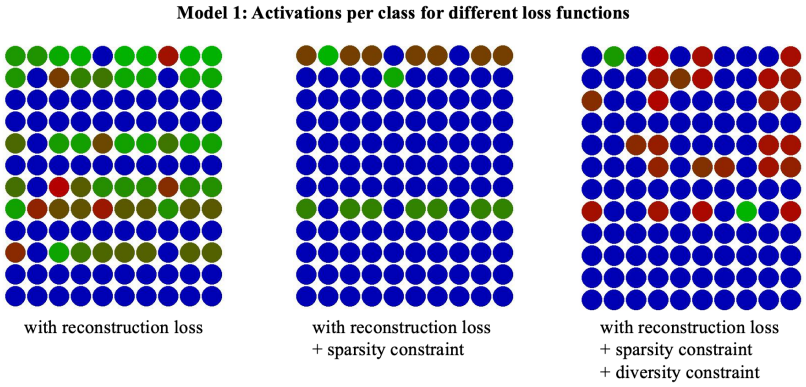

Figure B.6: The activations in the bottleneck layer of the smaller autoencoder "model 1" for different loss functions. For each loss function, the activity of the 12 neurons in the bottleneck layer is shown for each class (activations along the vertical axis, classes along the horizontal axis). Red means low activation, green means high activation, blue means activation is off (i.e. 0).

are used. This becomes even more obvious when the activations of the bigger autoencoder "model 2" are investigated (c.f. Figure B.7 (a)).

It is also examined what features the various neurons in the bottleneck layer represent. This is done by manually creating a bottleneck activation $\boldsymbol{z} = z_1, \ldots, z_m$ and feeding it through the decoder. To examine which feature the i-th neuron represents, $z_i = 1$ is set and $z_j = 0$, for $i \neq j$. Figure B.7 (b) shows which feature the 18-th neuron represents. Together with the 33-th neuron, this neuron represents the number 5 and together with the 36-th neuron the number 9. Thus, it is a feature (i.e. a net-fragment) that assembles with other features to represent an object.

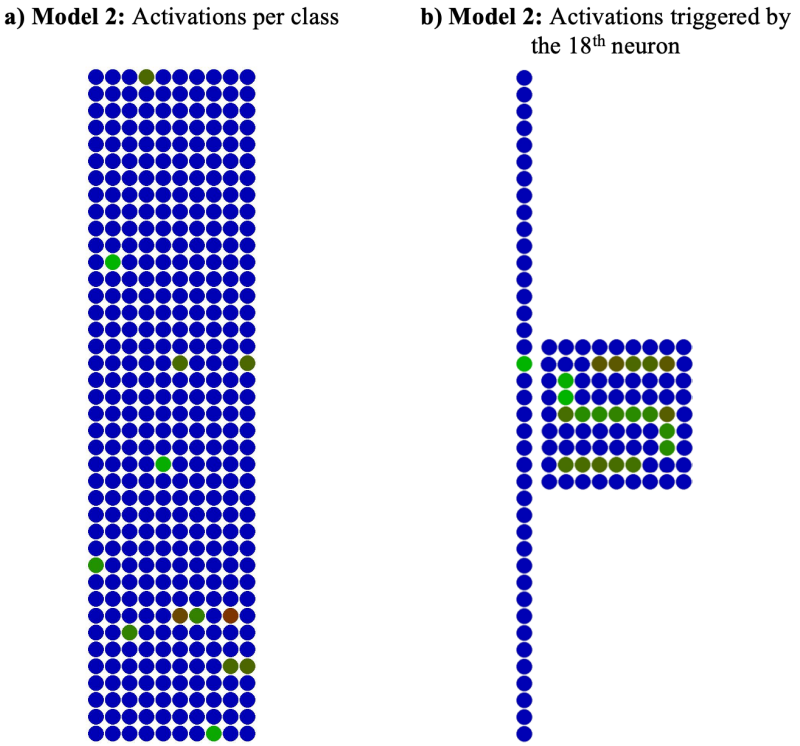

Figure B.7: On the lieft sinde (a), the activations in the bottleneck layer of the bigger autoencoder "model 2" are shown. The activity of the 40 neurons in the bottleneck layer is shown for each class (activations along the vertical axis, classes along the horizontal axis). On the right side (b), the output of the decoder is shown when the 18th neuron is set to 1 and all other neurons are set to 0. The 18-th neurons is a feature that is needed to generate the numbers 5 and 9.

B.3 Conclusion

Different architectures have been trained with different targets on a very simple and therefore easily interpretable data set. The network activations are tracked during training and analyzed for net-fragments. Typical vision architectures are sequential and build up a composition of features (i.e. net-fragments) over several layer. Usually, the embeddings of a pre-defined layer are used for down-stream tasks such as classification. Autoencoders typically use the bottleneck layer, a classification networks typically have a classification head (i.e. a FC layer with a Softmax activation) after the last model layer and thus extract these embeddings from the last model layer. Thus, embeddings are extracted from one layer. Net-fragments, on the other hand, span over multiple layers[4] and hence this principle is violated.

4: net-fragments can be thought of as the "path" of features through a network

Furthermore, net fragments are strongly active when the corresponding feature they represent is present in the input and are weakly active or not active at all when it is not present. Typical deep learning networks have neurons that are never active, while the active neurons usually remain active regardless of the class of the input data and only change their activity slightly. Thus, the information about different features is not distributed on different neurons but transported as activation strength of neurons through the network. Therefore, deep learning architectures do not comprise brain-like net fragments by default.

However, it was found that adding a sparsity and diversity constraint can alleviate this issue. Adding such constraints to the loss function encourages neurons to represent specific features that are typical for some of the objects.

Bibliography

Here is the list of references in citation order.

[1] Axios Media Inc. *Artificial intelligence pioneer says we need to start over*. 2017. URL: https://www.axios.com/2017/12/15/artificial-intelligence-pioneer-says-we-need-to-start-over-1513305524 (visited on 09/04/2022) (cited on page 1).

[2] Christoph von der Malsburg, Thilo Stadelmann, and Benjamin F. Grewe. 'A Theory of Natural Intelligence'. In: arXiv:2205.00002 (Apr. 2022). arXiv:2205.00002 [cs, q-bio] (cited on pages 2, 21, 22, 26, 30, 36, 38, 40, 74).

[3] Wikipedia. *Neuron*. 2023. URL: https://en.wikipedia.org/wiki/Neuron (visited on 02/19/2023) (cited on page 5).

[4] Hiroshi Takagi. 'Roles of ion channels in EPSP integration at neuronal dendrites'. en. In: *Neuroscience Research* 37.3 (July 2000), pp. 167–171. DOI: 10.1016/S0168-0102(00)00120-6 (cited on page 6).

[5] J. S. Coombs, J. C. Eccles, and P. Fatt. 'The specific ionic conductances and the ionic movements across the motoneuronal membrane that produce the inhibitory post-synaptic potential'. en. In: *The Journal of Physiology* 130.2 (Nov. 1955), pp. 326–373. DOI: 10.1113/jphysiol.1955.sp005412 (cited on page 6).

[6] Moheb Costandi. *Neuroplasticity*. The MIT Press essential knowledge series. Cambridge, MA: The MIT Press, 2016 (cited on page 6).

[7] Warren S. McCulloch and Walter Pitts. 'A logical calculus of the ideas immanent in nervous activity'. en. In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. DOI: 10.1007/BF02478259 (cited on page 6).

[8] F. Rosenblatt. 'The perceptron: A probabilistic model for information storage and organization in the brain.' en. In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519 (cited on page 6).

[9] G. Cybenko. 'Approximation by superpositions of a sigmoidal function'. en. In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: 10.1007/BF02551274 (cited on page 8).

[10] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *CoRR* abs/1412.6980 (2015) (cited on pages 8, 91, 94).

[11] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 'Learning representations by back-propagating errors'. en. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0 (cited on page 9).

[12] Coursera Inc. *Deep Learning Specialization*. 2022. URL: https://www.coursera.org/specializations/deep-learning (visited on 08/19/2022) (cited on page 9).

[13] Alexey Dosovitskiy et al. 'An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale'. In: arXiv:2010.11929 (June 2021). arXiv:2010.11929 [cs] (cited on pages 9, 35, 38, 55).

[14] Ilya O Tolstikhin et al. 'MLP-Mixer: An All-MLP Architecture for Vision'. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 24261–24272 (cited on page 9).

[15] Coenraad Mouton, Johannes C. Myburgh, and Marelie H. Davel. 'Stride and Translation Invariance in CNNs'. In: vol. 1342. arXiv:2103.10097 [cs]. 2020, pp. 267–281. DOI: 10.1007/978-3-030-66151-9_17 (cited on page 9).

[16] Wei Zhang et al. 'Parallel distributed processing model with local space-invariant interconnections and its optical architecture'. en. In: *Applied Optics* 29.32 (Nov. 1990), p. 4790. DOI: 10.1364/AO.29.004790 (cited on page 9).

[17] Sergey Ioffe and Christian Szegedy. 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: arXiv:1502.03167 (Mar. 2015). arXiv:1502.03167 [cs] (cited on pages 10, 27).

[18] Y. Lecun et al. 'Gradient-based learning applied to document recognition'. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. DOI: 10.1109/5.726791 (cited on pages 10, 22, 23, 26, 30, 50, 54, 58, 65).

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012 (cited on page 10).

[20] Karen Simonyan and Andrew Zisserman. 'Very Deep Convolutional Networks for Large-Scale Image Recognition'. In: arXiv:1409.1556 (Apr. 2015). arXiv:1409.1556 [cs] (cited on page 10).

[21] Christian Szegedy et al. 'Going Deeper with Convolutions'. In: arXiv:1409.4842 (Sept. 2014). arXiv:1409.4842 [cs] (cited on pages 10, 87).

[22] Kaiming He et al. 'Deep Residual Learning for Image Recognition'. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cited on pages 10, 87).

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 'U-Net: Convolutional Networks for Biomedical Image Segmentation'. In: arXiv:1505.04597 (May 2015). arXiv:1505.04597 [cs] (cited on pages 10, 11, 87).

[24] Kaiming He et al. 'Mask R-CNN'. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, Oct. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322 (cited on pages 10, 87).

[25] Wei Liu et al. 'SSD: Single Shot MultiBox Detector'. en. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Vol. 9905. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2 (cited on pages 10, 87).

[26] Joseph Redmon et al. 'You Only Look Once: Unified, Real-Time Object Detection'. In: arXiv:1506.02640 (May 2016). arXiv:1506.02640 [cs] (cited on pages 10, 87).

[27] Venture Beat. *New deep learning model brings image segmentation to edge devices*. 2023. URL: https://venturebeat.com/ai/new-deep-learning-model-brings-image-segmentation-to-edge-devices/ (visited on 02/19/2023) (cited on page 11).

[28] Huikai Wu et al. 'FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation'. In: arXiv:1903.11816 (Mar. 2019). arXiv:1903.11816 [cs] (cited on page 11).

[29] Niclas Simmler et al. 'A Survey of Un-, Weakly-, and Semi-Supervised Learning Methods for Noisy, Missing and Partial Labels in Industrial Vision Applications'. In: *2021 8th Swiss Conference on Data Science (SDS)*. Lucerne, Switzerland: IEEE, June 2021, pp. 26–31. DOI: 10.1109/SDS51136.2021.00012 (cited on page 11).

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985 (cited on pages 11, 31).

[31] Gordon E. Moore. 'Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.' In: *IEEE Solid-State Circuits Society Newsletter* 11.3 (Sept. 2006), pp. 33–35. DOI: 10.1109/N-SSC.2006.4785860 (cited on page 11).

[32] Suhas Kumar. 'Fundamental Limits to Moore's Law'. In: arXiv:1511.05956 (Nov. 2015). arXiv:1511.05956 [cond-mat] (cited on page 11).

[33] Open AI. *AI and Compute*. 2018. URL: https://openai.com/blog/ai-and-compute/ (visited on 08/19/2022) (cited on page 11).

[34] Matthew Peters et al. 'Deep Contextualized Word Representations'. en. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202 (cited on page 11).

[35] Tom Brown et al. 'Language Models are Few-Shot Learners'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901 (cited on page 11).

[36] Lambda Inc. *OpenAI's GPT-3 Language Model: A Technical Overview*. 2021. URL: https://lambdalabs.com/blog/demystifying-gpt-3/ (visited on 08/19/2022) (cited on page 11).

[37] Shaden Smith et al. 'Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model'. In: arXiv:2201.11990 (Feb. 2022). arXiv:2201.11990 [cs] (cited on page 11).

[38] Hao Wu et al. 'Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation'. In: arXiv:2004.09602 (Apr. 2020). arXiv:2004.09602 [cs, stat] (cited on page 11).

[39] Tejalal Choudhary et al. 'A comprehensive survey on model compression and acceleration'. en. In: *Artificial Intelligence Review* 53.7 (Oct. 2020), pp. 5113–5155. DOI: 10.1007/s10462-020-09816-7 (cited on page 11).

[40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 'Distilling the Knowledge in a Neural Network'. In: arXiv:1503.02531 (Mar. 2015). arXiv:1503.02531 [cs, stat] (cited on page 11).

[41] Yu Zhang and Qiang Yang. 'A Survey on Multi-Task Learning'. In: arXiv:1707.08114 (Mar. 2021). arXiv:1707.08114 [cs] (cited on page 12).

[42] Doyen Sahoo et al. 'Online Deep Learning: Learning Deep Neural Networks on the Fly'. In: arXiv:1711.03705 (Nov. 2017). arXiv:1711.03705 [cs] (cited on page 12).

[43] German I. Parisi et al. 'Continual lifelong learning with neural networks: A review'. en. In: *Neural Networks* 113 (May 2019), pp. 54–71. DOI: 10.1016/j.neunet.2019.01.012 (cited on page 12).

[44] Spandan Madan et al. 'When and how convolutional neural networks generalize to out-of-distribution category–viewpoint combinations'. en. In: *Nature Machine Intelligence* 4.2 (Feb. 2022), pp. 146–153. DOI: 10.1038/s42256-021-00437-5 (cited on page 12).

[45] Gary Marcus. 'Deep Learning: A Critical Appraisal'. In: arXiv:1801.00631 (Jan. 2018). arXiv:1801.00631 [cs, stat] (cited on page 12).

[46] Hans Moravec. *Mind children: the future of robot and human intelligence*. eng. 4. print. Cambridge: Harvard Univ. Press, 1995 (cited on page 12).

[47] D. J. Felleman and D. C. Van Essen. 'Distributed Hierarchical Processing in the Primate Cerebral Cortex'. en. In: *Cerebral Cortex* 1.1 (Jan. 1991), pp. 1–47. DOI: 10.1093/cercor/1.1.1 (cited on page 13).

[48] D. O. Hebb. *The organization of behavior; a neuropsychological theory.* The organization of behavior; a neuropsychological theory. Oxford, England: Wiley, 1949, pp. xix, 335 (cited on page 14).

[49] Erkki Oja. 'Simplified neuron model as a principal component analyzer'. en. In: *Journal of Mathematical Biology* 15.3 (Nov. 1982), pp. 267–273. DOI: 10.1007/BF00275687 (cited on page 15).

[50] El Bienenstock, Ln Cooper, and Pw Munro. 'Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex'. en. In: *The Journal of Neuroscience* 2.1 (Jan. 1982), pp. 32–48. DOI: 10.1523/JNEUROSCI.02-01-00032.1982 (cited on page 15).

[51] Nathan Intrator and Leon N Cooper. 'Objective function formulation of the BCM theory of visual cortical plasticity: Statistical connections, stability conditions'. en. In: *Neural Networks* 5.1 (Jan. 1992), pp. 3–17. DOI: 10.1016/S0893-6080(05)80003-6 (cited on page 15).

[52] Eero P Simoncelli and Bruno A Olshausen. 'Natural Image Statistics and Neural Representation'. en. In: *Annual Review of Neuroscience* 24.1 (Mar. 2001), pp. 1193–1216. DOI: 10.1146/annurev.neuro.24.1.1193 (cited on page 15).

[53] T. P. Vogels et al. 'Inhibitory Plasticity Balances Excitation and Inhibition in Sensory Pathways and Memory Networks'. en. In: *Science* 334.6062 (Dec. 2011), pp. 1569–1573. DOI: 10.1126/science.1211095 (cited on page 15).

[54] Prashant Joshi and Jochen Triesch. 'Rules for information maximization in spiking neurons using intrinsic plasticity'. In: *2009 International Joint Conference on Neural Networks*. 2009, pp. 1456–1461. DOI: 10.1109/IJCNN.2009.5178625 (cited on page 15).

[55] Michael Teichmann and Fred Hamker. 'Intrinsic plasticity: A simple mechanism to stabilize Hebbian learning in multilayer neural networks.' In: Mar. 2015 (cited on page 15).

[56] Stephen Grossberg. 'Nonlinear neural networks: Principles, mechanisms, and architectures'. en. In: *Neural Networks* 1.1 (Jan. 1988), pp. 17–61. DOI: 10.1016/0893-6080(88)90021-4 (cited on page 16).

[57] J J Hopfield. 'Neural networks and physical systems with emergent collective computational abilities.' en. In: *Proceedings of the National Academy of Sciences* 79.8 (Apr. 1982), pp. 2554–2558. DOI: 10.1073/pnas.79.8.2554 (cited on pages 16, 17).

[58] Evelyn Fix and J. L. Hodges. 'Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties'. In: *International Statistical Review / Revue Internationale de Statistique* 57.3 (Dec. 1989), p. 238. DOI: 10.2307/1403797 (cited on page 16).

[59] Jason Weston, Sumit Chopra, and Antoine Bordes. 'Memory Networks'. In: arXiv:1410.3916 (Nov. 2015). arXiv:1410.3916 [cs, stat] (cited on page 16).

[60] R. McEliece et al. 'The capacity of the Hopfield associative memory'. In: *IEEE Transactions on Information Theory* 33.4 (1987), pp. 461–482. DOI: 10.1109/TIT.1987.1057328 (cited on page 17).

[61] J. J. Hopfield, D. I. Feinstein, and R. G. Palmer. ''Unlearning' has a stabilizing effect in collective memories'. In: *Nature* 304.5922 (July 1983), pp. 158–159. DOI: 10.1038/304158a0 (cited on page 17).

[62] Dmitry Krotov and John J. Hopfield. 'Dense Associative Memory for Pattern Recognition'. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 1180–1188 (cited on page 17).

[63] Mete Demircigil et al. 'On a Model of Associative Memory with Huge Storage Capacity'. en. In: *Journal of Statistical Physics* 168.2 (July 2017), pp. 288–299. DOI: 10.1007/s10955-017-1806-y (cited on page 17).

[64] Hubert Ramsauer et al. 'Hopfield Networks is All You Need'. In: arXiv:2008.02217 (Apr. 2021). arXiv:2008.02217 [cs, stat] (cited on page 17).

[65] Volodymyr Mnih et al. 'Recurrent Models of Visual Attention'. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2204–2212 (cited on page 18).

[66] Sepp Hochreiter and Jürgen Schmidhuber. 'Long Short-Term Memory'. en. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cited on page 18).

[67] Corinna Cortes and Vladimir Vapnik. 'Support-vector networks'. en. In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. DOI: 10.1007/BF00994018 (cited on page 18).

[68] T. Cover and P. Hart. 'Nearest neighbor pattern classification'. In: *IEEE Transactions on Information Theory* 13.1 (Jan. 1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964 (cited on page 18).

[69] L. F. Abbott. 'Lapicque's introduction of the integrate-and-fire model neuron (1907)'. In: *Brain Research Bulletin* 50 (1999), pp. 303–304 (cited on page 18).

[70] E.M. Izhikevich. 'Simple model of spiking neurons'. en. In: *IEEE Transactions on Neural Networks* 14.6 (Nov. 2003), pp. 1569–1572. DOI: 10.1109/TNN.2003.820440 (cited on pages 18, 26).

[71] Romain Brette and Wulfram Gerstner. 'Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity'. en. In: *Journal of Neurophysiology* 94.5 (Nov. 2005), pp. 3637–3642. DOI: 10.1152/jn.00686.2005 (cited on page 18).

[72] Hélène Paugam-Moisy. 'Spiking Neuron Networks A survey'. In: (2006) (cited on page 18).

[73] Guo-qiang Bi and Mu-ming Poo. 'Synaptic Modification by Correlated Activity: Hebb's Postulate Revisited'. en. In: *Annual Review of Neuroscience* 24.1 (Mar. 2001), pp. 139–166. DOI: 10.1146/annurev.neuro.24.1.139 (cited on page 18).

[74] Saeed Reza Kheradpisheh et al. 'STDP-based spiking deep convolutional neural networks for object recognition'. In: *Neural Networks* 99 (2018), pp. 56–67. DOI: https://doi.org/10.1016/j.neunet.2017.12.005 (cited on pages 18, 26).

[75] Herbert Jaeger. 'The" echo state" approach to analysing and training recurrent neural networks-with an erratum note''. In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148 (Jan. 2001) (cited on page 19).

[76] Wolfgang Maass, Thomas Natschläger, and Henry Markram. 'Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations'. en. In: *Neural Computation* 14.11 (Nov. 2002), pp. 2531–2560. DOI: 10.1162/089976602760407955 (cited on page 19).

[77] Zoran Konkoli. 'Reservoir Computing'. en. In: *Unconventional Computing*. Ed. by Andrew Adamatzky. New York, NY: Springer US, 2018, pp. 619–629. DOI: 10.1007/978-1-4939-6883-1_683 (cited on page 19).

[78] Gouhei Tanaka et al. 'Recent advances in physical reservoir computing: A review'. en. In: *Neural Networks* 115 (July 2019), pp. 100–123. DOI: 10.1016/j.neunet.2019.03.005 (cited on page 19).

[79] P. Erdös and A. Rényi. 'On Random Graphs I'. In: *Publicationes Mathematicae Debrecen* 6 (1959), p. 290 (cited on page 19).

[80] Mantas Lukoševičius. 'A Practical Guide to Applying Echo State Networks'. en. In: *Neural Networks: Tricks of the Trade*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Vol. 7700. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 659–686. DOI: 10.1007/978-3-642-35289-8_36 (cited on page 19).

[81] John D. McPherson et al. 'A physical map of the human genome'. In: *Nature* 409.6822 (Feb. 2001), pp. 934–941. DOI: 10.1038/35057157 (cited on page 21).

[82] A.N. Kolmogorov. 'On tables of random numbers'. en. In: *Theoretical Computer Science* 207.2 (Nov. 1998), pp. 387–395. DOI: 10.1016/S0304-3975(98)00075-9 (cited on page 21).

[83] D. J. Willshaw and Christoph von der Malsburg. 'How patterned neural connections can be set up by self-organization'. en. In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 194.1117 (Nov. 1976), pp. 431–445. DOI: 10.1098/rspb.1976.0087 (cited on page 21).

[84] D. J. Willshaw and Christoph von der Malsburg. 'A marker induction mechanism for the establishment of ordered neural mappings: its application to the retinotectal problem'. en. In: *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 287.1021 (Nov. 1979), pp. 203–243. DOI: 10.1098/rstb.1979.0056 (cited on pages 21, 74).

[85] C. von der Malsburg and E Bienenstock. 'A Neural Network for the Retrieval of Superimposed Connection Patterns'. In: *Europhysics Letters (EPL)* 3.11 (June 1987), pp. 1243–1249. DOI: 10.1209/0295-5075/3/11/015 (cited on page 21).

[86] C. von der Malsburg. 'Concerning the Neuronal Code'. In: *Journal of Cognitive Science* 19.4 (Dec. 2018), pp. 511–550. DOI: 10.17791/JCS.2018.19.4.511 (cited on page 21).

[87] Walter J Freeman III and Christine A Skarda. 'Representations: Who needs them?' In: (1990) (cited on page 21).

[88] D. W. Arathorn. *Map-seeking circuits in visual cognition: a computational mechanism for biological and machine vision*. Stanford, Calif: Stanford University Press, 2002 (cited on page 22).

[89] Bruno A. Olshausen, Charles H. Anderson, and David C. Van Essen. 'A multiscale dynamic routing circuit for forming size- and position-invariant object representations'. In: *Journal of Computational Neuroscience* 2.1 (Mar. 1995), pp. 45–62. DOI: 10.1007/BF00962707 (cited on page 22).

[90] Tomas Fernandes and Christoph von der Malsburg. 'Self-Organization of Control Circuits for Invariant Fiber Projections'. en. In: *Neural Computation* 27.5 (May 2015), pp. 1005–1032. DOI: 10.1162/NECO_a_00725 (cited on pages 22, 74).

[91] Claude Lehmann. 'Leveraging Neuroscience for Deep Learning Based Object Recognition'. MA thesis. Zurich University of Applied Sciences, 2022 (cited on page 22).

[92] Marco Dorigo and Luca Maria Gambardella. 'Ant colony system: a cooperative learning approach to the traveling salesman problem'. In: *IEEE Transactions on evolutionary computation* 1.1 (1997), pp. 53–66 (cited on page 23).

[93] Edvinas Byla and Wei Pang. 'DeepSwarm: Optimising Convolutional Neural Networks Using Swarm Intelligence'. en. In: *Advances in Computational Intelligence Systems*. Ed. by Zhaojie Ju et al. Vol. 1043. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2020, pp. 119–130. DOI: 10.1007/978-3-030-29933-0_10 (cited on page 23).

[94] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017 (cited on page 23).

[95] Alex Krizhevsky, Geoffrey Hinton, et al. 'Learning multiple layers of features from tiny images'. In: (2009) (cited on pages 23, 30).

[96] Stephen Wolfram. 'Cellular automata as models of complexity'. In: *Nature* 311.5985 (Oct. 1984), pp. 419–424. DOI: 10.1038/311419a0 (cited on page 23).

[97] Gérard Y. Vichniac. 'Simulating physics with cellular automata'. In: *Physica D: Nonlinear Phenomena* 10.1 (1984), pp. 96–116. DOI: https://doi.org/10.1016/0167-2789(84)90253-7 (cited on page 23).

[98] N. H. Wulff and John A. Hertz. 'Learning Cellular Automation Dynamics with Neural Networks'. In: *NIPS*. 1992 (cited on page 23).

[99] William Gilpin. 'Cellular automata as convolutional neural networks'. In: *Phys. Rev. E* 100 (3 Sept. 2019), p. 032402. DOI: 10.1103/PhysRevE.100.032402 (cited on page 23).

[100] Alexander Mordvintsev et al. 'Growing Neural Cellular Automata'. In: *Distill* (2020) (cited on page 23).

[101] Alexander Mordvintsev, Ettore Randazzo, and Craig Fouts. 'Growing Isotropic Neural Cellular Automata'. In: arXiv:2205.01681 (June 2022). arXiv:2205.01681 [cs, q-bio] (cited on page 23).

[102] Alexander Mordvintsev et al. 'Growing Neural Cellular Automata'. In: *Distill* 5.2 (2020), e23 (cited on page 23).

[103] Rasmus Berg Palm et al. 'Variational Neural Cellular Automata'. In: arXiv:2201.12360 (Feb. 2022). arXiv:2201.12360 [cs] (cited on page 23).

[104] Diederik P. Kingma and Max Welling. 'Auto-Encoding Variational Bayes'. In: arXiv:1312.6114 (May 2014). arXiv:1312.6114 [cs, stat] (cited on pages 23, 31, 55, 56).

[105] Shyam Sudhakaran et al. 'Growing 3D Artefacts and Functional Machines with Neural Cellular Automata'. In: arXiv:2103.08737 (June 2021). arXiv:2103.08737 [cs] (cited on page 23).

[106] Kazuya Horibe, Kathryn Walker, and Sebastian Risi. 'Regenerating Soft Robots through Neural Cellular Automata'. In: arXiv:2102.02579 (Feb. 2021). arXiv:2102.02579 [cs, q-bio] (cited on page 23).

[107] Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. 'Learning Graph Cellular Automata'. In: arXiv:2110.14237 (Oct. 2021). arXiv:2110.14237 [cs] (cited on page 23).

[108] Jie Zhou et al. 'Graph Neural Networks: A Review of Methods and Applications'. In: arXiv:1812.08434 (Oct. 2021). arXiv:1812.08434 [cs, stat] (cited on page 23).

[109] Ettore Randazzo et al. 'Self-classifying MNIST Digits'. In: *Distill* (2020). https://distill.pub/2020/selforg/mnist. DOI: 10.23915/distill.00027.002 (cited on page 23).

[110] Alexandre Variengien et al. 'Towards self-organized control: Using neural cellular automata to robustly control a cart-pole agent'. In: arXiv:2106.15240 (July 2021). arXiv:2106.15240 [cs] (cited on page 24).

[111] Volodymyr Mnih et al. 'Playing Atari with Deep Reinforcement Learning'. In: arXiv:1312.5602 (Dec. 2013). arXiv:1312.5602 [cs] (cited on page 24).

[112] Elias Najarro and Sebastian Risi. 'Meta-Learning through Hebbian Plasticity in Random Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 20719–20731 (cited on pages 24, 83).

[113] Joachim Winther Pedersen and Sebastian Risi. 'Evolving and Merging Hebbian Learning Rules: Increasing Generalization by Decreasing the Number of Rules'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. arXiv:2104.07959 [cs]. June 2021, pp. 892–900. DOI: 10.1145/3449639.3459317 (cited on page 24).

[114] Louis Kirsch and Jürgen Schmidhuber. 'Meta Learning Backpropagation And Improving It'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021 (cited on pages 24, 33).

[115] Sebastian Risi. 'The Future of Artificial Intelligence is Self-Organizing and Self-Assembling'. In: *sebastianrisi.com* (2021) (cited on page 24).

[116] Teuvo Kohonen. 'Self-organized formation of topologically correct feature maps'. en. In: *Biological Cybernetics* 43.1 (1982), pp. 59–69. DOI: 10.1007/BF00337288 (cited on page 24).

[117] Teuvo Kohonen. *Self-Organization and Associative Memory*. eng. Third edition. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989 (cited on pages 24, 27).

[118] Bernd Fritzke. 'A Growing Neural Gas Network Learns Topologies'. In: *Advances in Neural Information Processing Systems*. Ed. by G. Tesauro, D. Touretzky, and T. Leen. Vol. 7. MIT Press, 1994 (cited on page 25).

[119] Douglas L. Reilly, Leon N. Cooper, and Charles Elbaum. 'A neural model for category learning'. en. In: *Biological Cybernetics* 45.1 (Aug. 1982), pp. 35–41. DOI: 10.1007/BF00387211 (cited on page 25).

[120] Bernd Fritzke. 'Growing cell structures—A self-organizing network for unsupervised and supervised learning'. en. In: *Neural Networks* 7.9 (Jan. 1994), pp. 1441–1460. DOI: 10.1016/0893-6080(94)90091-4 (cited on page 25).

[121] Stephen Marsland, Jonathan Shapiro, and Ulrich Nehmzow. 'A self-organising network that grows when required'. en. In: *Neural Networks* 15.8–9 (Oct. 2002), pp. 1041–1058. DOI: 10.1016/S0893-6080(02)00078-3 (cited on page 25).

[122] Luiza Mici, German I. Parisi, and Stefan Wermter. 'A self-organizing neural network architecture for learning human-object interactions'. en. In: *Neurocomputing* 307 (Sept. 2018), pp. 14–24. DOI: 10.1016/j.neucom.2018.04.015 (cited on page 25).

[123] Mike Davies et al. 'Loihi: A Neuromorphic Manycore Processor with On-Chip Learning'. In: *IEEE Micro* 38.1 (2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359 (cited on page 25).

[124] Timothée Masquelier and Simon J Thorpe. 'Unsupervised learning of visual features through spike timing dependent plasticity'. In: *PLoS computational biology* 3.2 (2007), e31 (cited on page 25).

[125] Qiang Yu et al. 'Rapid Feedforward Computation by Temporal Encoding and Learning With Spiking Neurons'. In: *IEEE Transactions on Neural Networks and Learning Systems* 24.10 (2013), pp. 1539–1552. DOI: 10.1109/TNNLS.2013.2245677 (cited on page 25).

[126] Saeed Reza Kheradpisheh et al. 'STDP-based spiking deep convolutional neural networks for object recognition'. In: *Neural Networks* 99 (2018), pp. 56–67 (cited on page 25).

[127] Milad Mozafari et al. 'Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks'. In: *Pattern recognition* 94 (2019), pp. 87–95 (cited on page 25).

[128] Peter U Diehl and Matthew Cook. 'Unsupervised learning of digit recognition using spike-timing-dependent plasticity'. In: *Frontiers in computational neuroscience* 9 (2015), p. 99 (cited on page 25).

[129] Yongqiang Cao, Yang Chen, and Deepak Khosla. 'Spiking deep convolutional neural networks for energy-efficient object recognition'. In: *International Journal of Computer Vision* 113.1 (2015), pp. 54–66 (cited on page 25).

[130] Amirhossein Tavanaei and Anthony S Maida. 'Bio-inspired spiking convolutional neural network using layer-wise sparse coding and STDP learning'. In: *arXiv preprint arXiv:1611.03000* (2016) (cited on page 25).

[131] Peter U Diehl et al. 'Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing'. In: *2015 International joint conference on neural networks (IJCNN)*. ieee. 2015, pp. 1–8 (cited on page 25).

[132] Friedemann Zenke and Surya Ganguli. 'Superspike: Supervised learning in multilayer spiking neural networks'. In: *Neural computation* 30.6 (2018), pp. 1514–1541 (cited on page 25).

[133] Paul Ferré, Franck Mamalet, and Simon J Thorpe. 'Unsupervised feature learning with winner-takes-all based stdp'. In: *Frontiers in computational neuroscience* 12 (2018), p. 24 (cited on page 25).

[134] Milad Mozafari et al. 'First-spike-based visual categorization using reward-modulated STDP'. In: *IEEE transactions on neural networks and learning systems* 29.12 (2018), pp. 6178–6190 (cited on page 25).

[135] Guruprasad Raghavan, Cong Lin, and Matt Thomson. 'Self-organization of multi-layer spiking neural networks'. In: arXiv:2006.06902 (June 2020). arXiv:2006.06902 [cs, q-bio] (cited on page 26).

[136] Guruprasad Raghavan and Matt Thomson. 'Neural networks grown and self-organized by noise'. In: *NeurIPS*. 2019 (cited on page 26).

[137] Ruthvik Vaila, John Chiasson, and Vishal Saxena. 'Deep Convolutional Spiking Neural Networks for Image Classification'. In: arXiv:1903.12272 (Sept. 2019). arXiv:1903.12272 [cs] (cited on page 26).

[138] Francis Crick. 'The recent excitement about neural networks'. en. In: *Nature* 337.6203 (Jan. 1989), pp. 129–132. DOI: 10.1038/337129a0 (cited on pages 27, 37).

[139] Stephen Grossberg. 'Competitive Learning: From Interactive Activation to Adaptive Resonance'. en. In: *Cognitive Science* 11.1 (Jan. 1987), pp. 23–63. DOI: 10.1111/j.1551-6708.1987.tb00862.x (cited on pages 27, 37).

[140] Jingzhao Zhang et al. 'Why gradient clipping accelerates training: A theoretical justification for adaptivity'. In: arXiv:1905.11881 (Feb. 2020). arXiv:1905.11881 [cs, math] (cited on page 27).

[141] Chen-Yu Lee et al. 'Deeply-Supervised Nets'. In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Guy Lebanon and S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, May 2015, pp. 562–570 (cited on page 27).

[142] Bernard Widrow et al. 'Nature's Learning Rule'. en. In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 1–30. DOI: 10.1016/B978-0-12-815480-9.00001-3 (cited on page 27).

[143] Kamil A. Grajski and Michael M. Merzenich. 'Hebb-Type Dynamics is Sufficient to Account for the Inverse Magnification Rule in Cortical Somatotopy'. en. In: *Neural Computation* 2.1 (Mar. 1990), pp. 71–84. DOI: 10.1162/neco.1990.2.1.71 (cited on page 27).

[144] P. Read Montague, Joseph A. Gally, and Gerald M. Edelman. 'Spatial Signaling in the Development and Function of Neural Connections'. en. In: *Cerebral Cortex* 1.3 (1991), pp. 199–220. DOI: 10.1093/cercor/1.3.199 (cited on page 27).

[145] Bartlett W. Mel. 'NMDA-Based Pattern Discrimination in a Modeled Cortical Neuron'. en. In: *Neural Computation* 4.4 (July 1992), pp. 502–517. DOI: 10.1162/neco.1992.4.4.502 (cited on page 27).

[146] Russell W. Anderson. 'Biased Random-Walk Learning: A Neurobiological Correlate to Trial-and-Error'. en. In: *Neural Networks and Pattern Recognition*. Elsevier, 1998, pp. 221–244. DOI: 10.1016/B978-012526420-4/50008-2 (cited on page 27).

[147] David E. Rumelhart and James L. McClelland. 'Learning Internal Representations by Error Propagation'. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362 (cited on page 27).

[148] Stephen Grossberg and Nestor A. Schmajuk. 'Neural dynamics of adaptive timing and temporal discrimination during associative learning'. en. In: *Neural Networks* 2.2 (Jan. 1989), pp. 79–102. DOI: 10.1016/0893-6080(89)90026-9 (cited on page 27).

[149] Shiyu Duan, Shujian Yu, and Jose C. Principe. 'Modularizing Deep Learning via Pairwise Learning With Kernels'. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.4 (Apr. 2022), pp. 1441–1451. DOI: 10.1109/TNNLS.2020.3042346 (cited on page 27).

[150] Shiyu Duan et al. 'On Kernel Method–Based Connectionist Models and Supervised Deep Learning Without Backpropagation'. en. In: *Neural Computation* 32.1 (Jan. 2020), pp. 97–135. DOI: 10.1162/neco_a_01250 (cited on page 27).

[151] Yulin Wang et al. 'Revisiting Locally Supervised Learning: an Alternative to End-to-end Training'. In: arXiv:2101.10832 (Jan. 2021). arXiv:2101.10832 [cs, stat] (cited on page 27).

[152] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. 'Greedy layerwise learning can scale to imagenet'. In: *International conference on machine learning*. PMLR. 2019, pp. 583–593 (cited on pages 27, 28).

[153] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. 'Deep Supervised Learning Using Local Errors'. In: *Frontiers in Neuroscience* 12 (Aug. 2018), p. 608. DOI: 10.3389/fnins.2018.00608 (cited on page 27).

[154] Enrique S. Marquez, Jonathon S. Hare, and Mahesan Niranjan. 'Deep Cascade Learning'. In: *IEEE Transactions on Neural Networks and Learning Systems* 29.11 (Nov. 2018), pp. 5475–5485. DOI: 10.1109/TNNLS.2018.2805098 (cited on page 27).

[155] Arild Nokland and Lars Hiller Eidnes. 'Training Neural Networks with Local Error Signals'. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, June 2019, pp. 4839–4850 (cited on page 27).

[156] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. 'Decoupled Greedy Learning of CNNs'. In: *Proceedings of the 37th International Conference on Machine Learning*. ICML'20. JMLR.org, 2020 (cited on page 28).

[157] Geoffrey Hinton. *The Forward-Forward Algorithm: Some Preliminary Investigations*. 2022. URL: https://www.cs.toronto.edu/~hinton/FFA13.pdf (visited on 12/17/2022) (cited on pages 28, 53, 73).

[158] Geoffrey Hinton. 'How to represent part-whole hierarchies in a neural network'. In: arXiv:2102.12627 (Feb. 2021). arXiv:2102.12627 [cs] (cited on page 28).

[159] Yoshua Bengio. 'How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation'. In: arXiv:1407.7906 (Sept. 2014). arXiv:1407.7906 [cs] (cited on page 29).

[160] Dong-Hyun Lee et al. 'Difference Target Propagation'. en. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Annalisa Appice et al. Vol. 9284. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 498–515. DOI: 10.1007/978-3-319-23528-8_31 (cited on pages 29, 30).

[161] Alexander Meulemans et al. 'A Theoretical Framework for Target Propagation'. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020 (cited on page 29).

[162] Max Jaderberg et al. 'Decoupled Neural Interfaces using Synthetic Gradients'. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 2017, pp. 1627–1635 (cited on page 30).

[163] Wojciech Marian Czarnecki et al. 'Understanding Synthetic Gradients and Decoupled Neural Interfaces'. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 904–912 (cited on page 30).

[164] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. 'Learning to solve the credit assignment problem'. In: arXiv:1906.00889 (Apr. 2020). arXiv:1906.00889 [cs, q-bio] (cited on page 30).

[165] Arild Nokland. 'Direct Feedback Alignment Provides Learning in Deep Neural Networks'. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 1045–1053 (cited on page 30).

[166] Timothy P. Lillicrap et al. 'Random synaptic feedback weights support error backpropagation for deep learning'. en. In: *Nature Communications* 7.1 (Dec. 2016), p. 13276. DOI: 10.1038/ncomms13276 (cited on page 30).

[167] Will Xiao et al. 'Biologically-plausible learning algorithms can scale to large datasets.' In: *International Conference on Learning Representations, (ICLR 2019)*. 2019 (cited on page 30).

[168] David Balduzzi, Hastagiri Vanchinathan, and Joachim M. Buhmann. 'Kickback Cuts Backprop's Red-Tape: Biologically Plausible Credit Assignment in Neural Networks'. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, 2015, pp. 485–491 (cited on page 30).

[169] Qianli Liao, Joel Z. Leibo, and Tomaso Poggio. 'How Important is Weight Symmetry in Backpropagation?' In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, 2016, pp. 1837–1844 (cited on page 30).

[170] Sergey Bartunov et al. 'Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures'. In: arXiv:1807.04587 (Nov. 2018). arXiv:1807.04587 [cs, stat] (cited on pages 30, 37, 54).

[171] Miguel Carreira-Perpinan and Weiran Wang. 'Distributed optimization of deeply nested systems'. In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*. Ed. by Samuel Kaski and Jukka Corander. Vol. 33. Proceedings of Machine Learning Research. Reykjavik, Iceland: PMLR, Apr. 2014, pp. 10–19 (cited on page 30).

[172] Gavin Taylor et al. 'Training Neural Networks without Gradients: A Scalable ADMM Approach'. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 2722–2731 (cited on page 30).

[173] Ziming Zhang and Matthew Brand. 'Convergent Block Coordinate Descent for Training Tikhonov Regularized Deep Neural Networks'. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1719–1728 (cited on page 30).

[174] Tim Tsz-Kit Lau et al. 'A Proximal Block Coordinate Descent Algorithm for Deep Neural Network Training'. In: arXiv:1803.09082 (Mar. 2018). arXiv:1803.09082 [cs, math, stat] (cited on page 30).

[175] Shiyu Duan and Jose C. Principe. 'Training Deep Architectures Without End-to-End Backpropagation: A Survey on the Provably Optimal Methods'. In: arXiv:2101.03419 (Aug. 2022). arXiv:2101.03419 [cs, stat] (cited on page 30).

[176] Jia Deng et al. 'ImageNet: A large-scale hierarchical image database'. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Miami, FL: IEEE, June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cited on page 30).

[177] Pierre Baldi. 'Autoencoders, Unsupervised Learning, and Deep Architectures'. In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. Ed. by Isabelle Guyon et al. Vol. 27. Proceedings of Machine Learning Research. Bellevue, Washington, USA: PMLR, July 2012, pp. 37–49 (cited on page 31).

[178] Marc'Aurelio Ranzato et al. 'Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition'. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. Minneapolis, MN, USA: IEEE, June 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383157 (cited on page 31).

[179] Quoc V. Le et al. 'Building High-Level Features Using Large Scale Unsupervised Learning'. In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, 2012, pp. 507–514 (cited on pages 31, 45, 95).

[180] Pascal Vincent et al. 'Extracting and Composing Robust Features with Denoising Autoencoders'. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294 (cited on page 31).

[181] Salah Rifai et al. 'Contractive Auto-Encoders: Explicit Invariance during Feature Extraction'. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 833–840 (cited on page 31).

[182] Omar Elharrouss et al. 'Image Inpainting: A Review'. en. In: *Neural Processing Letters* 51.2 (Apr. 2020), pp. 2007–2028. DOI: 10.1007/s11063-019-10163-0 (cited on page 32).

[183] Deepak Pathak et al. 'Context Encoders: Feature Learning by Inpainting'. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 2536–2544. DOI: 10.1109/CVPR.2016.278 (cited on page 32).

[184] Kaiming He et al. 'Masked autoencoders are scalable vision learners'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16000–16009 (cited on page 32).

[185] Yuge Shi et al. 'Adversarial masking for self-supervised learning'. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 20026–20040 (cited on page 32).

[186] Nikos Komodakis and Spyros Gidaris. 'Unsupervised representation learning by predicting image rotations'. In: *International Conference on Learning Representations (ICLR)*. 2018 (cited on page 32).

[187] Xiaohua Zhai et al. 'S4L: Self-Supervised Semi-Supervised Learning'. In: arXiv:1905.03670 (July 2019). arXiv:1905.03670 [cs] (cited on page 32).

[188] Yanbei Chen et al. 'Semi-Supervised and Unsupervised Deep Visual Learning: A Survey'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022) (cited on page 32).

[189] Ting Chen et al. 'A simple framework for contrastive learning of visual representations'. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607 (cited on pages 32, 42).

[190] Kaiming He et al. 'Momentum contrast for unsupervised visual representation learning'. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9729–9738 (cited on page 32).

[191] Mathilde Caron et al. 'Unsupervised learning of visual features by contrasting cluster assignments'. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9912–9924 (cited on pages 32, 42).

[192] Prannay Khosla et al. 'Supervised contrastive learning'. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 18661–18673 (cited on page 32).

[193] Timothy M Hospedales et al. 'Meta-Learning in Neural Networks: A Survey'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/TPAMI.2021.3079209 (cited on pages 32, 33).

[194] Sebastian Thrun and Lorien Pratt. 'Learning to Learn: Introduction and Overview'. en. In: *Learning to Learn*. Ed. by Sebastian Thrun and Lorien Pratt. Boston, MA: Springer US, 1998, pp. 3–17. DOI: 10.1007/978-1-4615-5529-2_1 (cited on page 32).

[195] Allan M. Schrier. 'Learning how to learn: The significance and current status of learning set formation'. en. In: *Primates* 25.1 (Jan. 1984), pp. 95–102. DOI: 10.1007/BF02382299 (cited on page 32).

[196] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 'Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks'. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 1126–1135 (cited on pages 33, 34).

[197] Chelsea Finn, Kelvin Xu, and Sergey Levine. 'Probabilistic Model-Agnostic Meta-Learning'. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montreal, Canada: Curran Associates Inc., 2018, pp. 9537–9548 (cited on page 33).

[198] Yoonho Lee and Seungjin Choi. 'Gradient-based meta-learning with learned layerwise metric and subspace'. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2927–2936 (cited on page 33).

[199] Siyuan Qiao et al. 'Few-shot image recognition by predicting parameters from activations'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7229–7238 (cited on page 33).

[200] Andrei A Rusu et al. 'Meta-learning with latent embedding optimization'. In: *arXiv preprint arXiv:1807.05960* (2018) (cited on page 33).

[201] TM Heskes. 'Empirical Bayes for learning to learn'. In: (2000) (cited on page 33).

[202] James Requeima et al. 'Fast and Flexible Multi-Task Classification Using Conditional Neural Adaptive Processes'. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019 (cited on page 33).

[203] David Ha, Andrew Dai, and Quoc V. Le. 'HyperNetworks'. In: arXiv:1609.09106 (Dec. 2016). arXiv:1609.09106 [cs] (cited on page 33).

[204] Jake Snell, Kevin Swersky, and Richard Zemel. 'Prototypical Networks for Few-Shot Learning'. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 4080–4090 (cited on pages 33, 34).

[205] Wei-Yu Chen et al. 'A Closer Look at Few-shot Classification'. In: arXiv:1904.04232 (Jan. 2020). arXiv:1904.04232 [cs] (cited on page 33).

[206] Sachin Ravi and Hugo Larochelle. 'Optimization as a Model for Few-Shot Learning'. In: (2017) (cited on pages 33, 34).

[207] Ke Li and Jitendra Malik. 'Learning to Optimize'. In: arXiv:1606.01885 (June 2016). arXiv:1606.01885 [cs, math, stat] (cited on page 33).

[208] Zhenguo Li et al. 'Meta-SGD: Learning to Learn Quickly for Few-Shot Learning'. In: arXiv:1707.09835 (Sept. 2017). arXiv:1707.09835 [cs] (cited on page 33).

[209] Eunbyung Park and Junier B. Oliva. 'Meta-Curvature'. In: arXiv:1902.03356 (Jan. 2020). arXiv:1902.03356 [cs, stat] (cited on page 33).

[210] Rein Houthooft et al. 'Evolved Policy Gradients'. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018 (cited on page 33).

[211] Flood Sung et al. 'Learning to Learn: Meta-Critic Networks for Sample Efficient Learning'. In: arXiv:1706.09529 (June 2017). arXiv:1706.09529 [cs] (cited on page 33).

[212] Giulia Denevi et al. 'Learning To Learn Around A Common Mean'. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018 (cited on page 33).

[213] Giulia Denevi et al. 'Online-Within-Online Meta-Learning'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019 (cited on page 33).

[214] Santiago Gonzalez and Risto Miikkulainen. 'Improved training speed, accuracy, and data utilization through loss function optimization'. In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2020, pp. 1–8 (cited on page 33).

[215] Yiying Li et al. 'Feature-Critic Networks for Heterogeneous Domain Generalization'. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3915–3924 (cited on pages 33, 34).

[216] Antreas Antoniou and Amos J Storkey. 'Learning to Learn By Self-Critique'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019 (cited on page 33).

[217] Rinu Boney and Alexander Ilin. 'Semi-Supervised Few-Shot Learning with MAML'. In: *International Conference on Learning Representations*. 2018 (cited on page 33).

[218] Barret Zoph and Quoc Le. 'Neural Architecture Search with Reinforcement Learning'. In: *International Conference on Learning Representations*. 2017 (cited on page 33).

[219] Justin Bayer et al. 'Evolving Memory Cell Structures for Sequence Learning'. In: *Artificial Neural Networks – ICANN 2009*. Ed. by Cesare Alippi et al. Vol. 5769. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 755–764. DOI: 10.1007/978-3-642-04277-5_76 (cited on page 33).

[220] Esteban Real et al. 'Regularized Evolution for Image Classifier Architecture Search'. In: AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. DOI: 10.1609/aaai.v33i01.33014780 (cited on page 33).

[221] Luca Franceschi et al. 'Bilevel Programming for Hyperparameter Optimization and Meta-Learning'. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 1568–1577 (cited on pages 33, 34).

[222] Paul Micaelli and Amos Storkey. 'Gradient-based Hyperparameter Optimization Over Long Horizons'. In: arXiv:2007.07869 (Sept. 2021). arXiv:2007.07869 [cs, stat] (cited on pages 33, 34).

[223] Luca Franceschi et al. 'Forward and Reverse Gradient-Based Hyperparameter Optimization'. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 1165–1173 (cited on pages 33, 34).

[224] Ekin D. Cubuk et al. 'AutoAugment: Learning Augmentation Strategies From Data'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019 (cited on pages 33, 34).

[225] Yonggang Li et al. 'Differentiable Automatic Data Augmentation'. en. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Vol. 12367. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 580–595. DOI: 10.1007/978-3-030-58542-6_35 (cited on page 33).

[226] Cheng Zhang et al. 'Active Mini-Batch Sampling Using Repulsive Point Processes'. In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. DOI: 10.1609/aaai.v33i01.33015741 (cited on page 33).

[227] Yang Fan et al. 'Learning to Teach'. In: *International Conference on Learning Representations*. 2018 (cited on page 33).

[228] Tongzhou Wang et al. 'Dataset Distillation'. In: arXiv:1811.10959 (Feb. 2020). arXiv:1811.10959 [cs, stat] (cited on page 33).

[229] Jonathan Lorraine, Paul Vicol, and David Duvenaud. 'Optimizing Millions of Hyperparameters by Implicit Differentiation'. In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, Aug. 2020, pp. 1540–1552 (cited on pages 33, 34).

[230] OpenAI: Marcin Andrychowicz et al. 'Learning dexterous in-hand manipulation'. en. In: *The International Journal of Robotics Research* 39.1 (Jan. 2020), pp. 3–20. DOI: 10.1177/0278364919887447 (cited on page 34).

[231] Nataniel Ruiz, Samuel Schulter, and Manmohan Chandraker. 'Learning To Simulate'. In: *International Conference on Learning Representations*. 2019 (cited on page 34).

[232] Quan Vuong et al. 'How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies?' In: arXiv:1903.11774 (Mar. 2019). arXiv:1903.11774 [cs, stat] (cited on page 34).

[233] Chen Huang et al. 'Addressing the loss-metric mismatch with adaptive loss alignment'. In: *International conference on machine learning*. PMLR. 2019, pp. 2891–2900 (cited on page 34).

[234] Yan Duan et al. 'RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning'. In: arXiv:1611.02779 (Nov. 2016). arXiv:1611.02779 [cs, stat] (cited on page 34).

[235] Jurgen Schmidhuber. 'Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook'. Diploma Thesis. Technische Universitat Munchen, Germany, May 1987 (cited on page 34).

[236] Kenneth O. Stanley et al. 'Designing neural networks through neuroevolution'. en. In: *Nature Machine Intelligence* 1.1 (Jan. 2019), pp. 24–35. DOI: 10.1038/s42256-018-0006-z (cited on page 34).

[237] Tim Salimans et al. 'Evolution Strategies as a Scalable Alternative to Reinforcement Learning'. In: arXiv:1703.03864 (Sept. 2017). arXiv:1703.03864 [cs, stat] (cited on pages 34, 83).

[238] Olga Wichrowska et al. 'Learned Optimizers That Scale and Generalize'. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 3751–3760 (cited on page 34).

[239] Antreas Antoniou, Harrison Edwards, and Amos Storkey. 'How to train your MAML'. In: *International Conference on Learning Representations*. 2019 (cited on page 34).

[240] JA Scott Kelso. *Dynamic patterns: The self-organization of brain and behavior*. MIT press, 1995 (cited on page 36).

[241] Birgitta Dresp. 'Seven Properties of Self-Organization in the Human Brain'. In: *Big Data Cogn. Comput.* 4 (2020), p. 10 (cited on page 36).

[242] Timothy P. Lillicrap et al. 'Random feedback weights support learning in deep neural networks'. In: arXiv:1411.0247 (Nov. 2014). arXiv:1411.0247 [cs, q-bio] (cited on page 37).

[243] Randall C. O'Reilly. 'Biologically Plausible Error-Driven Learning Using Local Activation Differences: The Generalized Recirculation Algorithm'. en. In: *Neural Computation* 8.5 (July 1996), pp. 895–938. DOI: 10.1162/neco.1996.8.5.895 (cited on page 37).

[244] Yann Le Cun. 'Learning Process in an Asymmetric Threshold Network'. en. In: *Disordered Systems and Biological Organization*. Ed. by E. Bienenstock, F. Fogelman Soulié, and G. Weisbuch. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 233–240. DOI: 10.1007/978-3-642-82657-3_24 (cited on page 37).

[245] Jia Deng et al. 'Imagenet: A large-scale hierarchical image database'. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (cited on page 37).

[246] Yann LeCun and Corinna Cortes. *THE MNIST DATABASE of handwritten digits*. 1996. URL: http://yann.lecun.com/exdb/mnist/ (visited on 09/05/2022) (cited on pages 37, 84).

[247] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 'CIFAR-10 (Canadian Institute for Advanced Research)'. In: () (cited on page 37).

[248] Javier R. Movellan. 'Contrastive Hebbian Learning in the Continuous Hopfield Model'. en. In: *Connectionist Models*. Elsevier, 1991, pp. 10–17. DOI: 10.1016/B978-1-4832-1448-1.50007-X (cited on page 37).

[249] Yu-An Chung et al. 'An Unsupervised Autoregressive Model for Speech Representation Learning'. In: arXiv:1904.03240 (June 2019). arXiv:1904.03240 [cs, eess] (cited on page 39).

[250] Nathalie L. Rochefort et al. 'Sparsification of neuronal activity in the visual cortex at eye-opening'. en. In: *Proceedings of the National Academy of Sciences* 106.35 (Sept. 2009), pp. 15049–15054. DOI: 10.1073/pnas.0907660106 (cited on page 40).

[251] Christos Louizos, Max Welling, and Diederik P. Kingma. 'Learning Sparse Neural Networks through L_0 Regularization'. In: arXiv:1712.01312 (June 2018). arXiv:1712.01312 [cs, stat] (cited on page 40).

[252] Torsten Hoefler et al. 'Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks'. In: arXiv:2102.00554 (Jan. 2021). arXiv:2102.00554 [cs] (cited on page 40).

[253] Konstantinos P. Panousis, Sotirios Chatzis, and Sergios Theodoridis. 'Stochastic Local Winner-Takes-All Networks Enable Profound Adversarial Robustness'. In: arXiv:2112.02671 (Dec. 2021). arXiv:2112.02671 [cs, stat] (cited on page 40).

[254] Charles D Gilbert, Joseph A Hirsch, and Torsten N Wiesel. 'Lateral interactions in visual cortex'. In: *Cold Spring Harbor symposia on quantitative biology*. Vol. 55. Cold Spring Harbor Laboratory Press. 1990, pp. 663–677 (cited on page 40).

[255] Ting Chen et al. 'Big self-supervised models are strong semi-supervised learners'. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 22243–22255 (cited on page 42).

[256] Georg B. Keller, Tobias Bonhoeffer, and Mark Hübener. 'Sensorimotor Mismatch Signals in Primary Visual Cortex of the Behaving Mouse'. en. In: *Neuron* 74.5 (June 2012), pp. 809–815. DOI: 10.1016/j.neuron.2012.03.040 (cited on pages 42, 77).

[257] Jan Storck, Sepp Hochreiter, Jürgen Schmidhuber, et al. 'Reinforcement driven information acquisition in non-deterministic environments'. In: *Proceedings of the international conference on artificial neural networks, Paris*. Vol. 2. 1995, pp. 159–164 (cited on pages 43, 79, 80).

[258] Magdalena Klapper-Rybicka, Nicol N. Schraudolph, and Jürgen Schmidhuber. 'Unsupervised Learning in LSTM Recurrent Neural Networks'. In: *Artificial Neural Networks — ICANN 2001*. Ed. by Georg Dorffner, Horst Bischof, and Kurt Hornik. Vol. 2130. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 684–691. DOI: 10.1007/3-540-44668-0_95 (cited on page 43).

[259] Vassileios Balntas et al. 'Learning local feature descriptors with triplets and shallow convolutional neural networks'. en. In: *Procedings of the British Machine Vision Conference 2016*. York, UK: British Machine Vision Association, 2016, pp. 119.1–119.11. DOI: 10.5244/C.30.119 (cited on page 47).

[260] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980 (cited on pages 48, 57, 83).

[261] Norman Mu and Justin Gilmer. 'MNIST-C: A Robustness Benchmark for Computer Vision'. In: arXiv:1906.02337 (June 2019). arXiv:1906.02337 [cs] (cited on pages 54, 65, 84).

[262] Samuel R Bowman et al. 'Generating sentences from a continuous space'. In: *20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016*. Association for Computational Linguistics (ACL). 2016, pp. 10–21 (cited on page 57).

[263] Hao Fu et al. 'Cyclical Annealing Schedule: A Simple Approach to Mitigating'. en. In: *Proceedings of the 2019 Conference of the North*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 240–250. DOI: 10.18653/v1/N19-1021 (cited on page 57).

[264] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu koray. 'Neural Discrete Representation Learning'. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017 (cited on page 65).

[265] Laurens van der Maaten and Geoffrey Hinton. 'Visualizing Data using t-SNE.' In: *Journal of Machine Learning Research* 9.11 (2008) (cited on page 66).

[266] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. en. The MIT Press, 2010 (cited on pages 73, 75).

[267] Philipp Wolfrum et al. 'A recurrent dynamic model for correspondence-based face recognition'. en. In: *Journal of Vision* 8.7 (Dec. 2008), p. 34. DOI: 10.1167/8.7.34 (cited on page 74).

[268] Laurenz Wiskott and Christoph von der Malsburg. *Face recognition by dynamic link matching*. Ruhr-Univ., Inst. für Neuroinformatik, 1996 (cited on page 74).

[269] Wolfgang Köhler. 'Gestalt psychology.' In: (1929) (cited on page 75).

[270] Jianwu Long, Zeran yan, and Hongfa chen. 'A Graph Neural Network for superpixel image classification'. In: *Journal of Physics: Conference Series* 1871.1 (Apr. 2021), p. 012071. DOI: 10.1088/1742-6596/1871/1/012071 (cited on page 76).

[271] Rex Ying et al. 'Hierarchical Graph Representation Learning with Differentiable Pooling'. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montreal, Canada: Curran Associates Inc., 2018, pp. 4805–4815 (cited on page 76).

[272] Varun Vasudevan et al. 'Image classification using graph neural network and multiscale wavelet superpixels'. en. In: *Pattern Recognition Letters* 166 (Feb. 2023), pp. 89–96. DOI: 10.1016/j.patrec.2023.01.003 (cited on page 76).

[273] Yujia Li et al. 'Graph matching networks for learning the similarity of graph structured objects'. In: *International conference on machine learning*. PMLR. 2019, pp. 3835–3845 (cited on page 76).

[274] Nancy Xu et al. *Image Keypoint Matching Using Graph Neural Networks*. en. Ed. by Rosa Maria Benito et al. Vol. 1016. Studies in Computational Intelligence. Cham: Springer International Publishing, 2022, pp. 441–451 (cited on page 76).

[275] Junuk Jung et al. 'Unified Negative Pair Generation toward Well-discriminative Feature Space for Face Recognition'. In: *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*. BMVA Press, 2022, p. 421 (cited on page 77).

[276] Hamza Keurti et al. 'Homomorphism Autoencoder – Learning Group Structured Representations from Observed Transitions'. In: arXiv:2207.12067 (July 2022). arXiv:2207.12067 [cs, math, stat] (cited on page 77).

[277] Jean Piaget. 'Part I: Cognitive development in children: Piaget development and learning'. en. In: *Journal of Research in Science Teaching* 2.3 (Sept. 1964), pp. 176–186. DOI: 10.1002/tea.3660020306 (cited on page 77).

[278] Yann LeCun. *A Path Towards Autonomous Machine Intelligence*. 2022. URL: https://openreview.net/forum?id=BZ5a1r-kVsf (visited on 09/21/2022) (cited on pages 77, 80, 95).

[279] Greg Brockman et al. *OpenAI Gym*. 2016 (cited on page 79).

[280] Emmanuel Bengio et al. 'Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation'. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 27381–27394 (cited on page 80).

[281] Yael Niv et al. 'Evolution of Reinforcement Learning in Uncertain Environments: Emergence of Risk-Aversion and Matching'. In: *Advances in Artificial Life*. Ed. by Jozef Kelemen and Petr Sosík. Vol. 2159. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 252–261. DOI: 10.1007/3-540-44811-X_27 (cited on page 83).

[282] Tailin Liang et al. 'Pruning and Quantization for Deep Neural Network Acceleration: A Survey'. In: arXiv:2101.09671 (June 2021). arXiv:2101.09671 [cs] (cited on page 85).