Master of Science in Engineering with Specialisation in Data Science

Master Thesis

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

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

written by

Pascal Sager

supervised by

Prof. Dr. Thilo Stadelmann

Dr. Jan Milan Deriu

Prof. Dr. Christoph von der Malsburg

March 27, 2023

Zurich University of Applied Sciences

Centre for Artificial intelligence

"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 Bildanalyse wie Klassifizierung sehr effizient, ist aber neurowissenschaftlich nicht plausibel und hat verschiedenste Schwächen wie fehlende Robustheit und Interpretierbarkeit sowie (noch) 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 Bildanalyse 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 können Erkenntnisse nicht direkt übernommen werden und ein grosser Beitrag dieser Arbeit besteht 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 auf der Untersuchung neuer Architekturideen und nicht auf dem Überbieten von Metriken wie Genauigkeit auf Benchmark-Datensätzen. 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 horizontaler und ein Modell mit vertikaler Selbst-Organisation vorgeschlagen. Das Modell mit horizontaler Selbst-Organisation optimiert jedes Modell-Layer separat mit Proxy-Fehlerfunktionen. 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 vertikaler Selbst-Organisation teilt die Eingabedaten in kleinere Einheiten auf und verteilt diese auf unabhängig trainierte Variational Autoencoders. 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, 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 analysis 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. Consequently, a significant contribution of this thesis is to adapt and 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. Furthermore, 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 horizontal self-organisation and a model with vertical self-organisation is proposed. The model with horizontal self-organisation optimises each layer of the model separately using proxy objective functions. 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 utilising vertical self-organisation splits the input data into smaller patches and distributes them to independently trained variational autoencoders. 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. Thus, I have learned to work scientifically on a daily basis, which is very beneficial for my future. 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. However, it is equally or even more important to reason why and how something works, to justify choices, and to show limitations. Moreover, scientific breakthroughs always require courage to try something completely new, even if this means that the work may not lead to outstanding results worth publishing. 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 spans two fields; computer science and neuroscience. I try to link these fields as clearly as possible and to write in a way so 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 an expert in one of these fields, feel free to skip (parts of) the fundamentals in Chapter 2. In general, this chapter also 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.

At this point, 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

Zusammenfassung iii

Abstract v

Preface vii

Contents ix

Mathematical Terms & Definitions xiii
1 Notation . . . xiii
2 Variables . . . xiii
3 Functions . . . xiv

1 Introduction 1
1.1 Contribution . . . 2
1.2 Organisation of Thesis . . . 2

2 Fundamentals 5
2.1 Biological Neurons . . . 5
2.2 Artificial Neural Networks . . . 6
2.2.1 Convolutional Networks . . . 9
2.3 Limitations . . . 11
2.4 Biological Learning . . . 13
2.5 Neurocomputing . . . 14
2.5.1 Hebbian Learning . . . 14
2.5.2 Hopfield Networks . . . 16
2.5.3 Spiking Neural Networks . . . 18
2.5.4 Reservoir Computing . . . 19

3 Related Work 21
3.1 Natural Intelligence . . . 21
3.2 Self-Organisation . . . 23
3.2.1 Growing Networks . . . 24
3.2.2 Spiking Networks . . . 25
3.2.3 Relevance . . . 26
3.3 Learning Algorithms . . . 26
3.3.1 Proxy Objective Functions . . . 27
3.3.2 Target Propagation . . . 28
3.3.3 Other Methods . . . 30
3.3.4 Relevance . . . 31
3.4 Representation Learning . . . 31
3.4.1 Autoencoders . . . 31
3.4.2 Self-Supervised Learning . . . 32
3.4.3 Relevance . . . 33
3.5 Meta-Learning . . . 33
3.5.1 Meta-Representation . . . 33
3.5.2 Meta-Optimiser . . . 34
3.5.3 Meta-Objective . . . 35

4 Neuroscientific Concepts **37**
4.1 The Discrepancy . . . 37
4.2 Self-Organisation . . . 38
4.3 Net Fragments . . . 40
4.3.1 Sparsity . . . 42
4.4 Lateral Connections . . . 42
4.5 Other Principles . . . 44

5 Horizontal Self-Organisation **47**
5.1 Methods . . . 47
5.1.1 Extraction of Representations . . . 50
5.1.2 Lateral Connections . . . 51
5.1.3 Hierarchical Features . . . 53
5.2 Results . . . 55
5.2.1 Dataset . . . 55
5.2.2 Baseline Model . . . 55
5.2.3 Lateral Connections . . . 56
5.2.4 Hierarchical Features . . . 56

6 Vertical Self-Organisation **57**
6.1 Methods . . . 57
6.1.1 Predicting bigger Patches . . . 60
6.1.2 Communication . . . 61
6.1.3 Class Prediction . . . 66
6.2 Results . . . 67

7 Future Work & Conclusion **73**
7.1 Neuroscientific Concepts . . . 73
7.2 Implemented Models . . . 74
7.2.1 Horizontal Self-Organisation . . . 74
7.2.2 Vertical Self-Organisation . . . 75
7.3 3-Staged Model . . . 76
7.4 Useful Representations . . . 77
7.5 Cognition and Reasoning . . . 79

Appendix **83**

A Experiments Hebbian Learning **85**
A.0.1 Hebbian Pruning . . . 85
A.0.2 Results . . . 87

B Net Fragments **89**
B.1 Classification . . . 90
B.1.1 Methods . . . 90
B.1.2 Results . . . 91
B.2 Autoencoders . . . 92
B.2.1 Methods . . . 92
B.2.2 Results . . . 94
B.3 Conclusion . . . 95

Bibliography **97**

List of Figures

2.1 Diagram of the components of a biological neuron 5
2.2 Overview of different image analysis tasks 11
2.3 Organization of the visual system in the cerebral cortex 13
2.4 Structure of an echo state network 19

4.1 Overview of horizontal and vertical self-organisation 40
4.2 Illustrative network architecture of a model with lateral connections 43

5.1 The flow of gradients within the network based on horizontal self-organisation 47
5.2 Architecture of the fully connected model with horizontal self-organisation . . 50
5.3 Lateral connections by concatenating the layer's output with the layer's input . 52
5.4 Data augmentation applied on 10 samples of the MNIST dataset 53
5.5 Architecture of the CNN for horizontal self-organisation 54
5.6 Horizontal self-organisation with mutual information loss 55

6.1 Annealing strategy of the KL weight term of variational autoencoders 59
6.2 Architecture of the VAEs 59
6.3 Architecture of the VAEs with bigger field-of-view up-sampling 60
6.4 Reconstruction of bigger output than input patches 60
6.5 Average sample of the MNIST dataset per class 61
6.6 Prediction of Gaussian parameters of other VAEs 62
6.7 Communication between VAEs by forwarding reconstructed patch 63
6.8 Communication between VAEs by forwarding prediction of neighbouring patches 64
6.9 Communication between VAEs by forwarding prediction of the entire image . 65
6.10 Communication between VAEs with a dedicated communication channel . . . 65
6.11 Change of the model's prediction over time 68
6.12 t-SNE plot of the μ values of different models 69

7.1 Different versions of the digit "4" in the MNIST dataset 75

A.1 Hebbian pruning network 86
A.2 NLL loss of the Hebbian pruning network 87

B.1 Straight line digits dataset 89
B.2 Network activations of the classification networks 92
B.3 Network activations of the smaller autoencoder model 94
B.4 Network activations of the bigger autoencoder model 95

List of Tables

6.1 Different models to evaluate vertical self-organisation 68
6.2 NMI score of different architectures 70
6.3 Accuracy of different architectures 71

A.1 NLL loss of the Hebbian pruning network 87

B.1 Different classification networks investigated for net fragments 91
B.2 Different autoencoders investigated for net fragments 92

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
Curved brackets with a point	$a(\cdot)$	A function, where $(\cdot)$ is a placeholder for a variable
Rectangular brackets	$a[t]$ $\boldsymbol{x}[c]$	Depends on context, either variable a at time t or vector $\boldsymbol{x}$ is from class c
Superscript rectangular brackets	$a^{[l]}$	Layer index: l is the index of the layer, e.g. $\boldsymbol{W}^{[l]}$ is the weight matrix of layer l
Superscript curved brackets	$a^{(i)}$	Sample index: i is the index of the sample, e.g. $\boldsymbol{x}^{(i)}$ is the i-th sample of a data set
Subscript & uppercase letter	$\boldsymbol{a}_{C=\bar{c}}$	The average activation of $\boldsymbol{a}$ for class c

2 Variables

Variable	Meaning
β	The weight factor of the Kullback-Leibler term in the loss of a variational autoencoder
λ	The weight factor for loss terms
η	The learning rate of an optimisation algorithm
ρ	The desired activation probability of neurons
$\hat{\rho}_i$	The activation probability of a neuron i
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 or as goodness threshold in the forward-forward network
a_i	The output of an activation function of a single neuron
$\boldsymbol{a}$	The output of an intermediate layer of a multi-layer network, the output of the last layer is typically denoted as $\hat{\boldsymbol{y}}$
b_i	The bias of a single neuron
$\boldsymbol{b}$	The bias of a network layer
C	The set of classes of a classification task
$\boldsymbol{h}$	The latent representations of a network that are typically used in a subsequent downstream task (specific instance of $\boldsymbol{a}$)
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

Variable	Meaning
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
$\boldsymbol{X} = \boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}$	A dataset with m samples
$\boldsymbol{x} = (x_1, ..., x_n)$	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
z	The output of the aggregation function $g(\cdot)$ of a single neuron
$\boldsymbol{z}$	The output of the aggregation functions $g_1(\cdot), ..., g_n(\cdot)$ of a network layer

3 Functions

Function	Meaning
$\mathcal{L}(\cdot)$	Loss function to calculate the goodness of the model's prediction
$D(\cdot)$	A decoder (multiple network layers) to transform a latent representation to an output
$d(\cdot)$	A decoder for a single layer to approximate the inverse of a layer h^{-1}
$E(\cdot)$	An encoder (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 (2.5))
$g(\cdot)$	The aggregation function that calculates the scalar value that is fed into the activation function $f(\cdot)$ (c.f. equation (2.1), equation (2.3), equation (2.4))
$h_l(\cdot)$	The function implemented by a layer l
$G(\cdot)$	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 (2.5))
$(\cdot)^+$	The ReLU activation function (c.f. equation (2.5))
$\tanh(\cdot)$	The hyperbolic tangent activation function (c.f. equation (2.5))

Introduction 1

1.1 Contribution 2
1.2 Organisation of Thesis 2

Humanity has always tried to simplify its life through technological progress. For example, the computer has influenced and changed every field imaginable in the last decades. 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. The problem is that we cannot describe to a computer how a cat looks precisely[1].

1: at least not on a pixel-level basis so that one can compare a given image with a precise 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 learning algorithm of such systems has serious flaws.

2: the Turing Award is recognised 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 is a learning system that does not have these problems. Thus, incorporating findings from

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

neuroscience into the deep 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 in this thesis. 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 further. 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 research dealing with alternative learning algorithms.
- Second, promising concepts from the neuroscience literature that may help to build 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 towards more general artificial intelligence.

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 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 proposes how these neuroscientific concepts could be implemented in

a deep learning setting. In Chapter 5 and 6, two different architectures incorporating these concepts are presented: Chapter 5 describes a network based on horizontal self-organisation. The proposed model trains each layer in parallel using proxy objective functions and can still extract hierarchical features. The model described in Chapter 6 uses vertical self-organisation by dividing the data into patches and analysing them on independent but communicating sub-models. 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

2.1 Biological Neurons 5
2.2 Artificial Neural Networks . . . 6
Convolutional Networks 9
2.3 Limitations 11
2.4 Biological Learning 13
2.5 Neurocomputing 14
Hebbian Learning 14
Hopfield Networks 16
Spiking Neural Networks . . 18
Reservoir Computing 19

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 step-wise improve the correlation function between input and output data (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. 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 itself. 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 position[1]. 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. Thus, the main difference between biological and artificial neurons is that biological neurons are highly time-dependent and fire asynchronously, whereas artificial neurons fire in a pre-defined synchronous order.

1: except for the activation function, c.f. Section 2.2

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 of 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. Their neuron takes a binary input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ 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)

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

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

In 1958, Rosenblatt [8] proposed the perceptron, which works with real numbers as input. Therefore, the input vector $\boldsymbol{x} \in \mathbb{R}^n$ is multiplied with a weight vector $\boldsymbol{w} \in \mathbb{R}^n$ of the same length n. 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. With real numbers as input, equation (2.2) remains the same but equation (2.1) is rewritten as

[8]: Rosenblatt (1958)

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

By convention, current literature usually uses a positive bias b instead of the negative threshold $-\theta$, leading to:

$$z = g(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = \left(\sum_{i=1}^{n} w_i \cdot x_i \right) + b \tag{2.4}$$

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

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

So far, only the output of a single neuron has been discussed. However, ANNs consist not only of one neuron but combine multiple neurons in a network. These neurons are organised 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 (FC) layer. In a fully connected layer, the input $\boldsymbol{x}$ is fed into all neurons to obtain $\hat{\boldsymbol{y}}$. If the layer has k neurons, the aggregation function is calculated for each neuron and the aggregated value becomes a vector $\boldsymbol{z} = (z_1, \ldots, z_k)$. The same applies to the output of the activation function $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_k)$ and the bias $\boldsymbol{b} = (b_1, \ldots, b_k)$. The weight vector, on the other hand, becomes a matrix $\boldsymbol{W} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k)$. For a fully connected layer, the aggregation and activation functions can be rewritten with matrix operations:

$$\boldsymbol{z} = \boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b} \tag{2.6}$$

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

which is equal to

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \\ \ldots \\ z_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_1 \cdot \boldsymbol{x} + b_1 \\ \boldsymbol{w}_2 \cdot \boldsymbol{x} + b_2 \\ \ldots \\ \boldsymbol{w}_k \cdot \boldsymbol{x} + b_k \end{bmatrix} = \begin{bmatrix} \left(\sum_{i=1}^{n} w_{1i} \cdot x_i\right) + b_1 \\ \left(\sum_{i=1}^{n} w_{2i} \cdot x_i\right) + b_2 \\ \ldots \\ \left(\sum_{i=1}^{n} w_{ki} \cdot x_i\right) + b_k \end{bmatrix} \tag{2.8}$$

$$\hat{\boldsymbol{y}} = \begin{bmatrix} y_1 = f(z_1) \\ y_2 = f(z_2) \\ \dots \\ y_k = f(z_k) \end{bmatrix} \tag{2.9}$$

[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 multi-layer perceptron (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 $L > 1$ 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 $\boldsymbol{a}^{[l]}$[2]. 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

2: by convention, $\hat{\boldsymbol{y}}$ is used only as model output and not as layer output. Therefore, outputs of intermediate layers are denoted as $\boldsymbol{a}^{[l]}$ hereafter.

$$\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]} \tag{2.10}$$

$$\boldsymbol{a}^{[l]} = f(\boldsymbol{z}^{[l]}) \tag{2.11}$$

So far, only the forward pass used to calculate the model's 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 be used to calculate the goodness of the model output $\hat{\boldsymbol{y}}$ compared to the target output $\boldsymbol{y}$. 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)[3] 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

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

of the loss value points to the direction of the steepest descent (i.e. in the direction where the loss becomes smaller). Therefore, SGD updates the network parameters by taking a step of size η[4] toward their negative gradient:

4: η is a hyper-parameter and also referred to as learning rate

$$\begin{aligned} \Delta \boldsymbol{W}^{[l]} &= -\eta \cdot (\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}) \\ \boldsymbol{W}^{[l]} &:= \boldsymbol{W}^{[l]} + \Delta \boldsymbol{W}^{[l]} \end{aligned} \tag{2.12}$$

and

$$\begin{aligned} \Delta \boldsymbol{b}^{[l]} &= -\eta \cdot (\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}) \\ \boldsymbol{b}^{[l]} &:= \boldsymbol{b}^{[l]} + \Delta \boldsymbol{b}^{[l]} \end{aligned} \tag{2.13}$$

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[5].

[11]: Rumelhart et al. (1986)

5: While a detailed discussion of 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 each neuron of one layer is connected to every neuron of the following layer (fully connected layer) has been described. Besides such dense connections, there are several alternatives. Since this work deals with the processing of images, the most well-known image-processing architecture called convolutional neural network (CNN), is presented in the following. This architecture is also primarily used in this thesis. However, various alternative architectures exist for computer vision, such as the vision transformer (ViT) [13] or MLP mixer [14]. However, describing these architectures 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 fully connected networks (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 the input and create translation-equivariant[6] [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, and the process is repeated 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

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

[15]: Mouton et al. (2020)

[16]: Zhang et al. (1990)

different input locations is 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, using multiple layers sequentially continuously enlarges the receptive field[7].

7: the area of input pixels that can influence a single value in the feature map of a layer

Pooling layers, on the other hand, do not extract features bit reduce the size of the input. Similar to convolutional layers, a filter slides along the input. However, this filter has no learned parameters 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[8]. 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)

8: 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. (2015)

[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. However, 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 "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, e.g. 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.

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

one or multiple labels per image) [18–22], object detection requires the coordinates of the bounding box [24–26], and semantic segmentation requires the labels on pixel-level [23, 28] (i.e. each pixel has a label of a class 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 by Simmler et al. [29]. Not only specific tasks can be learned, but also task-independent representations. These representations are typically learned unsupervised[9] and can be used for one or several downstream tasks. More details on visual representation learning are provided in Section 3.4. Especially autoencoders [30] are explained in this section as they are applied in this thesis.

[29]: Simmler et al. (2021)

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

[30]: Rumelhart et al. (1987)

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 in recent years, 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 (1965)

[32]: Kumar (2015)

[33]: Open AI (2018)

Many recent improvements in deep learning are due to massively growing model and dataset 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 called ELMo [34] had around 94M parameters, the state-of-the-art model in 2020 called GPT-3 [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 called Megatron-Turing NLG 530B [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

[34]: Peters et al. (2018)

[35]: Brown et al. (2020)

[36]: Lambda Inc. (2021)

[37]: Smith et al. (2022)

[38]: Wu et al. (2020)

pruning [39], or model distillation [40], it is questionable if making models bigger is the best way to develop more intelligent systems.

[39]: Choudhary et al. (2020)

[40]: Hinton et al. (2015)

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 datasets 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 even if a new task is learned. Catastrophic forgetting is also caused by the fact that learning is mostly done offline[10]. 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 only adapt better to changing conditions.

[41]: Zhang et al. (2021)

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

[42]: Sahoo et al. (2018)

[43]: Parisi et al. (2019)

Furthermore, some problems may not be solvable with the current principles of deep learning. First of all, it is questionable if deep learning models can achieve *real* generalization[11]. 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].

11: 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[12].

[45]: Marcus (2018)

12: some experiments by the author suggest that large language models might overcome this limitation and can learn such definitions in a one-shot manner. However, these models still lack understanding.

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 do not work well for processing high-dimensional data.

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[13]. 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 the models (called agents) 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

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

[46]: Moravec (1995)

perception and mobility". This statement still seems true almost 30 years later.

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[14]. These architectures are not divided into many different sections. Instead, deep learning architectures tend to be monoliths, i.e. one large unit of layers without well-defined 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 intelligence and has been learned over time through evolution.

14: 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 2.2) is the state-of-the-art algorithm that solves this problem by propagating the error signals back through the network. However, information in

the brain flows 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, 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.

Lastly, the neurons in the brain are self-organising. Self-organisation is the process 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[15]. 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.

15: unlike deep learning systems that are updated by a global optimisation algorithm that minimises an objective function

2.5 Neurocomputing

Neurocomputing is considered a subfield of neuroscience and focuses on implementing biologically more plausible learning algorithms. Therefore, the abovementioned discrepancy is 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)

Hebbian learning 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[16]

$$\Delta w_{ij} = \eta a_i a_j \tag{2.14}$$

16: 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

where η is the learning rate. Thus, the weights between frequently co-activated neurons increase; this process is 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 usually extended to leverage the covariance of neuronal activity. 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:

$$\Delta w_{ij} = \eta(a_i - \psi_i) \cdot (a_j - \psi_j) \tag{2.15}$$

where ψ_i and ψ_j are estimates of the expected pre- and post-synaptic activity[17]. 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 is to normalize the length of the weight vector [49].

17: the expected activity can be estimated, for example, by calculating a moving average

[49]: Oja (1982)

$$\Delta w_{ij} = \eta \left(a_i a_j - \alpha \cdot a_j^2 w_{ij} \right) \tag{2.16}$$

where $\alpha \cdot a_j^2 w_{ij}$ is the regularization term, and 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

$$||\boldsymbol{w}||^2 = \frac{1}{\alpha} \tag{2.17}$$

Another measure to limit the size of the vector $\boldsymbol{w}$ is to use 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)

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)[18] 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].

[52]: Simoncelli et al. (2001)

18: 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

[56]: Grossberg (1988)

combination of supervised and unsupervised Hebbian learning.

2.5.2 Hopfield Networks

[57]: Hopfield (1982)

[58]: Fix et al. (1989)

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

[59]: Weston et al. (2015)

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[19]. 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 $\boldsymbol{h}$ by feeding it through an encoder $E(\cdot)$:

$$\boldsymbol{h} = E(\boldsymbol{x}) \tag{2.18}$$

Afterwards, the memory $\boldsymbol{M}$ is updated by a memory-update function $G(\cdot)$ given the internal feature representation $\boldsymbol{h}$ of the input $\boldsymbol{x}$.

$$\boldsymbol{M} = G(\boldsymbol{M}, \boldsymbol{h}) \tag{2.19}$$

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

$$\hat{\boldsymbol{y}} = O(\boldsymbol{M}, \boldsymbol{h}) \tag{2.20}$$

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:

$$a_i = \sum_{i \neq j} w_{ij} \hat{y}_j + b \tag{2.21}$$

$$\hat{y}_i = \begin{cases} 1, & \text{if } a_i > 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.22}$$

Hopfield networks have their own dynamics, and the output evolves over time. If a binary Hopfield network's initial value $\hat{y}_i$ has a different sign than a_i, the output flips (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 $\hat{y}_i \cdot a_i$ is negative if $\hat{y}_i$ is not equal to a_i, otherwise, it is positive. Since the neuron flips if the term $\hat{y}_i \cdot a_i$ is negative or stays the same if this term is

positive, the change of this term can only be positive:

$$\Delta\left(\hat{y}_i \cdot a_i\right) \geq 0 \tag{2.23}$$

The negative sum of the term $\hat{y}_i \cdot a_i$ for the entire network is called the energy function $I(\hat{\boldsymbol{y}})$[20] of the network:

20: 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.

$$I(\hat{\boldsymbol{y}}) = -\sum_i \hat{y}_i \overbrace{\left(\sum_{j>i} w_{ji}\hat{y}_j + b\right)}^{a_i} \tag{2.24}$$

As shown in equation (2.23), the network energy function $I(\hat{\boldsymbol{y}})$ 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 $\hat{\boldsymbol{y}}$ accordingly. Afterwards, the binary neurons flip until a point attractor is reached. Thereby, an input 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.

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[21]:

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

$$\boldsymbol{w} = \frac{1}{P}\sum_{k=1}^{P} \boldsymbol{y}^{(k)} \times (\boldsymbol{y}^{(k)})^T - \boldsymbol{I} \tag{2.25}$$

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 [60].

[60]: Hopfield et al. (1983)

One of the limiting factors of binary Hopfield networks is the capacity C, i.e. the number of patterns that can be stored. A binary Hopfield network with N neurons has a capacity of $C = 0.138N$[61]. 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 binary Hopfield networks have an exponential capacity of $C \approx 2^{\frac{N}{2}}$.

[61]: McEliece et al. (1987)

[62]: Krotov et al. (2016)

[63]: Demircigil et al. (2017)

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

[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 the k-nearest neighbour (k-NN) algorithm [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) and precise timing of the spikes are relevant. The amplitude and duration of the spike only matter 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[22]. The neurons fire 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.

22: 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[23]. 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 them.

23: 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. However, only some systems are suited as a reservoir: A good reservoir system distributes different inputs into different regions of the computation space [77]. An ESN is a set of sparsely

[75]: Jäger (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].

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[24] [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 $\boldsymbol{W}_{\text{in}} \in \mathbb{R}^{N \times M}$. The output is calculated by multiplying the reservoir state with another weight matrix $\boldsymbol{W}_{\text{out}} \in \mathbb{R}^{M \times N}$. Interested readers are referred to [80] to learn more about the mathematical properties and how the network is updated in detail.

24: the Erdős–Rényi model is a model for generating random graphs where all graphs based 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. Afterwards, the recurrent connections inside the reservoir create different dynamics in the network. The readout neurons linearly transform the recurrent dynamics into temporal outputs. Thereby, readout weights $\boldsymbol{W}_{\text{out}}$ are trained to reproduce a target function $\boldsymbol{y}[t]$.

Liquid State machines, on the other hand, 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 other nodes. 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. In principle, the system should be capable of any computation if it has a high enough complexity [77]. Furthermore, reservoirs generalise better and faster than equivalent MLP. Current systems' main drawback is that they cannot deal well with high-dimensional input data such as images.

Related Work 3

This chapter presents the related work of the thesis. Section 3.1 gives an introduction to a publication entitled "A Theory of Natural Intelligence" by von der Malsburg et al. [2]. This work is the main source of inspiration and is therefore summarised in detail. It identifies various concepts as the fundamentals of intelligence, including self-organisation. Therefore, the work associated with self-organisation is reviewed in Section 3.2. Self-organisation is not compatible with end-to-end backpropagation of error. Thus, alternative training algorithms are presented in Section 3.3. Then, Section 3.4 reviews different methods for learning image representations. Finally, meta-learning is summarised in Section 3.5. Meta-learning is of limited relevance to this thesis but is important for the proposed future work.

3.1 Natural Intelligence 21
3.2 Self-Organisation 23
Growing Networks 24
Spiking Networks 25
Relevance 26
3.3 Learning Algorithms 26
Proxy Objective Functions . . 27
Target Propagation 28
Other Methods 30
Relevance 31
3.4 Representation Learning . . . 31
Autoencoders 31
Self-Supervised Learning . . 32
Relevance 33
3.5 Meta-Learning 33
Meta-Representation 33
Meta-Optimiser 34
Meta-Objective 35

3.1 Natural Intelligence

According to von der Malsburg et al. [2], the process of learning is influenced by "nature", "nurture", and "emergence"[1]. They point out that the human genome (as of nature) only contains 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 the human brain. Thus, there is a massive gap between the amount of information needed to describe the brain's structure and the amount of information needed to generate it. Therefore, they argue that the human brain must be highly structured and that the processes of nature and nurture construct the human brain by selecting from a set of pre-structured patterns. The authors call the process of generating the highly structured network in the human brain the "Kolmogorov Algorithm [82] of the Brain"[2]. Network self-organisation is the only mechanism that experiments have not yet disproved 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 implies 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-organisation has to start from an already established coarse global structure which is improved in a coarse-to-fine manner to avoid being caught in local optima.

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

1: nature refers to the influence of genes and evolution, nurture to the influence of experience and education, and emergence to the construction of network structures

[81]: McPherson et al. (2001)

[82]: Kolmogorov (1998)

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

[83]: Willshaw et al. (1976)

[84]: Willshaw et al. (1979)

[85]: von der Malsburg et al. (1987)

[86]: von der Malsburg (2018)

Also, von der Malsburg et al. [2] discuss scene representation (i.e. how a scene is represented in the brain) even though they point out that this is a contested concept [87]. Scene representation is an organisational framework to put abstract interpretations of scene layouts, elements,

[87]: Freeman et al. (1990)

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 primary goal of learning is to integrate a behavioural schema into the flow of scene representations. They hypothesise that the network structure resulting from self-organisation and the neural activation in the framework of scene representation is 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 network self-organisation groups neurons in one or multiple net fragments. The net fragments can be considered 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 comprise net fragments that can be formed by network self-organisation [90]. They also argue that net fragments are the compositional data structure used by the brain. Nested net fragments of different sizes may represent a hierarchy of features. Complex objects, such as mental constructs, can thus be seen as larger net fragments composed of mergers of pre-existing smaller net fragments.

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

3: adapt despite deformations, depth rotation, and position

[88]: Arathorn (2002)

[89]: Olshausen et al. (1995)

[90]: Fernandes et al. (2015)

Von der Malsburg et al. [2] have not addressed how their interesting theoretical concepts can be implemented as a mathematical model. However, a concrete implementation was proposed 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 2.5.1). Convolutional filters calculate multiple feature maps. 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 spatially close locations, their connection strength increases. Thus, Hebbian learning forms lateral connections between the feature maps with a high covariance. This process forms new filters based on existing filters. Lehmann found that LCL layers increase robustness for object recognition. He shows on the MNIST dataset [18] that for a slight reduction in accuracy of 1%, the performance on corrupted images increases by up to 21%. However, the proposed layer has only improved performance on small networks, and its effectiveness on larger networks has yet to be demonstrated.

[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, their 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-Organisation

Self-organisation is the process by which systems consisting of many units spontaneously acquire their structure or function without interference from an external agent or system. They organise their global behaviour through local interactions amongst themselves. The absence of a central control unit allows self-organising systems to adjust to new environmental conditions quickly. Additionally, such systems have built-in redundancy with a high degree of robustness as they consist of many simpler individual units. These individual units can even fail without the overall system breaking down.

In nature, groups of thousands of units that solve complex tasks using only local interactions can be observed. For example, ants can navigate rugged terrain with local pheromone-based communication and thus form a collective type of intelligence. Such observations inspired researchers to build algorithms based on local communication and self-organisation, for example, ant colony optimisation algorithms [92]. DeepSwarm [93] is a neural architecture search method that uses this algorithm to search for the best neural architecture. This method achieves competitive performance on relatively 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 updated periodically. Update rules define the transition from a given state to a subsequent state. During an update, cells can only communicate with the neighbouring cells. Thus, self-organisation 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, and the output is the following cell state. Usually, the same network is applied to all cells. In this case, a fully connected neural network 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 colour based on the colour of neighbouring pixels) and are surprisingly resistant to damage. For example, large parts of the images can be removed, and the system can still rebuild the entire image[4]. However, the aforementioned approaches can only grow the pattern they were trained on. A recent method called Variational Neural Cellular Automata [102] use an NCA as the decoder of a variational autoencoder [103]. This probabilistic generative model can grow images based on a vector sampled from a Gaussian distribution. 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 utilising 3D CNNs [104] or generate structures with a specific function such as simulated robots able to locomote [105]. Moreover, self-assembling approaches based on NCAs are not restricted to grid structures. NCAs can be generalised to graph neural networks [106]. Graph cellular automata (GCA) use graph neural networks [107] instead of CNNs to learn the state transition rules and can thus deal with more complex pattern structures than just 2D

[96]: Wolfram (1984)

[97]: Vichniac (1984)

[98]: Wulff et al. (1992)

[99]: Gilpin (2019)

[100]: Mordvintsev et al. (2020)

[101]: Mordvintsev et al. (2020)

4: a demo of this regeneration process is available at [100]

[102]: Palm et al. (2022)

[103]: Kingma et al. (2014)

[104]: Sudhakaran et al. (2021)

[105]: Horibe et al. (2021)

[106]: Grattarola et al. (2021)

[107]: Zhou et al. (2020)

and 3D grids. The process of growing images from cells of an NCA can also be inverted. Randazzo et al. [108] propose to use NCA to classify given structures such as images. They apply the same network to each pixel and its neighbours of an image. In an iterative process based on local communication, the image fragments agree on which object they represent. NCAs can even be used to control reinforcement learning (RL) agents. Variengien et al. [109] use the observations of the environment as cell states of the NCA. The following states predicted by the NCA are used as Q-value estimates of a deep Q-learning algorithm [110].

[108]: Randazzo et al. (2020)

[109]: Variengien et al. (2021)

[110]: Mnih et al. (2013)

Self-organisation can not only be used to generate structures, but also to optimise the weights of neural networks. For example, self-organisation can be used to meta-learn (c.f. Section 3.5) a Hebbian learning rule that better adapts to changing conditions in the environment [111]: The weights of an RL agent's policy are reset in each episode, and a Hebbian-based learning rule is used to learn them from scratch. The Hebbian learning rule improves over episodes thanks to self-organisation, allowing the agent to learn more robust weights. The method "evolve and merge" [112] implements this principle by using the so called "ABCD" Hebbian learning rule which updates the weights as follows:

[111]: Najarro et al. (2020)

[112]: Pedersen et al. (2021)

$$\Delta w_{ij} = \alpha(Aa_i a_j + Ba_i + Ca_j + D) \tag{3.1}$$

α is the learning rate, a_i and a_j are the activity levels of connected neurons and A, B, C, and D are learned parameters. For each connection in the network, the four parameters are learned. After a pre-defined number of epochs, the learning rules are clustered, and the ones with similar parameters are merged. By repeating this process, the number of parameters can be reduced, and robustness increases further.

Alternatively, it is also possible to initialise the network with shared parameters instead of starting with many rules and merging them over time. Kirsch and Schmidhuber [113] use multiple tiny recurrent neural networks (RNNs) that have the same weight parameters but different internal states[5]. By using self-organisation and Hebbian learning, they show that it is possible to learn powerful learning algorithms such as backpropagation while running the network in forward mode only. However, it works only for small-scale problems as it can quickly get stuck in local optima. In general, self-organising systems are hard to optimise and only work for small datasets or simple problems so far.

[113]: Kirsch et al. (2021)

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

Risi [114] describes why self-organising systems are hard to train; First, the system is hard to control because no central entity is in charge, but the system must still be nudged in the right direction. Second, self-organising systems are unpredictable (i.e., no mathematical model tells the outcome of the self-organising process). Therefore, he concludes that training such systems is much more complex than training a system with central supervision, such as end-to-end backpropagation of error.

[114]: Risi (2021)

3.2.1 Growing Networks

Unsupervised learning techniques usually map high-dimensional input data to a lower-dimensional representation. One approach implementing such a mapping is based on self-organising maps (SOM) [115, 116]. SOMs

[115]: Kohonen (1982)

[116]: Kohonen (1989)

map the input data to a discrete representation space of the training samples called a map. Unlike ANNs, they use competitive learning instead of error correction learning (i.e. backpropagation or error). A weight vector maps the data to a node in the mapping field. The data points "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 in the neighbourhood. By doing so, samples that are close in the input space are also closed in the resulting maps.

However, SOMs have two significant limitations; First, the network structure must be pre-defined, limiting the mapping accuracy. Second, the map's capacity is pre-defined through the number of nodes. Growing networks can overcome these limitations: Growing networks can 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 [117–119] add such units after a fixed number of iterations during 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.

[117]: Fritzke (1994)

[118]: Reilly et al. (1982)

[119]: Fritzke (1994)

Grow When Required (GWR) networks [120] 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 sample cannot be matched with the current nodes by some pre-defined accuracy. This allows the network to adapt to the growing process rather fast; The network stops growing when the inputs can be matched with existing latent representations with a good-enough accuracy, and the network starts growing again if the input distribution changes.

[120]: Marsland et al. (2002)

Such GWR networks can be used to build self-organising architectures. For example, Mici et al. [121] build a self-organising architecture based on GWR to learn human-object interactions from videos. They use two GWRs in parallel, one to process feature representations of body postures and another to process manipulated objects. A third GWR combines these two streams and creates action–object mappings in a self-organised manner. By doing so, they can learn human-object interactions and exhibit a model more robust to unseen samples than comparable deep learning approaches.

[121]: Mici et al. (2018)

3.2.2 Spiking 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 [122]. There exist several methods to self-organise such architectures. For completeness, two well-known approaches are described in the following. However, since this thesis focuses on the self-organisation of deep learning systems without time-dependent spikes, these approaches are only roughly described, and interested readers may refer to the respective literature.

[122]: Davies et al. (2018)

Similar to deep learning, there exists a multitude of different spiking network architectures; Shallow [123, 124] and deep [74, 125] network structures, networks based on fully connected layers [126] or convolutional layers [127, 128], as well as networks that are trained with different learning principles such as supervised [129, 130], unsupervised [126, 131] and reinforcement learning [132].

[123]: Masquelier et al. (2007)

[124]: Yu et al. (2013)

[74]: Kheradpisheh et al. (2018)

[125]: Mozafari et al. (2019)

[127]: Cao et al. (2015)

[128]: Tavanaei et al. (2017)

[129]: Diehl et al. (2015)

[133]: Raghavan et al. (2020)

One type of methodology for self-organising SNNs is based on a stackable toolkit to assemble multi-layer neural networks [133]. This toolkit is a sub-system that encapsulates the dynamics of spiking neurons, their interactions, and the plasticity rules that control the flow of information between layers. A model consists of multiple sub-models created by this toolkit. Based on the input, spatiotemporal waves are generated that travel across the sub-systems. A dynamic learning rule tunes the connectivity between sub-systems based on the properties of the waves [6].

6: for more information, please refer to Raghavan et al. [133]

[134]: Raghavan et al. (2019)

An alternative method is to use self-organisation to grow spiking networks [134]; The growing process starts with a single computational "cell", and a wiring algorithm is used to generate a pooling architecture in a self-organising fashion. The pooling architecture emerges through two processes; First, a network layer is grown. Second, self-organisation of its inter-layer connections forms receptive fields. The first layer uses the Izikhevich neuron model [70] to generate spatiotemporal waves. The neurons in the following layers 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].

[70]: Izhikevich (2003)

7: for more information, please refer to [134]

In general, SNNs must "convert" static input data such as images to a dynamic signal. However, such approaches lose a lot of information about the input. For example, all information about colour and thin structures is lost if DoG filters (c.f. Section 2.5.3) are used to generate spikes. Currently, this seems to be one of the limiting factors why such SNNs cannot match the performance of deep learning algorithms and often only work well for small grey-scale image datasets such as MNIST [18].

3.2.3 Relevance

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

Self-organisation is, according to von der Malsburg et al. [2], one of the fundamental principles for natural intelligence. Therefore, this concept of local optimisation and interaction plays an essential role in this thesis. Specifically, two types of self-organisation are proposed in this thesis; Horizontal self-organisation described in Chapter 5 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-organisation proposed in this thesis is vertical self-organisation. With this approach, data is divided into patches and analysed by independent networks. Thus, small networks form self-organising units and communicate with each other by adding the output of one network to the input of another network. Vertical self-organisation is described in Chapter 6.

3.3 Learning Algorithms

Neural networks are usually optimised with backpropagation of errors (c.f. Section 2.2). Soon after backpropagation was published, it was questioned whether this algorithm is suitable to explain learning in the

brain [135, 136]. Besides the fact that backpropagation seems biologically not plausible and might be responsible for many of the limitations described in Section 2.3, it also has technical shortcomings: First, gradients might vanish or explode when propagating through too many layers [137]. Second, it turns the networks into "black boxes" and prevents users from getting valuable insights from trained models[8]. Training end-to-end does not allow controlling the loss function's effect on a hidden layer because of the non-linearity in the network [138]. Finally, the loss landscapes are non-optimal, which might lead to slow convergence, or the optimisation process can get stuck in a local minimum [17].

[135]: Crick (1989)

[136]: Grossberg (1987)

[137]: Zhang et al. (2020)

8: training a model layer-wise allows modularisation, i.e., dividing and conquering

[138]: Lee et al. (2015)

[17]: Ioffe et al. (2015)

Hebbian learning (c.f. Section 2.5.1), on the other hand, is widely accepted in the fields of neurocomputing, psychology, neurology, and neurobiology [139]. Hebbian learning works well, for example, for topographic mappings[9] [116, 140], neuroplasticity[10] [141], or recognising non-linear patterns [142], but also has limitations [143]. Hebb's learning rule is insufficient as a general rule for time-shifted signals as it requires (almost) synchronous stimuli [143, 144]. However, many signals, such as motor control problems, are sequential and Hebbian learning has to be combined with additional memory mechanisms [145].

[139]: Widrow et al. (2019)

9: mapping input data to a discrete representation

[116]: Kohonen (1989)

[140]: Grajski et al. (1990)

10: the networks ability to reorganise itself by forming new connections

[141]: Montague et al. (1991)

[142]: Mel (1992)

[143]: Anderson (1998)

[144]: Rumelhart et al. (1987)

[145]: Grossberg et al. (1989)

3.3.1 Proxy Objective Functions

Many alternative learning functions exist to these two well-known algorithms that might overcome the aforementioned limitations. Local optimisation without end-to-end backward passes can be implemented with *proxy objective functions*. Using proxy objective functions means having a loss function for each layer instead of a global loss function for the entire model. The loss function of the first layer (i.e. the proxy objective of the first layer) influences only the trainable parameters of the first layer but not the parameters of the second layer. Thus, proxy objective functions allow decoupling layers. The different instances of this method mainly differ in the details of the proxy objective function's definition. However, the aim is always the same: improving the separability of the hidden representations according to some measure. Minimising the proxy objective function encourages a layer to improve the separability of its output representations, making the separability problem simpler for the subsequent layer. A simple way to measure the separability between hidden representation vectors is to use distance or similarity metrics [146, 147]. Instead of measuring the distance between hidden representations directly, an auxiliary neural network can be used to first map the hidden representations to another feature space before calculating their distance [148]. This allows to reduce the dimensionality of the feature space before calculating the proxy objective function. Reducing the dimensionality is especially helpful for layers with large output vectors, such as the activation maps of convolutional layers.

[146]: Duan et al. (2022)

[147]: Duan et al. (2020)

[148]: Wang et al. (2021)

An auxiliary network can also be used to classify the latent representations in each layer and to use this auxiliary network's accuracy to quantify data separability [149–151]. Nokland and Eidnes [152] combine the two aforementioned approaches by using two auxiliary networks; one auxiliary network reduced the dimensionality and another auxiliary network quantifies similarity.

[149]: Belilovsky et al. (2019)

[150]: Mostafa et al. (2018)

[151]: Marquez et al. (2018)

[152]: Nokland et al. (2019)

A significant issue of models based on proxy objective functions is that they are usually trained layer-wise[11]. The resulting computational inefficiency can be reduced with synchronous or asynchronous training. In the synchronous setting, a forward pass through all layers is done, and afterwards, all layers are trained simultaneously by minimising their own proxy objective function [149]. In the asynchronous setting, all layers are trained simultaneously by using a replay buffer for each layer that stores the layer's output [153]. Each layer receives input from such a buffer instead of the previous layer, which eliminates the need for an end-to-end forward pass.

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

[153]: Belilovsky et al. (2020)

Recently, Hinton [154] introduced the *forward-forward (FF) algorithm*. This algorithm uses two forward passes, one with positive (i.e. real) data and one with negative data. Each layer has its objective function that aims to have a high "goodness" for positive data and a low "goodness" for negative data. The sum of the squared neural activities measures the goodness. Thus, having many (strongly) active neurons means having high goodness. The goal of the learning process is to push the goodness above a threshold for real data and below that threshold for negative data. The probability that an input vector is positive $P(\text{positive})$ is given by applying the logistic function σ to the goodness minus the threshold θ

[154]: Hinton (2022)

$$P(\text{positive}) = \sigma\left(\sum_j \boldsymbol{a}_j^2 - \theta\right) \tag{3.2}$$

where $\boldsymbol{a}_j$ is the output of a hidden unit. For supervised learning, the label must be included in the model's input. Positive data consists of an input vector (e.g. an image) and the correct label, and negative data consists of an input vector with the wrong label. The goal is 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 the prediction. One major weakness is that one layer at a time is learned and later layers cannot influence what is learned in earlier layers. This limitation can be overcome by treating the static input vector as a sequence (i.e. using the same input for multiple timesteps) and using a multi-layer recurrent network [155]. The trick is to feed the input into the network from one side and the label into the network from the other side. A layer receives as input the activity vector of the previous layer and the activity vector of the subsequent layer at the previous timestep. By enforcing high goodness when the input and label match, information must flow in both directions. Thus, hierarchical features can be learned without a backward pass or layer-wise training.

[155]: Hinton (2021)

3.3.2 Target Propagation

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[12] rather than gradients at each layer. Similar to gradients, the targets are propagated backwards through the network. Each layer is trained to minimise the difference between its activations and its received target values. While the output layer obviously uses the true label as its target (i.e. the goal is to predict the true label), the targets of previous layers are found sequentially (from the output layer to the input layer). Good targets are those that minimise the loss in the output layer if they are realised in the forward pass. Intuitively, each layer tells the previous layer the best possible input.

12: a target can be interpreted as the value that a layer should predict

Let's define $f^{[l]}(\cdot)$ as a layer's function, i.e. layer l calculates its activation $\hat{\boldsymbol{a}}^{[l]}$ based on the output of the previous layer $l-1$ as

$$\hat{\boldsymbol{a}}^{[l]} = f^{[l]}(\hat{\boldsymbol{a}}^{[l-1]}) \tag{3.3}$$

In the context of target propagation, $\hat{\boldsymbol{a}}^{[l]}$ is the actual layer activation and $\boldsymbol{a}^{[l]}$ the desired layer activation (i.e. the target). The output of a network with L layers is the prediction of layer L, i.e. $\hat{\boldsymbol{a}}^{[L]} = \hat{\boldsymbol{y}}$. If for each layer an inverse function $\hat{\boldsymbol{a}}^{[l-1]} = (f^{[l]})^{-1}(\hat{\boldsymbol{a}}^{[l]})$ would exist, a layer's input could be calculated based on a layer's output. This idea is used by target propagation: During the forward pass, a prediction $\hat{\boldsymbol{y}}$ is calculated. However, the goal is to calculate the correct label distribution $\boldsymbol{y}$. Therefore, $\boldsymbol{y}$ is fed through the inverse function of each layer $(f^{[l]})^{-1}(\cdot)$ to calculate a target for the previous layer. Thus, if each layer predicted this target as output, the network's output would be the correct label distribution $\boldsymbol{y}$. However, advanced neural networks are usually not invertible, and approximate inverse transformations have to be learned with decoders $D^{[l]}$

$$D^{[l]}(\hat{\boldsymbol{a}}^{[l]}) \approx (f^{[l]})^{-1}(\hat{\boldsymbol{a}}^{[l]}) \tag{3.4}$$

Thus, the target activation can be approximated by a decoder:

$$\boldsymbol{a}^{[l]} \approx D^{[l+1]}(\boldsymbol{a}^{[l+1]}) \tag{3.5}$$

Different instances of target propagation methods mainly differ in the way the targets are generated or the loss function of the decoder $\mathcal{L}_D$ is calculated. Vanilla target propagation [156] directly derives the target from the decoder $\boldsymbol{a}^{[l]} = D^{[l+1]}(\boldsymbol{a}^{[l+1]})$ and trains the decoder $D^{[l]}(\cdot)$ by minimizing

[156]: Bengio (2014)

$$\mathcal{L}_d^{[l]} = \|D^{[l+1]}(\hat{\boldsymbol{a}}^{[l+1]} + \epsilon) - (\hat{\boldsymbol{a}}^{[l]} + \epsilon)\|_2^2 \tag{3.6}$$

where ϵ is some added Gaussian noise to enhance generalisation. It is obvious that this loss function encourages the decoder $D^{[l]}(\cdot)$ to learn the inverse of $f^{[l]}(\cdot)$ by minimising the difference between the activations from the forward-pass $\hat{\boldsymbol{a}}^{[l]}$ of layer l and the decoder's prediction of $\hat{\boldsymbol{a}}^{[l]}$ (calculated based on the activations of the subsequent layer $\hat{\boldsymbol{a}}^{[l+1]}$).

The targets are predicted by feeding the target activations $\boldsymbol{a}^{[l+1]}$ of the subsequent layer through the decoder[13].

13: note that these are two different kinds of activations: The inverse function is learned based on the activation of the forward pass $\hat{\boldsymbol{a}}^{[l+1]}$, the target activation is predicted based on the loss-optimal output activation $\boldsymbol{a}^{[l+1]}$

A major improvement of vanilla target propagation is to extend the decoder $D(\cdot)$ with a correction term that enables a more robust optimality guarantee for bigger networks [157]. This method is known as *difference target propagation* and extends Equation (3.5) as follows:

[157]: Lee et al. (2015)

$$\boldsymbol{a}^{[l]} \approx D^{[l+1]}(\boldsymbol{a}^{[l+1]}) + \left[\hat{\boldsymbol{a}}^{[l]} - D^{[l+1]}\left(\hat{\boldsymbol{a}}^{[l+1]}\right)\right] \tag{3.7}$$

The additional term $\left[\hat{\boldsymbol{a}}^{[l]} - D^{[l+1]}\left(\hat{\boldsymbol{a}}^{[l+1]}\right)\right]$ is to correct errors of the decoder in estimating the inverse. Difference target propagation has been further improved by novel loss functions for the decoder. These methods are known as *difference target propagation with difference reconstruction loss* and *direct difference target propagation* [158]. An advantage of calculating targets is that only layer-wise gradients are required, which allows models to have non-differentiable operations [157]. A significant drawback is that auxiliary decoders must be trained to approximate the hidden targets $\boldsymbol{a}^{[l]}$, which might outweigh the savings achieved by eliminating a complete backward pass.

[158]: Meulemans et al. (2020)

3.3.3 Other Methods

Synthetic gradients methods [159, 160] replace the gradients of backward passes with approximated local gradients with auxiliary models and utilise gradient-based optimisation algorithms such as stochastic gradient descent 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 backward passes are only performed occasionally to acquire actual gradients that can be used to train the gradient approximation models. Thus, the frequency of end-to-end backward passes is reduced. When additional auxiliary input models are used to predict the layer's input, the frequency of the forward pass can be reduced as well in the same manner as for the backward pass [159]. It is also possible to eliminate the backward pass by training the auxiliary gradient models with local information only [161]. However, this works significantly worse than when actual gradients are used to train auxiliary models. An advantage of synthetic gradient methods is that they can approximate backpropagation through time (BTT) for an unlimited number of timesteps [159]. It has been shown that this allows more efficient training for learning long-range dependencies compared to BTT.

[159]: Jaderberg et al. (2017)

[160]: Czarnecki et al. (2017)

[161]: Lansdell et al. (2020)

Many other methods are motivated purely by biological plausibility [162–166]. However, these biologically plausible methods have been significantly outperformed by backpropagation of error on meaningful benchmark datasets [167]. Probably the most popular group of biologically plausible alternatives to backpropagation are methods based on *feedback alignment*. From a biological point of view, one of the major criticisms of backpropagation is the "weight transportation problem" [163]; it is believed that the human brain does not have a backward pass that passes an error signal to previous layers. *Feedback alignment* algorithms use

[162]: Nokland (2016)

[163]: Lillicrap et al. (2016)

[164]: Xiao et al. (2019)

[165]: Balduzzi et al. (2015)

[166]: Liao et al. (2016)

[167]: Bartunov et al. (2018)

[163]: Lillicrap et al. (2016)

fixed, random weights during the backward pass [163]. The gradients are calculated similarly to stochastic gradient descent but with random weights. Thus the symmetry between the weights used during the forward and the backward pass is broken. The network still learns to make the feedback useful even though the weights during forward and backward passes are different. Later, the algorithm was improved by using fixed, random weights that share the signs with the actual weights of the network [166].

[163]: Lillicrap et al. (2016)

[166]: Liao et al. (2016)

Lastly, another group of learning algorithms use *auxiliary variables* [168–171]. These methods use the idea of variable splitting, i.e. transforming 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 usually scale to specific use cases only.

[168]: Carreira-Perpinan et al. (2014)

[169]: Taylor et al. (2016)

[170]: Zhang et al. (2017)

[171]: Lau et al. (2018)

3.3.4 Relevance

End-to-end backpropagation is incompatible with local self-organisation, which is one of the fundamental 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 to organise themselves. Therefore, alternative learning algorithms are very relevant to this thesis. The approach based on horizontal self-organisation uses proxy objective functions to train the layer locally as independent units (c.f. Chapter 5). Proxy objective functions are used in this thesis as this method is the only known alternative to backpropagation of error that scales well to large neural networks and large datasets [167, 172].

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

[167]: Bartunov et al. (2018)

[172]: Duan et al. (2022)

3.4 Representation Learning

3.4.1 Autoencoders

Autoencoders are one of the most-used methods to learn visual representations in an unsupervised fashion. Autoencoders were introduced in 1985 [30] and learn an encoder and decoder function [173]. The encoder E maps the input from a high-dimensional space to a lower-dimensional embedding space $E : \mathbb{R}^n \rightarrow \mathbb{R}^e$ and the decoder D reverses this mapping $D : \mathbb{R}^e \rightarrow \mathbb{R}^n$. Typically, neural networks are used to learn the encoder and decoder function by minimizing a reconstruction loss between the encoder input and the decoder output [174]. However, regularisation is needed so that the functions E and D do not convergence to the identity operators. One of the simplest regularisation methods is to introduce a bottleneck, i.e., compress the representation with the encoder while still re-creating the original input as well as possible with the decoder. Thus, a bottleneck layer limits the number of neurons. An alternative (or a supplement) is to limit the number of active neurons by enforcing sparsity. Autoencoders with a sparsity constraint are also known as *sparse autoencoders*. A sparsity constraint can be imposed with L_1 or L_2 regularization or with a kullback-leibler (KL) divergence[14] between

[30]: Rumelhart et al. (1987)

[173]: Baldi (2012)

[174]: Ranzato et al. (2007)

14: the kullback-leibler divergence is a measure of how one probability distribution differs from another reference probability distribution

[175]: Le et al. (2012)

[176]: Vincent et al. (2008)

[177]: Rifai et al. (2011)

[103]: Kingma et al. (2014)

average neuron activations and an ideal activation distribution [175]. Other popular versions of autoencoders are denoising autoencoders [176], contractive autoencoders [177], and variational autoencoders (VAE) [103]. For *denoising autoencoders*, the input is disrupted with noise (e.g. by adding Gaussian noise or removing some pixels by using dropout) before it is fed into the encoder. The goal of the decoder is to reconstruct the clean version of the input without the added noise. By doing so, denoising autoencoders learn to create a 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 unimportant for the reconstruction, and only essential 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 latent vector, the input is mapped to a vector representing the mean μ and another vector representing the standard deviation σ of a multivariate Gaussian distribution. This is done by adding two fully connected layers after the encoder, one layer to predict μ and the other layer to predict σ. Afterwards, a variable is sampled from this distribution $z \sim \mathcal{N}(\mu,\ \sigma^2 \cdot \epsilon)$, where $\epsilon \sim \mathcal{N}(0,\ 1)$ [15] and fed into the decoder D. By doing so, the latent space becomes, by design, continuous and allows random sampling and data interpolation.

15: this process is also called the reparameterisation trick

3.4.2 Self-Supervised Learning

Other approaches do not compress data but generate a supervision signal themselves. These methods are thus based on *self-supervised learning*. Typically, the input is augmented, and either the augmentation parameters are predicted, the original version of the image is reconstructed, or consistent representations among augmented views of the image are learned. 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) [178]. This not only allows to generate part of images but also to learn visual representations of the image [179–181]. Other approaches rotate images and predict the rotation angle to generate representations [182, 183]. Many further methods and a comprehensive overview is given by Chen et al. [184].

[179]: Pathak et al. (2016)

[180]: He et al. (2022)

[181]: Shi et al. (2022)

[182]: Komodakis et al. (2018)

[183]: Zhai et al. (2019)

[184]: 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 representations of these two views are pushed together in the embedding space by maximising their similarity [185–187]. Subsequently, the encoder can be used to generate image representations that are used for downstream tasks, such as image classification. However, contrastive learning can also leverage information from external annotations. For example, Khosla et al. [188] 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.

[185]: Chen et al. (2020)

[186]: He et al. (2020)

[187]: Caron et al. (2020)

[188]: Khosla et al. (2020)

3.4.3 Relevance

In this thesis, image representations are learned with two different principles; horizontal self-organisation uses contrastive learning by forcing samples from the same class to have similar representations and samples from different classes to have high diversity. Vertical self-organisation uses independently trained variational autoencoders to obtain good representations of input patches. Thus, image representations are learned with the aforementioned methods.

3.5 Meta-Learning

TODO: NOT SURE IF THIS SECTION IS NEEDED / SHOULD BE KEPT...
-> NOT THAT RELEVANT SO FAR

Meta-learning is the process of distilling experience from multiple learning cycles and using the accumulated experience to improve future learning processes [189]. Therefore, meta-learning is also referred to as "learning-to-learn" [190] and improves learning on a lifetime (i.e. single agent) and evolutionary timescale (i.e. population of agents) [191]. 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 an 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.

[189]: Hospedales et al. (2021)

[190]: Thrun et al. (1998)

[191]: Schrier (1984)

16: typical meta objective functions aim to improve generalisation or learning speed

In the following, the categorisation according to Hospedales et al. [189] is used to describe the different characteristics of meta-learning algorithms.

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

Meta-Optimiser How the outer meta-learning algorithm learns, i.e. the form of the learning algorithm.

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

3.5.1 Meta-Representation

The first dimension of the meta-learning landscape is the *meta-representation*, i.e. defining the meta-knowledge to be learned. One possibility is to learn good initial parameters that the model can use during base learning. Good initial parameters are only a few gradient steps away from a set of parameters that can solve a task drawn from a set of tasks. A popular algorithm to learn initial parameters is called MAML [192, 193] 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 [194–196].

[192]: Finn et al. (2017)

[193]: Finn et al. (2018)

Black-box models use meta-learning to directly provide the parameters to the base learning algorithms that are required to fulfil a task such as classification, i.e. the base learning algorithm assigns a sample directly to a class with the given parameters, without parameter optimisation [197–199]. Special cases of black-box models are approaches based on metric learning: The outer meta-learning process optimises a model to transform inputs into representations that the inner process can use for classification by comparing the cosine similarity or euclidean distance between them [195, 200, 201].

Other approaches meta-learn the optimiser of the inner base learning algorithm [202–204]. Typically, such approaches generate each optimisation step of the base learning algorithm based on the model's parameters and a given base objective function. The trainable component can, for example, be simple hyper-parameters such as the learning rate [204] or more sophisticated such as pre-conditioned matrices [205]. Even the learning algorithm itself can be learned [113].

[204]: Li et al. (2017)

[205]: Park et al. (2020)

[113]: 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 used as loss value by the inner optimiser. Common goals of these approaches are to obtain a loss that has less local minima [206, 207], provides better generalisation [208–210], leads to more robust models [211], or to learn from unlabelled data [212, 213].

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 [214], while other approaches use evolutionary algorithms to learn the topology of LSTM cells [215] or to model the network by a graph of network-blocks [216].

[214]: Zoph et al. (2017)

[215]: Bayer et al. (2009)

[216]: Real et al. (2019)

The outer meta-learning loop can also meta-learn hyper-parameters such as regularisation-strength [217, 218], task-relatedness for multi-task learning [219], or sparsity-strength [219]. Furthermore, it can meta-learn suitable data augmentation methods [220, 221] or improve the selection process for the samples of a mini-batch [222, 223]. Even the dataset itself can be learned with dataset distillation; a rather large dataset can be summarised as a smaller dataset with only a few support images [224, 225] that still allow good generalisation on test images. In sim2real learning [226], also the graphics engine [227, 228] can be trained in an outer loop so that the performance on real-world data is maximised.

[219]: Franceschi et al. (2017)

[226]: Andrychowicz et al. (2020)

3.5.2 Meta-Optimiser

The second dimension of the meta-learning landscape is the *meta-optimiser*. The meta-optimiser describes how the algorithm in the outer meta-learning loop is learned. Many methods use backpropagation of error and gradient descent (c.f. Section 2.2) to optimise the meta parameters [192, 202, 211, 217–219, 225]. However, this algorithm has some well-known weaknesses (c.f. Section 2.3) and requires differentiability. When the inner base learning algorithm includes non-differentiable steps [220] or the outer meta objective function is non-differentiable [229], many methods utilise the RL paradigm to optimise the outer objective [230].

[220]: Cubuk et al. (2019)

[229]: Huang et al. (2019)

[230]: Duan et al. (2016)

However, using RL to alleviate the differentiability requirement is usually computationally very costly. Besides gradient descent and RL, evolutionary algorithms can be used for learning the outer meta-learning function [231–233]. These algorithms have no differentiability constraints, do not suffer from gradient degradation issues, are highly parallelisable, and can often avoid local minima better than gradient-based methods [233]. The downside of evolutionary algorithms is that 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.

[233]: Salimans et al. (2017)

3.5.3 Meta-Objective

The third dimension of the meta-learning landscape is the *meta-objective*. The meta-objective describes the goal of the meta-learning algorithm. 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 [192, 202] or many [219, 234] 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 the sum of the loss calculated after each update step to encourage *faster* learning [235]. Third, the goal of the meta-learning can either be to solve better any task drawn from a set of (often related) tasks (i.e. multi-task setting) [192, 200, 211], or to solve one specific task better than when only the inner base learning algorithm is used (i.e. single task setting) [219]. Finally, the meta-optimisation can either be done offline (i.e. the inner and outer loop alternate) [192] or online (i.e. the meta-learning takes place within a base learning episode) [211].

[202]: Ravi et al. (2017)

[192]: Finn et al. (2017)

[219]: Franceschi et al. (2017)

[234]: Wichrowska et al. (2017)

[192]: Finn et al. (2017)

[200]: Snell et al. (2017)

[211]: Li et al. (2019)

[219]: Franceschi et al. (2017)

[192]: Finn et al. (2017)

Neuroscientific Concepts 4

4.1 The Discrepancy 37
4.2 Self-Organisation 38
4.3 Net Fragments 40
Sparsity 42
4.4 Lateral Connections 42
4.5 Other Principles 44

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

This chapter discusses the following: Section 4.1 describes the discrepancy between biological learning and deep learning and points out why it is hard to transfer biological concepts into algorithms. Next, Section 4.2 presents self-organisation as one of the fundamental concepts for natural intelligence and proposes possible implementations. Afterwards, Section 4.3 discusses net fragments. Next, Section 4.4 proposes a method for implementing lateral connections in a deep learning setting. Finally, Section 4.5 points out other promising principles that are not implemented in this thesis but might be promising in future work.

4.1 The Discrepancy

The human brain inspired deep learning, and consequently, many essential components of a deep learning model are named after their biological counterparts. Nevertheless, this linkage is misleading: both systems work fundamentally different, and transferring concepts from one field to the other is not straightforward. 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 the main component of the nervous system. A neuron is a well-defined cell consisting of components such as the cell body, dendrites, and an axon communicating with other neurons via synapses (c.f. Section 2.1). 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.

In the field of deep learning, this definition is much vaguer. It could even be argued that neural networks do not contain neurons: 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 the mathematical model of a neuron[2] (c.f. Section Section 2.2). However, only the parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ exist as actual parameters, while an artificial neuron is rather a mental construct. Furthermore, modern network architectures are typically not

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

solely based on such fully-connected modelling pattern. Modern deep learning architectures for computer vision are usually based on either convolutional operations (c.f. Section 2.2.1) or attention mechanisms [13]. The convolutional and pooling operations of CNNs cannot properly be explained with the simple model of artificial neurons[3]. The same applies to the attention mechanism of vision transformers. Furthermore, both architectures use various other concepts that have been shown to improve results and computation efficiency, such as normalisation layers, etc. As a result, findings from neuroscience are often based on the investigation of neurons or neuronal structures, and consequently, the theories and findings derived from them cannot be transferred to deep learning architectures in a simple way, as neural networks model parameters, which are processed by various mathematical functions.

[13]: Dosovitskiy et al. (2021)

3: for example, it is unclear how the pre-synaptic and post-synaptic activity is to be interpreted in the context of convolutional layers

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 at a certain point in time, it could be argued that a neuron's activity is binary (excited or not excited). The output of artificial neurons, on the other hand (or, more precisely, the result of the matrix operation, often referred to as a neuron), is a floating point number and thus can represent an infinite number of states. Furthermore, artificial neurons have no temporal dynamics, while biological neurons communicate with temporal patterns of spikes. Thus, a large part of the intelligence of biological networks lies in the timing of the spikes, which are not taken into account by artificial neural networks. These differences make it extremely difficult to link biological and artificial networks.

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 points out that biological concepts are to be interpreted somewhat abstractly and cannot be implemented as an algorithm in the same form as in the human brain. Therefore, in the following sections, various concepts are discussed, and intuitions are given on how these concepts can be interpreted in a deep learning setting without time dynamics but with complex neuron states.

4.2 Self-Organisation

It is known that large parts of the human brain are self-organising [236]. Self-organisation is the process by which systems consisting of many units acquire their function through local interaction and without interference from an external supervisory system. Recently, renowned scientists [2] put forward the hypothesis that this process of self-organisation is *the* key mechanism of natural intelligence. Dresp [237] describes seven clearly identified properties of self-organisation 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 organisation, and (vii) dynamic system growth. However, it is not obvious how these insights from neuroscience can be integrated into a deep learning framework.

[236]: Kelso (1995)

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

[237]: Dresp-Langley (2020)

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

Claim 1 *End-to-end backpropagation of error violates the principle of self-organisation. 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 [135, 136]. The biologically most plausible learning algorithm is Hebbian learning (c.f. Section 2.5.1) and its variants, such as contrastive Hebbian learning [238]. However, Hebbian learning seems to be not well suited for learning good image representation if a network is trained from scratch. Instead, Hebbian learning might be promising for post-processing already retrieved representations (c.f. Appendix A).

[135]: Crick (1989)

[136]: Grossberg (1987)

[238]: Movellan (1991)

Besides Hebbian learning, many alternative and biologically more plausible algorithms have been proposed, such as the feedback alignment (FA) algorithm [163], generalised recirculation [239], as well as target propagation (TP) [240] (c.f. Section 3.3). However, Bartunov et al. [167] have demonstrated that these algorithms do not scale to large vision datasets such as ImageNet [241], and only work for smaller datasets such as MNIST [242], and CIFAR-10 [95]. The only algorithms that scale well and rely on local updates only use proxy objective functions (c.f. Section 3.3). Thus, the use of proxy objective functions[4] seems promising; the weight updates are done in separate local units, yet the power of current deep learning training algorithms can be exploited. Thus, it allows to create complex systems that scale to large datasets while being based on local units interacting with each other.

[163]: Lillicrap et al. (2016)

[239]: O'Reilly (1996)

[240]: LeCun (1986)

[167]: Bartunov et al. (2018)

4: 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 can be applied to local units. Thus, self-organisation takes place through optimising multiple local units instead of optimising a single system.*

Another ambiguity is the definition of local units in a deep learning setting. One of the strengths of deep learning systems is that matrix multiplications allow to calculate the layer's outputs in one step. Calculating each neuron's activity separately, would be very inefficient. Therefore, the smallest meaningful unit for local updates is a layer and not a single neuron. In this thesis, the layers of a network are optimised separately with proxy objective functions. Doing so makes the local units small but allows optimising them efficiently. If a neural network is visualised layer-wise from left to right, the self-organising units (i.e. the layers) line up horizontally. That is why layer-wise self-organisation is referred to as "horizontal self-organisation" in this thesis (c.f. Figure 4.1).

Implementation 1.2 *Self-organisation takes place within local units. A local unit can be a part of a model, such as a layer. In this thesis, optimising layers individually with proxy objective functions is called horizontal self-organisation.*

Figure 4.1: Two ways to build self-organising units: Self-organisation 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 self-organising 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 be processed by independent units (i.e. models). It is important that one model does not process the entire set of existing patches, as it is the case with the vision transformer [13]. If only one model is used, end-to-end backpropagation of error is applied to a single unit. However, if a graph of independent models processes the patches, each model can be considered a self-organising unit. In this thesis, this kind of self-organisation is called "vertical self-organisation" (c.f. Figure 4.1).

[13]: Dosovitskiy et al. (2021)

Implementation 1.3 *A second type of self-organising units are models that process a subset of the input data. In this thesis, this type of self-organisation is called vertical self-organisation.*

Self-organisation is considered a key for natural intelligence and not compatible with end-to-end backpropagation of error. Therefore, self-organising units are built by either splitting the model into smaller units (horizontal self-organisation) or by splitting the data into patches (vertical self-organisation).

4.3 Net Fragments

Another fundamental principle for intelligence, 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 3.1). For example, some net fragments may represent shapes and structures while a multitude of such net fragments together represents objects such as persons or entire scenes. Net fragments are a compositional data structure: Some low-level features can be composed into a higher-level feature, and multiple higher-level features represent an entire object. To some extent, neural networks do this as well; data is fed into the network, the first layer extracts low-level patterns, and the subsequent layers combine these patterns in higher-level features hierarchically. However, there is one big difference: The features used to fulfil a specific task, such as classification, are extracted from the latent space of *one* single layer. On the other hand, net fragments in the human brain are not believed to be present at a specific layer but to be built up over multiple neurons and over a short period of time.

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

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

Deep learning models are not based on time-dependent spikes of neurons. 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 detected 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 should not be extracted from a single but from multiple layers. Intuitively, this seems promising: For example, auto-regressive models applied to speech capture different information at different layers of the network [243]. 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 relevant information.

[243]: Chung et al. (2019)

Implementation 2.1 *Net fragments cannot be extracted from a 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 sparse and diverse activations: The activations must be sparse and diverse because the object in the image consists of only a small subset of all learned features. Thus, only a small subset 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 in Appendix B. In summary, it was found that neural networks do not contain net fragments with meaningful activation patterns by default. Typically, some neurons 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 a sparsity and diversity constraint is added to the objective function, net fragments can be identified in the neuron's activation patterns.

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 are not necessary, for example, to obtain better classification performance. An alternative that leads to robust and meaningful activations as well is to model the net fragments as a probability distribution. For example,

variational autoencoders (c.f. Section 3.4) model their latent space as a multivariate Gaussian distribution.

In this thesis, object representations are extracted from multiple layers. Furthermore, the model's latent space is either constraint with sparsity and diversity functions or modelled as probability distribution.

4.3.1 Sparsity

Sparistiy is 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 and thus have a binary state (i.e. active or inactive). Artificial neurons, on the other hand, use a floating point number as an internal state and can thus represent an infinite number of states. Thus, an artificial neuron can 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 *can be* sparse.

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 [244]. 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 used to make models smaller or to increase inference speed [245, 246]. From a biological point of view, this process of first creating an extensive network and then shrinking it is not plausible[6]. Sparse activations, on the other hand, can increase robustness [247]. 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 better interpretable activations (c.f. appendix Chapter B).

[244]: Rochefort et al. (2009)

[245]: Louizos et al. (2018)

[246]: Hoefler et al. (2021)

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

4.4 Lateral Connections

In addition to forward connections, also lateral connections are located in the visual cortex [248]. Thus, the biological neurons are connected to the neurons in the subsequent layer and 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.

[248]: Gilbert et al. (1990)

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

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

Neural networks lack this time dynamic, and neurons do not suddenly become inactive. 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. Then, in the next step, the input can be slightly changed through data augmentation to obtain 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 input sample $\boldsymbol{x}$ is augmented n times to obtain n different views $\boldsymbol{x}'[1], \ldots, \boldsymbol{x}'[n]$ of the same sample. At every time-step t, the sample $\boldsymbol{x}'[t]$ 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 can be implemented by interpreting the lateral connection as a recurrent connection. An illustrative architecture of such a model is shown in Figure 4.2. A data sample $\boldsymbol{x}$ can be augmented with data augmentation methods n times to obtain n different views $\boldsymbol{x}'[1], \ldots, \boldsymbol{x}'[n]$ 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 $\boldsymbol{a}^{[l]}[t]$ 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 $\boldsymbol{a}^{[l]}[t-1]$. The activation map of the previous time-step is provided by a recurrent (lateral) connection. Thus, the activation map $\boldsymbol{a}^{[l]}[t]$ is not only calculated based on the activation map of the previous layer $\boldsymbol{a}^{[l-1]}[t]$ but also based on the activation map of the previous time-step $\boldsymbol{a}^{[l]}[t-1]$. Lateral connections allow each layer to preserve the activation map $\boldsymbol{a}^{[l]}[t-1]$ from the previous time-step or to correct it based on a slightly different input $\boldsymbol{a}^{[l-1]}[t]$ 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, thus supporting activated features.

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 or correct their previous activation maps (lateral support).*

With horizontal self-organisation, the lateral connections can be implemented as recurrent connections. However, for vertical self-organisation, the lateral connections are interpreted differently: In this case, each sub-model extracts a distinct low-level feature. Based on this feature, a guess can be made as to which higher-level feature or object the extracted feature belongs to. This guess can be supported or rejected through communication with neighbouring sub-models. Thus, the lateral connections can be seen as a "support channel" or "communication channel" between sub-models.

Implementation 4.2 *In the case of vertical self-organisation, a lateral connection can be implemented as a connection between sub-models to vote 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. 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 object change 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. Continuous input might help to get a better representation of objects through self-supervised learning. Suppose 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. In that case, the object representations derived from this continuous stream can be homogenised. These principles are already applied to some extent by self-supervised learning systems for computer vision (c.f. Section 3.4.2). In contrastive learning, two different views are derived from one image by data augmentation, and their representations are then pushed closed together in the feature space [185, 187, 249]. 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.

[185]: Chen et al. (2020)

[249]: Chen et al. (2020)

[187]: 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.*

In the nervous system, efference copies of motor signals in the form of neuronal activities are directly sent to the brain's sensory system [250]. Such efference copies can be helpful for understanding objects and how they behave when undergoing object transformations. Therefore, the agent must be allowed to interact with its world. This gives the agent more information about the objects and also about the physical properties of a (virtual or real) world. For example, he can perform different actions on different objects: Rotating, squeezing, stretching or moving objects. By doing so, he receives 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.

[250]: 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 his own. For example,

suppose the agent examines an object, and his current internal object representation does not describe how the object looks from the side. In that case, the agent can rotate the object accordingly and complete or correct the representation. Such a behaviour could be implemented within the agent, for example, by optimising an entropy-based loss so that the agent explores the data and tries to learn unknown things [251, 252].

[251]: Storck et al. (1995)

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

Horizontal Self-Organisation 5

5.1 Methods 47
Extraction of Representations 50
Lateral Connections 51
Hierarchical Features 53
5.2 Results 55
Dataset 55
Baseline Model 55
Lateral Connections 56
Hierarchical Features 56

In this thesis, two different concepts of self-organisation are implemented. This chapter presents a method based on horizontal self-organisation. Horizontal self-organisation is based on training small units within a network independent from each other. The first section presents the methodology (i.e. how it is implemented), and the second section evaluates the obtained results.

5.1 Methods

The first choice for networks based on horizontal self-organisation is what the independent units are, i.e. which group of network parameters are trained independently. In this thesis, a single layer is used as a self-contained unit. A layer 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.

Each layer optimises a proxy objective function. Proxy objective functions are used because this type of local optimisation allows self-organisation and excellent performance on larger datasets [167, 172], in contrast to other local learning algorithms such as target propagation, synthetic gradients or feedback alignment (c.f. Section 3.3). Figure 5.1 visualises the updates based on a proxy objective function. An objective function is calculated for each layer, and the optimisation algorithm only updates the parameters within this layer. Thus, the gradients do not flow back to preceding layers.

[167]: Bartunov et al. (2018)

[172]: Duan et al. (2022)

Figure 5.1: The flow of gradients within the network based on horizontal 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.

A combination of diversity and sparsity constraints is used as a loss function, which leads to representations that are easy to interpret, 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 B, the sparsity is achieved by using the kullback-leibler (KL) divergence [175]. The model consists of several linear layers l with a ReLU activation function (c.f. equation (2.5)). A layer l consists of $n^{[l]}$ neurons and the activations $\boldsymbol{a}^{[l]}$ are calculated for a given input $\boldsymbol{a}^{[l-1]}$ as

[175]: Le et al. (2012)

$$\boldsymbol{a}^{[l]} = \boldsymbol{W}^{[l]} \cdot \boldsymbol{a}^{([l-1]} + \boldsymbol{b}^{[l]} \tag{5.1}$$

whereby $\boldsymbol{W}^{[l]}$ is the weight and $\boldsymbol{b}^{[l]}$ the bias of layer l. If a mini-batch contains m samples, the activation probability $\hat{\rho}_i^{[l]}$ of a neuron $a_i^{[l]}$ can be calculated as:

$$\hat{\rho}_i^{[l]} = \frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{1 + e^{-a_i^{[l]}}} \right) \tag{5.2}$$

where $\left(\frac{1}{1+e^{-a_i^{[l]}}} \right)$ is the sigmoid function that squeezes the activation in the range between 0 and 1. With the KL divergence, the divergence of the actual activation probability $\hat{\rho}_i^{[l]}$ 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 $\mathcal{L}_s^{[l]}$ of layer l is the sum of the divergence between each neurons' activation probability $\hat{\rho}_i^{[l]}$ and the target activation probability ρ:

$$\mathcal{L}_s^{[l]} = \sum_{i=1}^{n} KL(\rho || \hat{\rho}_i^{[l]}) \tag{5.4}$$

Thus, it is enforced that 5% of the neurons are active per sample. The second constraint is a diversity constraint. The goal is that the activations of *different* objects are diverse. For this purpose, the activations $\boldsymbol{a}^{[l]}$ 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 the activations $\boldsymbol{a}^{[l](i)}$ and $\boldsymbol{a}^{[l](j)}$ of two samples $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$:

$$\cos\left(\boldsymbol{a}^{[l](i)}, \boldsymbol{a}^{[l](j)}\right) = \frac{\boldsymbol{a}^{[l](i)} \cdot \boldsymbol{a}^{[l](j)}}{\max\left(||\boldsymbol{a}^{[l](i)}||_2, ||\boldsymbol{a}^{[l](j)}||_2\right)} \tag{5.5}$$

The information of image labels $y^{(i)} \in C$ is needed to push representations of identical objects together or to push representations of different objects apart. The diversity loss $\mathcal{L}_d^{[l]}$ minimises the cosine similarity if the activations stem from different classes (i.e. $y^{(i)} \neq y^{(j)}$), or maximises the similarity if they stem from the same class (i.e. $y^{(i)} = y^{(j)}$).

$$\mathcal{L}_d^{[l]} = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} k \cdot \cos\left(\boldsymbol{a}^{[l](i)}, \boldsymbol{a}^{[l](j)}\right) \text{ if } i \neq j \tag{5.6}$$

whereby k changes the 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, this loss function has two significant drawbacks: First, comparing two samples can be unstable; if one sample is not representative of a class, the other sample may be pushed away from its cluster centre. Second, if the dataset has more than two classes, more negative samples ($y^{(i)} \neq y^{(j)}$) than positive samples ($y^{(i)} = y^{(j)}$) are compared. With many classes, the optimisation algorithm focuses mainly on pushing negative samples apart and does not pay enough attention to pushing positive samples together. These problems are alleviated with two measures: Instead of comparing two samples, a sample is compared to the average activation of a class, which can be interpreted as the cluster centre of a class. Furthermore, each sample is once compared with the cluster centre from the same class and once with a cluster centre from another class. Thus, positive and negative samples are compared equally often.

Lets denote $\boldsymbol{a}^{[l]}_{C=\bar{c}}$ as the average activation of class c in layer l. Thus, the goal is to push the activation of a sample $\boldsymbol{a}^{[l](i)}$ closer to $\boldsymbol{a}^{[l]}_{C=\overline{y}^{(i)}}$ and away from $\boldsymbol{a}^{[l]}_{C=\overline{v}^{(i)}}$, where $v^{(i)}$ is a random class samples drawn from $C \backslash \{y^{(i)}\}$[1]. Such a loss could be implemented as

1: the set of all classes without $y^{(i)}$

$$\mathcal{L}_d^{[l]} = \frac{1}{n} \sum_{i=1}^{n} \cos\left(\boldsymbol{a}^{[l](i)}, \boldsymbol{a}^{[l]}_{C=\overline{v}^{(i)}}\right) - \cos\left(\boldsymbol{a}^{[l](i)}, \boldsymbol{a}^{[l]}_{C=\overline{y}^{(i)}}\right) \tag{5.8}$$

However, the problem with this loss is that all cosine similarities become 1, and the loss becomes 0 if all activations are identical. Therefore, a margin between the similarities is enforced, similar to the triplet-margin-loss [253].

[253]: Balntas et al. (2016)

$$\begin{aligned} \mathcal{L}_d^{[l]} = \frac{1}{n} \sum_{i=1}^{n} \max\Bigg[& \cos\left(\boldsymbol{a}^{[l](i)}, \boldsymbol{a}^{[l]}_{C=\overline{v}^{(i)}}\right) \\ & - \cos\left(\boldsymbol{a}^{[l](i)}, \boldsymbol{a}^{[l]}_{C=\overline{y}^{(i)}}\right) + \text{margin}, 1 \Bigg] \end{aligned} \tag{5.9}$$

where margin is a hpyerparameter that is set to 1. The loss applied to each layer is the sum of sparsity loss and diversity loss, with the diversity loss weighted by $\lambda = 0.1$:

$$\mathcal{L}^{[l]} = \mathcal{L}_s^{[l]} + \lambda \cdot \mathcal{L}_d^{[l]} \tag{5.10}$$

One problem with proxy objective functions is that this type of training often requires layer-wise training. First, layer 1 is trained until convergence, then the weights are frozen, then layer 2 is trained and so on. Such sequential training 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 function 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 to previous layers. Since the gradients are only needed locally, no large graph of gradients has to be calculated, which reduces the memory utilisation on GPUs and allows larger mini-batch sizes. In addition, this type of architecture is straightforward to parallelise since each layer (i.e. each independently trained unit) or a group of layers can be trained on a different GPU:

Figure 5.2: The network architecture of the fully connected model for horizontal 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 $\mathcal{L}^{[l]}$ (c.f. equation (5.10)) with the Adam optimizer [10] and a learning rate of $\eta = 1 \cdot 10^{-3}$. The mini-batch size is $60,000$.

[10]: Kingma et al. (2017)

5.1.1 Extraction of Representations

In accordance with net fragments as defined in Section 4.3, the representations are not extracted at a specific point in the network (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 $\boldsymbol{a}^{[l]}_{C=\bar{c}}$ for each class $c \in C$ and layer l is determined. These averages from the training set represent prototypes of each class in each layer and can be considered a 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. When a new sample $\boldsymbol{x}^{(i)}$ is classified, the cosine similarity between the activations $\boldsymbol{a}^{[l](i)}$ of this sample and the class prototypes $\boldsymbol{a}^{[l]}_{C=\bar{c}}$ is calculated in each layer.

$$\cos^{[l](i)}_{C=c} = \cos\left(\boldsymbol{a}^{[l](i)}, \boldsymbol{a}^{[l]}_{C=\bar{c}}\right) = \frac{\boldsymbol{a}^{[l](i)} \cdot \boldsymbol{a}^{[l]}_{C=\bar{c}}}{\max\left(||\boldsymbol{a}^{[l](i)}||_2, ||\boldsymbol{a}^{[l]}_{C=\bar{c}}||_2\right)} \tag{5.11}$$

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

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

This results in a weighted voting; if a layer is certain that the sample belongs to a specific class, 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 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 4.4, lateral connections in the human brain serve neurons to support each other. There, it is also described that lateral connections can be implemented as recurrent connections. There are several ways to implement recurrent connections. A straightforward possibility is to concatenate the layer input $\boldsymbol{a}^{[l-1]}[t]$ at time t with the layer output $\boldsymbol{a}^{[l]}[t-1]$ at time $t-1$. This is visualised in Figure 5.3. Of course, $\boldsymbol{a}^{[l]}[t-1]$ is undefined at $t = 0$. In this case, $\boldsymbol{a}^{[l]}[t-1]$ is initialized with zeros.

Another option is to use a second weight matrix, as it is usually done in recurrent layers. Therefore, the equation equation (5.1) is extended as follows:

$$\boldsymbol{a}^{[l]}[t] = \boldsymbol{W}^{[l]}_x \cdot \boldsymbol{a}^{[l-1]}[t] + \boldsymbol{b}^{[l]} + \overbrace{\boldsymbol{W}^{[l]}_h \cdot \boldsymbol{a}^{[l]}[t-1]}^{\text{lateral connection}} \tag{5.13}$$

whereby $\boldsymbol{W}^{[l]}_x$ is the weight multiplied with the layer input and $\boldsymbol{W}^{[l]}_h$ the weight multiplied with the previous layer output. In both cases, the layer receives information about the activations at the previous time

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.

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 step. Thus, the model receives different views of the same image and can adjust its activations 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 5.4 visualises how this augmentation affects samples of the MNIST dataset [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.

Recurrent connections are typically used to process sequential data or text. Thereby, he data is usually first read token by token before an output is generated.[2] Thus, such models depend on sequential timesteps and 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.

2: for example, in text classification, all word tokens of a sentence are typically read before the model classifies a sentence. This is necessary because all information is required, and classification cannot be done on the basis of a single token.

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 exploding 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 5.1.1. If an object representation can be assigned to one class label with a high probability, the sequential processing of different image views can be aborted. If this is not the case, further time steps can be carried out until the model has sufficiently high confidence in its prediction. Thus, the number of time steps can be sample-dependent.

Figure 5.4: Data augmentation applied on 10 samples of the MNIST dataset. The original samples from the dataset are shown on the left, and 9 augmented versions of the same samples are shown on the right.

5.1.3 Hierarchical Features

A criticism of the proposed model is that it does not learn hierarchical features, even though hierarchical features are one of the main reasons for the excellent performance of deep learning systems. The loss 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 features; 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, two 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) While specific information in the form of the input image 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. Thus, the training process enforces a hierarchy by fusing specific and general information over multiple layers. These measures can be applied either individually or in combination with each other.

Convolutional Architecture

In a CNN, the field of view in the first layers is restricted by design but becomes larger in subsequent layers. Consequently, using a CNN

architecture instead of a model based on fully connected layers alleviates the problem of non-hierarchical features.

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

The CNN architecture used in this thesis is shown in Figure 5.5. Three convolutional layers with ReLU activation functions 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.

Two-Way Information Flow

[154]: Hinton (2022)

Hinton [154] introduced with the forward-forward (FF) algorithm (c.f. 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 with the wrong label and the network is trained to push the activations below a certain threshold. He found that a feature hierarchy is created 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 from the other end. 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.

In this thesis, this process is implemented in a different way, which does not have this disadvantage. Identical to the FF algorithm, the image is fed into the network's input layer for multiple time steps. Conversely, the label is not fed directly into the output layer but is made available to the last layer within the loss function. Each layer maximises the mutual information (MI) between its activations and the activations of the previous and subsequent layers. Thus, the first layer maximises 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 samples to abstract class labels is learned. During inference, the last layer directly predicts latent representations that can be assigned to a class. This process is visualised in Figure Figure 5.6.

Figure 5.6: Horizontal self-organisation with mutual information loss: Each layer maximises the mutual information (MI) between its activations and the activations of the previous and subsequent layers. The first layer maximises the MI between its activations and the input image (instead of the previous layer), and the last layer maximises the MI between its activations and the image label (instead of the subsequent layer).

TODO: Currently, various experiments are ongoing to find out how this can be implemented (therefore, yet to be explained in more detail).

5.2 Results

The results obtained by this model are presented in the following. It is important to note that these models aim not to achieve the best possible results in terms of classification accuracy on a benchmark. In large-scale comparisons, it has often been found that end-to-end backpropagation of error is far superior to other methods, especially in image classification [167]. Instead, the aim here is to examine interesting concepts that might lead to new learning principles in the long run.

[167]: Bartunov et al. (2018)

5.2.1 Dataset

The model is trained and evaluated on the MNIST [18] dataset. MNIST is a dataset 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 [254], a corrupted MNIST benchmark for testing the out-of-distribution robustness of image classification models. This dataset consists of the MNIST data with 15 types of corruptions applied to the samples.

[254]: 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 by 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?

Vertical Self-Organisation 6

6.1 Methods 57
Predicting bigger Patches . . . 60
Communication 61
Class Prediction 66
6.2 Results 67

This chapter presents a method based on vertical self-organisation. The idea of vertical 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. This agreement takes place through lateral communication. It is essential that each model is independent of the other models and that the parameters are not shared between them. Otherwise, the architecture would be comparable to vision transformer [13] and the input patches would no longer be analysed independently. Thus, the architecture would not be self-organising anymore. In the following, the methodology (i.e. how vertical self-organisation is implemented) is presented, and the obtained results are discussed in detail.

[13]: Dosovitskiy et al. (2021)

6.1 Methods

A crucial design decision for vertical self-organisation is the architecture of the models. In this thesis are variational autoencoders [103] (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).

[103]: Kingma et al. (2014)

Typical autoencoders 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 (c.f. Section 3.4.1).

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

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

ϵ is a random variable sampled from $\mathcal{N}(0, 1)$ and determines the magnitude of the variance. The sampled latent representation $\boldsymbol{z}'$ is fed through the decoder, and the image $\hat{\boldsymbol{x}}$ is recreated. The model is trained to minimise 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 m images $\boldsymbol{X} = \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}$, the loss is defined as

$$\mathcal{L}_{\text{rec}} = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)})^2 \tag{6.2}$$

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

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

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

In the 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 $P(\boldsymbol{h}|\boldsymbol{X})$ 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 $Q(\boldsymbol{h}|\boldsymbol{X})$ as $\mathcal{N}(\boldsymbol{\mu}_Q, \boldsymbol{\sigma}_Q)$, then the KL divergence simplifies to

$$\begin{aligned}\mathcal{L}_{KLD} = KL\left[\mathcal{N}_Q(\boldsymbol{\mu}_Q, \boldsymbol{\sigma}_Q)||\mathcal{N}(0,1)\right] &= -\log\boldsymbol{\sigma}_Q + \frac{\boldsymbol{\sigma}_Q^2\boldsymbol{\mu}_Q^2}{2} - \frac{1}{2} \\ &= \frac{1}{2}\left(\boldsymbol{\sigma}_Q^2\boldsymbol{\mu}_Q^2 - 1 - 2\log\boldsymbol{\sigma}_Q\right)\end{aligned} \tag{6.5}$$

Interested readers may find a formal proof of the formulas above and a more detailed derivation in the original paper [103]. Thus, the loss for the variational autoencoder is defined as:

$$\begin{aligned}\mathcal{L}_{\text{VAE}} &= \mathcal{L}_{\text{rec}} + \beta\cdot KL\left[\mathcal{N}_Q(\boldsymbol{\mu}_Q, \boldsymbol{\sigma}_Q)||\mathcal{N}(0,1)\right] \\ &= \underbrace{\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i)^2}_{\text{reconstruction loss}} + \lambda\cdot\overbrace{\frac{1}{2}(\boldsymbol{\sigma}_Q^2\boldsymbol{\mu}_Q^2 - 1 - 2\log\boldsymbol{\sigma}_Q)}^{\text{KL divergence}}\end{aligned} \tag{6.6}$$

where β is a weight factor of the KL divergence. Training a variational autoencoder with this loss function can lead to excellent image reconstruction. In the case of vertical self-organisation, however, a well-formed latent space is of more interest than an outstanding image reconstruction: The goal is to obtain good image representations, and thus the goal is to have a well-formed latent space and not perfect reconstruction. Variational autoencoders have the notorious problem that the KL divergence term becomes vanishingly small during training [255]. This issue is known as the KL vanishing problem. This problem can be alleviated by applying annealing schedules to the KL term (i.e., changing β over time). In this thesis, monotonic annealing is used as proposed by Bowman et al. [255]. However, cyclic annealing strategies might lead to even better results [256]. The disadvantage is that training with cyclic annealing takes longer, which outweighs the advantage of only minimally better latent representation. These annealing strategies are shown in Figure 6.1. When β is small, the model is forced to focus on reconstructing the input rather than minimising the KL loss. When β is increased, the model gradually improves the probabilistic distribution in the latent space.

[255]: Bowman et al. (2016)

[255]: Bowman et al. (2016)

[256]: 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 autoencoders. The upper graph shows a monotonic increase of β, and the lower graph a cyclical annealing strategy. The picture is from [256].

In this thesis, 4 variational autoencoders 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 input size 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 6.2.

Figure 6.2: Architecture of the variational autoencoders used for vertical self-organisation.

Each VAE is trained independently to minimise the loss function as described in equation (6.6). Adam [10] is used as an optimisation algorithm with a learning rate of $1 \cdot 10^{-3}$, and the mini-batch size is 32.

[10]: Kingma et al. (2017)

Figure 6.3: Architecture of the variational autoencoders used for vertical self-organisation with bigger field-of-view upsampling.

6.1.1 Predicting bigger Patches

Applying 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 distinguish similar-looking patches better. For example, the patches for the digits 4, 5, and 6 look very similar for the first VAE that receives patches 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 a VAE predicts bigger patches or the entire image, the target predictions of the digits 4, 5, and 6 become different. Thus, while similar latent representations are sufficient to predict only a patch of these digits, predicting larger patches or the whole picture requires more separated latent representations. Thus, predicting bigger patches helps to push apart latent representations of objects with a high patch-wise similarity but a low image-wise similarity.

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

In this thesis, an additional layer is used in the decoder to predict larger patches: The last layer with 32 channels and a stride of 2 is used twice so that the decoder has 4 transposed convolutional layers with 128, 64, 32, and 32 channels. Thus, the output size of the decoder is doubled. This architecture is shown in Figure 6.3. When using 4 VAEs, the input is

divided into 4 patches. If the decoder makes the output twice as large as the input by up-sampling, the entire image is predicted by each of the VAEs. This process is visualised in Figure 6.4.

6.1.2 Communication

The VAEs receive patches of the input image and thus have a limited field of view on the image. Figure 6.5 visualises the patches that are fed into the VAEs when the MNIST dataset [18] is used. The first row shows the average of all images of the same class. Thus, these images are very representative for the dataset. However, none of the models receives the entire image as input. Instead, the patches shown in the 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. Thus, these classes are not distinguishable by *one* single VAE on its own. Therefore, communication with neighbouring models is necessary to agree on an image-level representation.

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

This concept links well to the biological model: A single VAE can be seen as a neuron or a group of neurons. Initially, the VAEs have many suggestions to which classes the received patch could belong. However, through communication with neighbouring VAEs, suggestions are constantly ruled out because neighbouring VAEs do not support them. 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 quickly becomes sparse as only the neurons supporting each other remain active. This process leads to the emergence of higher-level features (i.e. net fragments) that are representative for the input [85].

[85]: von der Malsburg et al. (1987)

One of the biggest strengths of autoencoders is that they do not rely on labels but can be trained in an unsupervised manner. To keep this strength, also communication between the models should not rely on labels. Therefore, different types of communication are proposed that work without labels: communication based on latent representations, communication based on 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 the latent space. This information can be communicated to the neighbouring VAEs. Thus, in a second step, the

VAEs have more information available; their prediction and information from neighbouring VAEs. This additional information allows them to improve their prediction. The process can be repeated over a fixed number of time steps or until the network reaches an attractor state (i.e. the latent representations no longer change). In the following, different types of communication are proposed.

Model Heads

Each VAE learns the mapping from a patch into the latent space, i.e. obtains a μ and σ 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 self-organised. A consequence of independent systems is that μ and σ have different meanings in the latent spaces of different VAEs. Thus, one VAE cannot interpret the Gaussian parameters of another VAE.

In this thesis, it is proposed to learn a linear transformation to map the Gaussian parameters of one VAE to the Gaussian parameters of another VAE. The mapping from μ_1 to μ_2, respectively from σ_1 to σ_2 is done by the simple transformation

$$\hat{\mu}_2 = w_{m12} \cdot \mu_1 \tag{6.7}$$

resp.

$$\hat{\sigma}_2 = w_{s12} \cdot \sigma_1 \tag{6.8}$$

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 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.

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

Figure 6.7: Communication between VAEs by forwarding reconstructed patch, example based on 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.

If each VAE predicts the Gaussian parameters of the other VAEs, then these predictions can be communicated, and each VAE receives further suggestions from neighbouring VAEs. Thus, each VAE has its own calculated Gaussian parameters and suggestions from neighbours about what its Gaussian parameters might be. This allows a VAE to correct its prediction. Each VAE receives additional information 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. On the other hand, the VAE that sees the patch at the bottom left can distinguish these digits. Thus, the VAE at the bottom left can help the VAE at the top left to choose a better latent representation.

Reconstructed Images

As described in Section 6.1.1, the prediction of larger patches helps to arrange the latent space better. However, 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 previous section, the reconstructed image can also be communicated. 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 how the reconstructed version of its input patch looks. 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 visualised for the first VAE in Figure 6.7. The problem with this approach is that during the first time-step, all VAEs communicate their reconstructed patch to the neighbouring VAEs. As a result,

Figure 6.8: Communication between VAEs by forwarding prediction of neighbouring patches, example based on 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 as input.

in the second time-step, each VAE receives information about the entire image and thus has access to global instead of local image information. Using more VAEs could alleviate this problem as the field of view becomes bigger but remains spatial-limited.

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 VAE receives 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 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 each VAE i ,that reconstructs the input patch $\boldsymbol{p}_i \subset \boldsymbol{x}$, masks out $\boldsymbol{p}_i$ in its predictions, i.e. only communicates $\boldsymbol{c}_i \subset \hat{\boldsymbol{x}}$ where $\boldsymbol{p}_i \nsubseteq \boldsymbol{c}_i$. Otherwise, an input patch of a VAE is fed through the encoder, and the decoder's prediction is communicated to the other VAEs. Then implicitly reconstructed data, not just neighbourhood predictions, are forwarded to other VAEs 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

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

channel but 4 input and output channels. The input patch is fed into the first channel, and a reconstructed version is predicted in the first output channel. Thus, the first channel correspond to a classic VAE. The remaining three channels are for communication: Each VAE generates a separate communication output for its three neighbouring VAEs. Thus, each VAE receives an input patch to reconstruct and communication data. These four input matrices are stacked and fed into the encoder. 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 patch reconstruction on the output channel 1 and communication data that is sent 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 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 and to have a well-shaped latent space. The only difference is that model can now access four input channels instead of one. However, the communication channels must be trained in order for them to be helpful for other VAEs. Therefore, an additional loss term is applied to the 3 output communication channels. Thereby, the idea is to optimise the output of these communication channels 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 $\mathcal{L}_{\text{VAE}_i}$ based on the reconstruction goodness and the shape of the latent space (c.f. equation (6.6)). The 3 communication channels of VAE i can influence the loss $\mathcal{L}_{VAE_j}$ 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 loss of the other VAEs.

$$\mathcal{L}_{c_i} = \frac{1}{n-1} \sum_{n}^{j=1} \mathcal{L}_{\text{VAE}_j}, \text{ if } i \neq j \tag{6.9}$$

Thus, the loss of a VAE with a communication channel is

$$\mathcal{L}_{\text{VAE}_{\text{C}\,i}} = \mathcal{L}_{\text{VAE}_i} + \beta \cdot \mathcal{L}_{\text{KLD}_i} + \beta_2 \cdot \mathcal{L}_{c_i} \tag{6.10}$$

where β_2 is a weight factor for the communication channel. Thus, each model optimises 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 still needs to be implemented (therefore, not explained in more detail).

6.1.3 Class Prediction

The VAEs are examined to determine whether they are suitable for predicting the class label. First, all VAEs are trained until they can reliably reconstruct images and the latent space is well formed. After training, the average value of $\boldsymbol{\mu}$ is calculated for each class $c \in C$ from the training set and each VAE. This average value is denoted as $\overline{\boldsymbol{\mu}}_{vc}$ for VAE v and class c. It is defined for n_c samples of class c as

$$\overline{\boldsymbol{\mu}}_{vc} = \frac{1}{n_c} \sum_{i=1}^{n_c} \boldsymbol{\mu}_i \tag{6.11}$$

This average value can be considered the "cluster centre" of a class in latent space or as a world model of a class. A given sample is matched against these world models per class to make a prediction. As with horizontal self-organisation, the cosine similarity between a sample and the class prototypes is used to determine their similarity. To predict a sample s, it is fed through the encoder to obtain $\boldsymbol{\mu}_{vs}$. Then, this sample is compared with all class prototpyes by computing the cosine similarity:

$$\cos_{vsc} = \cos(\boldsymbol{\mu}_{vs}, \overline{\boldsymbol{\mu}}_{vc}) = \frac{\boldsymbol{\mu}_v s \cdot \overline{\boldsymbol{\mu}}_{vc}}{\max(||\boldsymbol{\mu}_{vs}||_2, ||\overline{\boldsymbol{\mu}}_{vc}||_2)} \tag{6.12}$$

Thus, the cosine similarity $\cos_{vsc}$ is calculated for each VAE v between the class prototypes c and the sample s. Afterwards, the class c with the highest average cosine similarity between the sample activations $\boldsymbol{\mu}_{vs}$ and the class prototype $\overline{\boldsymbol{\mu}}_{vc}$ over all VAEs v is used as the prediction.

$$\arg\max_{c \in C} \frac{1}{4} \sum_{v=1}^{4} \cos_{vsc} \tag{6.13}$$

Similar to horizontal self-organisation, VAEs with 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 autoencoders (VQ-VAE) [257] 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. Thus, an image is fed into the encoder to obtain the encoder output $\boldsymbol{h}_e$. Afterwards, a nearest neighbour lookup is made to find the code vector most similar to $\boldsymbol{h}_e$. The code-vector, also called the quantised vector $\boldsymbol{h}_q$, is fed into the decoder to reconstruct the image.

[257]: van den Oord et al. (2017)

TODO: This still needs to be implemented (therefore, not explained in more detail).

6.2 Results

As for horizontal self-organisation, the MNIST [18] and MNIST-C [254] datasets are used to evaluate the models (c.f. Section 5.2.1).

Different models are trained to evaluate the proposed architecture. Table 6.1 gives an overview of the models. A single VAE is used as the baseline (model no. (i)), which receives the entire image as input and reconstructs it. This VAE is obviously trained without time steps and communication. Vertical 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 (model no. (ii)) or predicts the entire image (model no. (iii)). Furthermore, different types

of communication are added: First, the VAEs forward 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 (model no. (iv)) or predicting the entire image[2] (model no. (v)). Second, each VAE predicts the whole image and forwards the predicted *input patch of the neighbour* to the other VAEs over 4 time steps (model no. (vi)) (c.f. Figure 6.8). Third, the VAEs are 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) (model no. (vii)).

2: 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 vertical 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 hyperparameter: 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 when the entire image is sent, and no further communication cycles are necessary. However, if only a patch of the image is forwarded to the other VAEs, the predictions may be updated over several time steps. Therefore, 4 time steps are used. The predictions usually do not change after 4 time steps, and thus, using 4 time steps seems enough. When only patches are forwarded, the models contain much more dynamics: A VAE makes a prediction, 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 VAEs correct each other over several time steps. An example of four predicted samples is shown in Figure 6.11: The left side shows the initial prediction of the model no. (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 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 communication cycles has an error of 0.16.

Figure 6.11: Four random predictions of the model no. (vi) over 4 time-steps: The initial prediction is shown on the left and the prediction after 4 communication cycles on the right.

Of particular interest 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 [258] and visualised. Since 25 VAEs are trained in this evaluation, visualising all t-SNE plots is infeasible. Instead, Figure 6.12 only shows the plot of the baseline (model no. (i)), the plot of the model no. (ii) as an example of four VAEs reconstructing patches, and the plot of the model no. (vi) as a representative example of an architecture with communication and image-reconstruction.

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

As expected, the latent space of model no. (i), which receives the entire image as input, is well organised, and the classes are distinguishable within the latent space. Model no. (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 patches 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 communication between the VAEs. By applying the proposed measures (i.e. communication and predicting the whole image instead of reconstructing the input patch), the structure 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 no. (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. The suitability for classification can be measured by applying a k-means clustering and then calculating the normalised mutual information (NMI) score between the cluster centres and the labels. This metric compares cluster assignments and labels by calculating their consistency: It measures how well a cluster can be assigned to a label. Table 6.2 shows the NMI scores of the models. The NMI score is challenging to interpret as an absolute number. However, a relative comparison of the scores is of much more interest. Model no. (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. Model no. (ii) consists of

four VAEs without any measures and is thus considered a lower bound. Overall, most models seem to have a well-formed latent space thanks to the proposed measures and achieve scores close to the upper bound.

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 no. (ii) and (iii) as well as models no. (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 no. (ii) → model no. (iii)) and from 0.37 to 0.59 (model no. (iv) → model no. (v)) respectively.

A slight improvement through communication can also be observed. Model no. (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 no. (v)), 0.54 (model no. (vi)) and 0.54 (model no. (vii)). However, the improvement is not as significant 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 no. (iii) has a very high NMI score compared to the baseline, indicating that this model can classify the samples even though no communication is used, and therefore, the patches contain too much information. This makes communication less important, and adding communication does not significantly improve results.

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. The results are shown in Table 6.3. For each VAE of a model, the cosine similarity according to equation (6.12) between a sample and a class prototype is calculated. Thus, the accuracy can be calculated not only for the entire model (i.e. all 4 VAEs together) but also for each VAE independently. Table 6.3 shows the average accuracy per VAE in the column "Avg. VAEs". Furthermore, by calculating the average cosine similarity, a voting between VAEs is done. The accuracy of this voting is shown in the column "Overall". The voting works exceptionally well: for example, model no. (ii) has 4 VAEs with an average accuracy of 62.4% on the MNIST dataset. The best VAE has an accuracy of 66.1%, and the worst 59.7%. By voting, the overall accuracy increases to 81.7%, significantly better than the best VAE.

On the MNIST dataset, the base model (model no. (i)) is significantly outperformed by some models in terms of accuracy. However, 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 the MNIST and MNIST-C dataset. The average accuracy per VAE and overall accuracy are reported for both datasets. 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 (6.12)). The overall accuracy is calculated by averaging the cosine similarity according to equation (6.13).

The accuracy is calculated by comparing a sample with *one* prototype per class. However, using only one prototype is 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 horizontal self-organisation (c.f. Section 7.2.2). Thus, the number of prototypes per class should be increased.

Future Work & Conclusion 7

7.1 Neuroscientific Concepts . . . 73
7.2 Implemented Models 74
Horizontal Self-Organisation 74
Vertical Self-Organisation . . 75
7.3 3-Staged Model 76
7.4 Useful Representations 77
7.5 Cognition and Reasoning . . . 79

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 (while the goal should still be to build a system that supports humans and not behaves like one). However, the implementations presented in this thesis are far from such a system. Consequently, a conclusion is drawn, and the next steps are proposed for the models presented in this thesis, but a long-term vision is presented as well. Specifically, some insights are given to the neuroscientific concept that seems very promising in section Section 7.1, and future work is described for horizontal self-organisation in Section 7.2.1 as well as for vertical 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 for biological intelligence. However, since the discrepancy between biological and artificial models is large, it is difficult to incorporate these biological concepts into a computer system. This thesis makes concrete suggestions on how such concepts can be implemented. Implementing these concepts in the form of horizontal and vertical self-organisation demonstrates that these ideas can facilitate the learning process. The evaluation of the suitability of neuroscientific concepts was very iterative: many concepts were tested but discarded in the course of the work because they did not add enough value to existing systems. For example, much time was spent on sparsifying activations over several time steps as it is done in the human brain. However, it was found that the 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.

The most promising concepts identified in this work for improving deep learning systems are self-organisation, net fragments, sparsity[1], lateral connections, continuous input, and embodiment. For each of these mechanisms, a concept for implementation is proposed, except for continuous input and embodiment. However, these proposals are the author's interpretation and are only one of many possible solutions. These concepts can also be interpreted differently, and an alternative implementation could

1: but not over time

also lead to promising results. Furthermore, exploring continuous input and embodiment in future work could help to build more intelligent systems.

7.2 Implemented Models

The concepts from neuroscience are implemented in two specific ways, these are called horizontal and vertical 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 not considered an appropriate baseline.

Rather, the proposed implementations showcase interesting neuroscientific concepts. 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 future work (c.f. Section 7.3, 7.4 and 7.5) that improves AI in the long-term.

7.2.1 Horizontal Self-Organisation

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

The proposed horizontal self-organisation approach uses a novel 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 [154], which have caused a stir in the community by challenging the classical backpropagation of error as an optimisation algorithm. However, despite the fact that algorithms using proxy objective functions are biologically more plausible, these algorithms do not 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, evaluating the models presented in this work and other models, such as the forward-forward algorithm, has crushed this initial belief: No model that uses proxy objective function achieved better results on the corrupted MNIST-C dataset than end-to-end backpropagation of error.

[154]: Hinton (2022)

For classification, a sample is compared to the average activation of each class. This average activation can be understood as an 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 not sufficient. As an example, Figure 7.1 shows different versions of the digit 4 that can be found in the MNIST dataset. This shows that a digit can be written in different ways[2]. Consequently, the digits and, thus, the activations look different for samples from the same class, and the use of more than

2: this applies to all kinds of datasets: Most objects can have various visual characteristics (size, colour, shape) or look different from different viewpoints

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 a lot more accurate.

Figure 7.1: Different versions of the digit "4" in the MNIST dataset.

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 with a supervised approach since the labels are used in the loss function. However, the use of labels is generally not desirable because the manual creation of labels is labour-intensive. On the other hand, an unsupervised (or self-supervised) approach extracts features without labels. Moreover, it is known that the brain also learns unsupervised to a large extent and is thus biologically more plausible. Thus, the implementation of a loss function that does not rely on labels would be beneficial. Furthermore, the model should be tested on other datasets than MNIST to evaluate how well it can scale.

7.2.2 Vertical Self-Organisation

TODO: this section is incomplete as many experiments are ongoing...

Vertical self-organisation splits the input image into multiple patches and processes each patch with an independent model. Similar to horizontal 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 vertical self-organisation, the μ values per VAE and class 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.

An essential concept in the human brain is the prevention of early commitment [259] (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 into higher-level features. Low-level features are detected locally; thus, it is already determined which feature is within the image before global information is considered. The proposed architecture for vertical self-organisation reduces this problem by creating representations of image patches. The patches are so small that they contain only enough 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 higher-level features or an object. Therefore, the VAEs can be considered low-level feature extractors, such as the first layer of a convolutional 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

[259]: Marr (2010)

features (local view) and communicates with other VAEs (global view) to determine their representation. Therefore, 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 image compression is learned during training. This further reduces the problem of individual VAEs committing to a class.

In future work, this effect should even be intensified by further reducing the size of the patches and using more VAEs. With 16 VAEs (4×4), the field of view is restricted to (8×8) pixels and each VAE can only produce very poor image-level representations, that can only be randomly assigned to a class. With more VAEs, communication can also be restricted locally: With 2×2 VAEs, each VAE automatically receives information about the entire image when communicating 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, meaning that it implicitly receives local and global image information. This could reduce the independence of each VAE. Consequently, it should be investigated how this architecture can become translation invariant similar to CNNs.

7.3 3-Staged Model

Christoph von der Malsburg, co-supervisor of this thesis and one of the authors of the "Theory of Natural Intelligence" paper[3] [2], is a pioneer of the theory of self-organising fibre connections in the visual system [84, 90, 260, 261]. His theory theory refers to an 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 fundamental principle is to avoid the "fallacy of early commitment" [259]. The idea is that local decisions should only be taken if plausible in the light of high-level features, while high-level features can only be defined based on low-level patterns. The brain can handle this very well, as the Gestalt psychology demonstrates[4] [262]. Convolutional networks, on the other hand, extract local low-level patterns in the first layers and compose them into higher-level patterns in subsequent layers. Thus, by definition, the first layers commit to a pattern based on local information without considering global information.

The 3-staged model does not suffer from this problem: The stage S1 extracts local features. However, no commitment to higher-level features is made. Stage S2, on the other hand, contains object representations. The local features from S1 are mapped to S2 with bi-directional projection

3: the work that served as *the* source of inspiration of this thesis

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

[260]: Wolfrum et al. (2008)

[84]: Willshaw et al. (1979)

[261]: Wiskott et al. (1996)

[90]: Fernandes et al. (2015)

[259]: Marr (2010)

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

[262]: Köhler (1929)

fibres. This mapping iteratively composes low-level features into high-level features. Thus, local information from S1 is used to find a suitable object representation in S2 and vice versa. These stages are described in more detail in the following. Stage S1 is a mechanism based on lateral connections between neurons that are spatially close to each other. Initially, many neurons are activated by the retina's impulses. However, these neurons are immediately turned off if they do not have sufficient lateral support from neighbouring neurons. The lateral connections are thus used to keep each other active. This not only has the effect of "mutually confirming each other" but also helps to form higher-level features from low-level patterns: There are oscillations in the brain that also influence the number of active neurons. In other words, the number of active neurons continuously increases and decreases. Only those with the enough local support remain if the number of active neurons is reduced. Thus, many neurons are deactivated in a short time. With each reduction of active neurons, higher-level features (so-called net fragments) are selected through lateral support. Since only pixel groups that support each other remain active, neurons that represent insignificant features are continually switched off.

The lateral support, however, is limited to local patterns, i.e. the lateral support in S1 is not sufficient to recognise whole objects or scenes, but only parts of them. In stage S2, however, entire objects are represented. These object representations in S2 are independent of transformation and position. Projection fibres do the mapping of several low-level features in S1 to a concrete object in S2. These fibres implicitly remove transformations and position information from the features and assemble them into objects. Projection fibres can be interpreted as a kind of graph that combines features hierarchically, ignoring local distortions to create transformation-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 instances can finally be related to the brain's memory.

The 3-staged model fundamentally differs from current CNN architectures and includes promising concepts such as preventing early commitment and using dynamic fibres. However, a concrete implementation that works for complex images like natural photos has yet to be implemented, despite a lot of effort in research. The model based on vertical self-organisation could be used as a feature extractor. This model is well suited because only local features are extracted that cannot be assigned to an object and thus have no early commitment. However, future work needs to extend this architecture with projection fibres.

7.4 Useful Representations

In this thesis, models are proposed that extract representations of images. However, the question of how useful these representations are still remains[5].

The usefulness of representation depends on the use case: Usually, good representations should contain exactly the information that is needed[6] (i.e. be compact), be robust, be interpolatable, and be well-aligned with

5: despite the usual ML tasks such as image classifications

6: retaining as much information as possible is often not useful as this corresponds to storing (a compressed version) of the image

the real world. For example, representations from modern deep learning systems are very useful for typical vision tasks such as classification or segmentation. In some cases, it could be argued that deep learning representations are even superior to humans: For example, modern deep learning systems [263] can identify millions of faces with more than 99% accuracy, which is very difficult for humans [7]. However, what works poorly for deep learning systems is recognising an object as the same instance, regardless of which transformations have been applied. In general, deep learning systems cannot learn a good world model and understand transformations applied to various objects with different characteristics.

[263]: Jung et al. (2022)

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

One way to address this problem is to allow the model to interact with the world (i.e. perform actions). Interacting allows a model to learn how an action changes the view of an object. Object-independent transformation such as rotations can be learned if the same actions are applied to different objects. 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 [250]. Thus, animals integrate their action (i.e., their movement) 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. [264] argue that such efference copies are helpful 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. They enforce that transformations of the real world can also be applied to latent representations, i.e. that the mental objects and the real-world objects remain consistent when similar transformations are applied to them.

[250]: Keller et al. (2012)

[264]: Keurti et al. (2022)

Giving an agent an embodiment[8] 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 [265] argues that perceiving an object is rather about understanding how it transforms and behaves and not creating a mental copy of it.

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

[265]: 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 postulated by LeCun [266]. He divides the process into 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 $(a[0], \ldots, a[t], \ldots a[T])$ that the virtual world model executes. (iii) The world model, in turn, predicts a sequence of world state representations resulting from the proposed action sequence $(s[1], \ldots, s[t], \ldots s[T])$. (iv) A cost model estimates the total costs for each state sequence as a sum over time steps $F(x) = \sum_{t=1}^{T} C(s[t])$. (v) Based

[266]: LeCun (2022)

on multiple cost predictions, the actor proposes the action sequence with the lowest costs. (vi) The actor 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 optimise the system.

Overall, it seems very important that a model does not receive a sequence of unrelated images and their labels as input, as in typical classification tasks. Instead, the agent should receive a continuous stream of related input images and information about their transformation. In this way, transformations can be observed and learned, which is fundamental to understanding visual scenes.

7.5 Cognition and Reasoning

At the end of this thesis, I want to share my personal opinion on how deep learning systems can be improved fundamentally: 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[9]. According to my definition, these pixel representations are only a part of object representations. Object representations are mental constructs that contain 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, many other parameters make up an object, such as its abilities (e.g. can it move?) 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. However, for us humans, these two objects have a clear relationship defined by their ability to swim and the typical place where they are found, i.e. the sea. Another example is heat: It feels hot, and in my mental world model have things like fire, the blue flame of a welding machine, and a hob, an obvious relationship. However, all these objects look completely different. Many modern pattern extraction systems only model one-dimensional pixel representations and not multi-dimensional object representations and thus cannot build such relationships. Extending these one-dimensional object representations ("what does the object look like") to multi-dimensional representations might be one of the key elements to creating 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.

9: end-to-end backpropagation of error seems to work excellent to generate "pixel"-representations

In addition to this construction of multi-dimensional object representations, an understanding of the world's physics is necessary. This second type of representation summarises knowledge that relates to all objects, for example, the earth's gravitational force, the fact that living beings

cannot walk through solid materials such as walls, etc. The question that arises now is how such a system can be implemented. It is obvious that more than simply showing pictures and the corresponding labels is required 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, getting visual observations together with efference copies allows an agent to understand how objects behave. Furthermore, such a world allows understanding differences and similarities between objects. 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 [251]). Thus, also continuous input is necessary. Training a model with a sequence of random and unrelated images is most likely not sufficient.

[251]: Storck et al. (1995)

The network architecture must enable and encourage the capture of the world's structure. I think that a mental multi-dimensional world model is helpful for storing and relating the appearance, behaviour, transformation capabilities, abilities, etc., of each object and a second model to store general knowledge about the world. Such an architecture allows modelling a multi-dimensional relationship between objects[10].

10: in a purely visual perception model, a division into the dimensions of appearance, behaviour during transformation, and physics (e.g. the object is standing on the ground) would already be a massive step forward

The learning algorithm should consider several things to learn a good world model: A key element is that the learning algorithm decides 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[11].

11: this also applies to humans: We need prior knowledge to survive - We have prior knowledge through evolution, and, after birth, our parents explain the world to us. In fact, we are not able to survive without proper support in the first years

An intelligent system consists of several components: One very crucial component is the perception system. From today's point of view, this is the component that is best mastered, i.e. deep learning systems can capture useful patterns in high-dimensional data very well[12]. Around this system, a world model has to be created, which stores the understanding of the world. It is 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. Such behaviour can be implemented by an entropy-based loss function [251]. The second question is 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 multi-dimensional object information, as 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 a real understanding of the world, interactions with the world and predictions of how the next state of the world looks seem to be very central[13]. A straightforward 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) [266], more advanced are GFlowNetworks [267]. After such predictions are learned, reversing this process, as in diffusion models, could be a first step towards casual

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

[251]: Storck et al. (1995)

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

[266]: LeCun (2022)

[267]: Bengio et al. (2021)

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 and random motions. If a similar concept is applied to the learned prediction of actions, 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 take years and, unfortunately, are out-of-scope of this thesis. However, I hope to be able to address such questions as concrete research topics soon, ideally in a similar constellation as in this Master's thesis.

Appendix

Experiments Hebbian Learning A

Hebbian learning is the most plausible learning algorithm from a biological point of view (c.f. Section 2.5.1). Because of the high plausibility and the self-organising property, the effectiveness of Hebbian learning for extracting image representations was investigated in multiple preliminary experiments. In this thesis, it was observed that Hebbian learning works well for tiny networks but does not scale to larger networks or more complex datasets. Of course, it cannot be concluded from these preliminary experiments that Hebbian learning cannot be used to train networks for extracting image representations on a larger scale. However, it was found that it is not trivial and that applying the Hebbian learning rule on modern architectures is not sufficient to obtain good feature extractors.

When the classical Hebbian learning rule as described in equation (2.14) is used, the weights in the network become symmetric. As a result, the model does not learn different features, and the image representations are very poor. This symmetry can be broken using the ABCD-Hebbian learning rule [268]. The ABCD learning rule calculates the weight update Δw_{ij} from neuron i to neuron j based on the pre-synaptic activity a_i of neuron i and post-synaptic activity a_j of neuron j as follows:

[268]: Niv et al. (2001)

$$\Delta w_{ij} = \eta_{w_{ij}} \cdot (A_{w_{ij}} a_i a_j + B_{w_{ij}} a_i + C_{w_{ij}} a_j + D_{w_{ij}}) \tag{A.1}$$

where $\eta_{w_{ij}}$ is a weight-specific learning rate, $A_{w_{ij}}$ is a correlation coefficient, $B_{w_{ij}}$ is a presynaptic coefficient, $C_{w_{ij}}$ is a postsynaptic coefficient, and $D_{w_{ij}}$ is a bias coefficient. The bias coefficient $D_{w_{ij}}$ can be interpreted as an individual inhibitory or excitatory bias of each connection in the network. As proposed by Najarro and Risi [111], the coefficients are learnt through an evolution strategy [233]. The symmetry is broken by applying the ABCD learning rule, and the image representations improve. However, image processing networks usually consist of many parameters and using an evolutionary strategy to find them does not scale well. Thus, the author's impression is that current Hebbian learning rules are only of limited use for learning deep learning-based image extraction networks.

[111]: Najarro et al. (2020)

[233]: Salimans et al. (2017)

A.0.1 Hebbian Pruning

Hebbian learning has not led to good image representations. However, it was found that Hebbian learning might be promising to prune models trained with end-to-end backpropagation of error: In the following, Hebbian learning is used to push weights between neurons towards zero or to set them exactly to zero.

Therefore, a neural network consisting of two convolutional layers, followed by a max-pooling layer, and three linear layers is used. The

convolutional layers have a kernel size of 3×3 and a stride of 1. The first convolutional layer has 32 channels, and the second has 64 channels. The max-pooling layer has a kernel size of 2×2 and thus halves the size of the feature map. The activation map is then flattened and fed into 3 fully connected layers with an output size of 1024, 128, and 10, respectively. Between each layer, ReLU activations are employed. The network is visualized in Figure A.1.

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

[10]: Kingma et al. (2017)

The model is trained to classify digits with backpropagation of error on the MNIST dataset [18]. Adam [10] is used as optimisation algorithm to minimise the negative log-likelihood (NLL) loss. The mini-batch size is 32 and the learning rate is $5 \cdot 10^{-4}$. When the loss reaches a plateau, the learning rate is reduced to $1 \cdot 10^{-4}$.

After the model is trained, the first two fully connected layers within the network are pruned with Hebbian learning: 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. Therefore, a mini-batch is sampled, and the correlation between all neurons 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 reduced:

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

$$w_{ij} := w_{ij} + \Delta w_{ij} \tag{A.3}$$

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

The *second* Hebbian-based method proposed pushes some 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.

A.0.2 Results

For both methods, the model is trained on the MNIST training dataset [242], and the weights are frozen. Afterwards, the weights are pruned based on the MNIST training dataset and evaluated on the MNIST and MNIST-C[1] [254] test dataset. The *first* Hebbian-based method can further improve the results even after the training has converged, as shown in Table A.1.

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%	99.31%
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 the MNIST-C test dataset decreases from 0.6013 to 0.4766. Furthermore, the accuracy increases roughly by 1.2% on MNIST-C, which is remarkable, considering that the model has been trained with gradient descent until convergence and only the weights are reduced after training (without further parameter updates). It also has to be considered that this is only a preliminary experiment, and this method most likely can be further improved. Thus, this experiment indicates that Hebbian could be used as a post-training step to improve robustness by reducing the model's weights.

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 shows the ratio of weights kept.

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 [269]. Furthermore, it leads to sparse representations that can improve robustness. 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. It can be observed

[269]: Liang et al. (2021)

that the performance is better when weights between neurons with low correlation are set to 0 than if random weights are set to 0. Thus, Hebbian learning might be promising the reduce the number of parameters within a network.

B Net Fragments

This chapter examines how representations of typical deep learning architectures look like. Especially net fragments are of interest, i.e. (groups of) neurons representing an object's features. 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

Most deep learning architectures violate the first principle as the latent representations are extracted from a single layer and subsequently are used for a downstream task. Since deep learning architectures violate this first principle 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 net fragments are contained in the models, specific features would be expected to trigger the activation of a group of neurons when they are present, and the same neurons would be inactive when this feature is not present. Thus, different neurons should be active for different objects. There, net fragments improve interpretability and lead to sparse and diverse activations[1], which could increase robustness. However, sparsity and diversity are not necessary to achieve higher classification accuracy.

1: since each object only contains a (small) subset of the learned features, only a (small) subset of neurons should be active

Deep learning architectures usually consist of several layers with millions of neurons, leading to millions of activations [21–26]. The input data is processed in a complex way, making 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 is 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). Tiny networks are sufficient to analyse these images, and this dataset thus leads to fewer network activations, simplifying 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 depicting the numbers $0 - 9$, each with 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 dataset, various features and, consequently, many net fragments exist. It is unclear which features are learned by the network. Regardless of what these net fragments represent, the network should have specific characteristics; If a neuron or a group of neurons are part of a net fragment representing 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. Thus, different neurons should be active depending on the digit.

Digits with common characteristics should have some overlapping network activities; For example, 7 is contained in 3, 3 is contained in 9, and 9 is contained in 8. Therefore, these digits should have some common network activations. The digit 1, on the other hand, has nothing in common with 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 and examined for net fragments: First, classification networks are used to investigate whether different architectures are more likely to contain net fragments than others. Second, autoencoders are used to investigate whether adding constraints to the loss function can encourage net fragments.

B.1 Classification

B.1.1 Methods

Different architectures are trained to investigate the emergence of net fragments in classification networks. 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 B.1. 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 an FC layer as a feature extractor, and model no. 11 has a Conv. layer with two hand-crafted kernels to extract horizontal and vertical lines as a feature extractor. After the feature extractor, all models have a similar "head" consisting of 2 fully connected layers. Between each layer, a ReLU activation is employed. The feature extractors aim to extract specific features from the images, 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, thus, this capacity is 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 B.1. The "head" is identical for all models and consists of a sequence of the following layers; FC (out size=12) → FC (out size=10) → softmax.

The models' parameters are optimised by minimising the cross-entropy loss with the Adam optimiser [10] so that the models learn to predict the label 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 10 epochs. After each epoch, the model's weights and activations of each layer are stored. These vectors are visualised and investigated for net fragments.

[10]: Kingma et al. (2017)

No.	Encoder Description
1	conv. layer (ks=3×3, ch=2) $\rightarrow$ head
2	conv. layer (ks=3×3, ch=4) $\rightarrow$ head
3	conv. layer (ks=5×5, ch=2) $\rightarrow$ head
4	conv. layer (ks=5×5, ch=4) $\rightarrow$ head
5	conv. layer (ks=3×3, ch=2) $\rightarrow$ conv. layer (ks=3×3, ch=4) $\rightarrow$ head
6	conv. layer (ks=3×3, ch=4) $\rightarrow$ conv. layer (ks=3×3, ch=8) $\rightarrow$ head
7	conv. layer (ks=3×3, ch=2) $\rightarrow$ max-pooling $\rightarrow$ conv. layer (ks=3×3, ch=4) $\rightarrow$ head
8	conv. layer (ks=3×3, ch=4) $\rightarrow$ max-pooling $\rightarrow$ conv. layer (ks=3×3, ch=8) $\rightarrow$ head
9	head
10	FC (in size=$9 * 9$, out size=12) $\rightarrow$ ReLU $\rightarrow$ head
11	Hand Crafted conv. layer for vertical & horizontal edge detection (ks=3×3, ch=2) $\rightarrow$ ReLU $\rightarrow$ head

Table B.1: A description of the different classification networks investigated for net fragments. A ReLU activation is employed between each layer. "ks" is the kernel size of a convolutional layer, and "ch" is the number of channels.

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 identify 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. This layer contains high-level net fragments that are simple to interpret[2].

2: the last layer cannot be used as it consists of 10 neurons representing the 10 classes

Figure B.2 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 precisely 0, red circles are low activations > 0, and green circles represent strong activations[3].

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

It can be observed 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 images 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 evident 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. This supports the assumption that typical deep learning architectures do not contain brain-like net fragments.

Figure B.2: 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, and blue means activation is off (i.e. 0).

B.2 Autoencoders

B.2.1 Methods

Similar to the experiments with classification networks, different autoencoders are investigated. Thereby, it is investigated if constraints in the loss function can encourage net fragments. The analysed autoencoder architectures used are shown in Table B.2.

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 optimiser [10] so that the output looks similar to the input even though a bottleneck-layer reduces the network's capacity between encoder and decoder. The learning rate is $5 \cdot 10^{-4}$, the mini-batch size is 32 samples, and the model is trained for 10 epochs.

[10]: Kingma et al. (2017)

The reconstruction loss $\mathcal{L}_{\text{rec}}$ 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:

$$\mathcal{L}_{\text{rec}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \begin{cases} \frac{1}{n} \sum_{i=1}^{n} |x_i - \hat{x}_i|, & \text{if L1 distance} \\ \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2, & \text{if L2 distance} \end{cases} \tag{B.1}$$

As described above, activations containing net fragments should be sparse and diverse. Therefore, a sparsity and a diversity loss are added to the reconstruction loss to encourage net fragments. This combination of sparsity and diversity is also in line with the ideas of LeCun to obtain

2) autonomous machine intelligence [266]. For the sparsity loss $\mathcal{L}_s$, the kullback-leibler (KL) divergence is used [175]. The hidden representations in the bottleneck are denoted as $\boldsymbol{h} = h_1, \ldots, h_{n_h}$. Furthermore, the average activation probability of a neuron h_j over m input samples can be calculated as

[175]: Le et al. (2012)

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{1 + e^{-h_j}} \right) \tag{B.2}$$

where $\left(\frac{1}{1+e^{-h_j}}\right)$ is the sigmoid function that squeezes the activation in the range between 0 and 1. Furthermore, $\rho = 0.05$ is a sparsity parameter that defines a target activation probability of each neuron. If the kullback-leibler divergence between all $\hat{\rho}_j$ and ρ is minimised, an average activation probability of ρ is enforced for each neuron in the hidden layer. The KL divergence 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 $\mathcal{L}_s$ is the sum of the KL divergence between all $\hat{\rho}_j$ and ρ and thus enforces that on average only 5% of the hidden representations $\boldsymbol{h}$ are active:

$$\mathcal{L}_s(\rho, \hat{\rho}) = \sum_{j=1}^{n_h} KL(\rho || \hat{\rho}_j) \tag{B.4}$$

Thus, this constraint ensures that the activations are sparse, consistent with the net fragments concept. Another property of net fragments is that the activations should be diverse. Therefore, a novel diversity loss $\mathcal{L}_d$ is proposed. This loss is based on the fact that the activations $\boldsymbol{h}$ in the bottleneck layer should be as different as possible for all samples of a mini-batch $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$. The sum of the dot product between the activation of a sample $\boldsymbol{h}^{(i)}$ and all other samples $\boldsymbol{h}^{(j)}$, for $i \neq j$ is small if the samples within a batch have a high diversity. Therefore, the diversity loss can be defined as:

$$L\mathcal{L}_d(\boldsymbol{z}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n_h} \boldsymbol{h}^{(i)} \cdot \boldsymbol{h}^{(j)}, \text{ if } i \neq j \tag{B.5}$$

Minimising $\mathcal{L}_d$ leads to diverse activations in the bottleneck layer. The overall loss can be defined as the sum of these three loss components, with λ_s and λ_d being hyperparameters that control the influence of the diversity and sparsity loss.

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

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 visualised 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 represents 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) without knowing the strength of the activation, even though labels are not used during training. Thus, an autoencoder with additional sparsity and diversity constraint can represent net fragments to some extent[4].

4: except that constraint 1, which says that net fragments span several layers by design, is violated (c.f. Section 4.3)

Figure B.3: 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, and blue means activation is off (i.e. 0).

Figure B.3 shows the activations of the "model 1". On the left are the activations without constraint, in the middle are the activations with sparsity constraint and on the right are the activations with sparsity and diversity constraints. For each of these three loss functions, the activations in the bottleneck layer are shown per class (activations along the vertical axis and classes along the horizontal axis). It is observable that adding sparsity and diversity constraints gives more meaning to the

individual neurons: While the same neurons are always active without both constraints, the activations are significantly more varied between the classes when both constraints are used. This becomes even more obvious when the activations of the bigger autoencoder "model 2" are investigated (c.f. Figure B.4 (a)).

Figure B.4: On the left side (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.

It is also examined what features the various neurons in the bottleneck layer represent. This is done by manually creating a bottleneck activation $\boldsymbol{h} = h_1, ..., h_{n_h}$ and feeding it through the decoder. To examine which feature the i-th neuron represents, h_i is set equal 1 and all other neurons are set equal 0. Figure B.4 (b) shows which feature the 18-th neuron represents. Together with the 33-th neuron, this neuron represents the number 5 and 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.

B.3 Conclusion

Different architectures have been trained with different targets on a very simple and easily interpretable dataset. The network activations are tracked during training and analysed for net fragments. Typical vision architectures are sequential and build up a composition of features (i.e. net fragments) over several layers. Usually, the embeddings of a pre-defined layer are used for downstream 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 encoder layer and thus extract these embeddings from the last model layer. Therefore, embeddings are extracted from one layer. Net fragments, on the other hand, span over multiple layers[5] and hence this principle is violated.

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

Furthermore, net fragments are strongly activated 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 input data and only change their activity strength. 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, adding a sparsity and diversity constraint alleviate this issue: Adding such constraints to the loss function encourages neurons to represent specific features typical for a subset 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). DOI: 10.48550/arXiv.2205.00002 (cited on pages 2, 21, 22, 26, 31, 38, 40, 42, 76).

[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] Jo S. Coombs, John C. Eccles, and Paul 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] Frank 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 7).

[9] George 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: arXiv:1412.6980 (Jan. 2017). DOI: 10.48550/arXiv.1412.6980 (cited on pages 8, 50, 59, 86, 90, 92).

[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: *International Conference on Learning Representations (ICLR)* (2021) (cited on pages 9, 38, 40, 57).

[14] Ilya O. Tolstikhin et al. 'MLP-Mixer: An All-MLP Architecture for Vision'. In: *Advances in Neural Information Processing Systems (NeurIPS)* 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'. en. In: *Artificial Intelligence Research*. Ed. by Aurona Gerber. Vol. 1342. Communications in Computer and Information Science. Cham: Springer International Publishing, 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: *International Conference on Machine Learning (ICML)*. PMLR. 2015, pp. 448–456 (cited on pages 10, 27).

[18] Yann 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, 11, 22, 23, 26, 52, 55, 61, 67, 86).

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012 (cited on pages 10, 11).

[20] Karen Simonyan and Andrew Zisserman. 'Very Deep Convolutional Networks for Large-Scale Image Recognition'. In: *International Conference on Learning Representations (ICLR)*. Computational and Biological Learning Society. 2015, pp. 1–14 (cited on pages 10, 11).

[21] Christian Szegedy et al. 'Going Deeper with Convolutions'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594 (cited on pages 10, 11, 89).

[22] Kaiming He et al. 'Deep Residual Learning for Image Recognition'. In: *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, 11, 89).

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 'U-Net: Convolutional Networks for Biomedical Image Segmentation'. en. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Vol. 9351. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28 (cited on pages 10, 11, 89).

[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, 11, 89).

[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, 11, 89).

[26] Joseph Redmon et al. 'You Only Look Once: Unified, Real-Time Object Detection'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91 (cited on pages 10, 11, 89).

[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). DOI: 10.48550/arXiv.1903.11816 (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 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 pages 11, 31).

[31] Gordon E. Moore. 'Cramming More Components onto Integrated Circuits'. In: *Electronics* 38.8 (1965), pp. 114–1116 (cited on page 11).

[32] Suhas Kumar. 'Fundamental Limits to Moore's Law'. In: arXiv:1511.05956 (Nov. 2015). DOI: 10.48550/arXiv.1511.05956 (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 E. 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*. Vol. 1. 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 (NeurIPS)*. Ed. by H. Larochelle et al. Vol. 33. Vancouver, BC, Canada: 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). DOI: 10.48550/arXiv.2201.11990 (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). DOI: 10.48550/arXiv.2004.09602 (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 12).

[40] Geoffrey E. Hinton, Oriol Vinyals, and Jeff Dean. 'Distilling the Knowledge in a Neural Network'. In: arXiv:1503.02531 (Mar. 2015). DOI: 10.48550/arXiv.1503.02531 (cited on page 12).

[41] Yu Zhang and Qiang Yang. 'A Survey on Multi-Task Learning'. In: *IEEE Transactions on Knowledge and Data Engineering* 34.12 (2021), pp. 5586–5609. DOI: 10.1109/TKDE.2021.3070203 (cited on page 12).

[42] Doyen Sahoo et al. 'Online Deep Learning: Learning Deep Neural Networks on the Fly'. en. In: *International Joint Conference on Artificial Intelligence (ICAJ)*. Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 2660–2666. DOI: 10.24963/ijcai.2018/369 (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). DOI: 10.48550/arXiv.1801.00631 (cited on page 12).

[46] Hans Moravec. *Mind Children: The Future of Robot and Human Intelligence*. en. Cambridge: Harvard University Press, 1995 (cited on page 12).

[47] Daniel J. Felleman and David 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] Donald O. Hebb. *The Organization of Behavior; A Neuropsychological Theory*. Oxford, England: Psychology Press, 1949 (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] Elie L. Bienenstock, Leon N. Cooper, and Paul W. Munro. 'Theory for the Development of Neuron Selectivity: Orientation Specificity and Binocular Interaction in Visual Cortex'. 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] Tim 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: *International Joint Conference on Neural Networks (IJCNN)*. Atlanta, USA, 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: *Workshop New Challenges in Neural Computation*. Citeseer. Mar. 2015, pp. 103–111 (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 15).

[57] John J. Hopfield. 'Neural Networks and Physical Systems with Emergent Collective Computational Abilities'. en. In: *Proceedings of the National Academy of Sciences (PNAS)* 79.8 (Apr. 1982), pp. 2554–2558. DOI: 10.1073/pnas.79.8.2554 (cited on pages 16, 17).

[58] Evelyn Fix and Joseph L. Hodges Jr. 'Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties'. In: *International Statistical Review* 57.3 (Dec. 1989), pp. 238–247. DOI: 10.2307/1403797 (cited on page 16).

[59] Jason Weston, Sumit Chopra, and Antoine Bordes. 'Memory Networks'. In: arXiv:1410.3916 (Nov. 2015). DOI: 10.48550/arXiv.1410.3916 (cited on page 16).

[60] John J. Hopfield, David I. Feinstein, and Richard 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).

[61] Robert J. McEliece et al. 'The Capacity of the Hopfield Associative Memory'. In: *IEEE Transactions on Information Theory* 33.4 (July 1987), pp. 461–482. DOI: 10.1109/TIT.1987.1057328 (cited on page 17).

[62] Dmitry Krotov and John J. Hopfield. 'Dense Associative Memory for Pattern Recognition'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by D. Lee et al. Vol. 29. 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). DOI: 10.48550/arXiv.2008.02217 (cited on page 17).

[65] Volodymyr Mnih et al. 'Recurrent Models of Visual Attention'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. NeurIPS'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] Thomas Cover and Peter 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] Larray F. Abbott. 'Lapicque's Introduction of the Integrate-and-Fire Model Neuron'. In: *Brain Research Bulletin* 50.5 (1999), pp. 303–304. DOI: https://doi.org/10.1016/S0361-9230(99)00161-6 (cited on page 18).

[70] Eugene 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: *IDIAP Research Report* 6.11 (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'. en. In: *Neural Networks* 99 (Mar. 2018), pp. 56–67. DOI: 10.1016/j.neunet.2017.12.005 (cited on pages 18, 25).

[75] Herbert Jäger. 'The" Echo State" Approach to Analysing and Training Recurrent Neural Networks-with an Erratum Note''. In: *German National Research Institute for Computer Science 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 pages 19, 20).

[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] Paul Erdös and Alfred 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] Andrei 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] David 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] David 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, 76).

[85] Christoph von der Malsburg and Elie 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 pages 21, 61).

[86] Christoph 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 and Christine A. Skarda. 'Representations: Who needs them?' In: (1990). Ed. by J. McGaugh, Jerry Weinberger, and G. Lynch (cited on page 21).

[88] David W. Arathorn. *Map-Seeking Circuits in Visual Cognition: A Computational Mechanism for Biological and Machine Vision*. Stanford, Califorina: 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, 76).

[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 (Apr. 1997), pp. 53–66. DOI: 10.1109/4235.585892 (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'. In: arXiv:1708.07747 (Sept. 2017). DOI: 10.48550/arXiv.1708.07747 (cited on page 23).

[95] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. Canadian Institute for Advanced Research, 2009 (cited on pages 23, 39).

[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. Wulff and J. A. Hertz. 'Learning Cellular Automaton Dynamics with Neural Networks'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by S. Hanson, J. Cowan, and C. Giles. Vol. 5. Morgan-Kaufmann, 1992 (cited on page 23).

[99] William Gilpin. 'Cellular Automata as Convolutional Neural Networks'. In: *Physical Review E* 100.3 (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). DOI: 10.23915/distill.00023 (cited on page 23).

[101] Alexander Mordvintsev et al. 'Growing Isotropic Neural Cellular Automata'. In: *Distill* (2020). DOI: 10.23915/distill.00023 (cited on page 23).

[102] Rasmus B. Palm et al. 'Variational Neural Cellular Automata'. In: arXiv:2201.12360 (Feb. 2022). DOI: 0.48550/arXiv.2201.12360 (cited on page 23).

[103] Diederik P. Kingma and Max Welling. 'Auto-Encoding Variational Bayes'. In: arXiv:1312.6114 (May 2014). DOI: 10.48550/arXiv.1312.6114 (cited on pages 23, 32, 57, 58).

[104] Shyam Sudhakaran et al. 'Growing 3D Artefacts and Functional Machines with Neural Cellular Automata'. In: arXiv:2103.08737 (June 2021). DOI: 10.48550/arXiv.2103.08737 (cited on page 23).

[105] Kazuya Horibe, Kathryn Walker, and Sebastian Risi. 'Regenerating Soft Robots Through Neural Cellular Automata'. en. In: *Genetic Programming*. Ed. by Ting Hu, Nuno Lourenço, and Eric Medvet. Vol. 12691. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 36–50. DOI: 10.1007/978-3-030-72812-0_3 (cited on page 23).

[106] Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. 'Learning Graph Cellular Automata'. In: 34 (2021). Ed. by M. Ranzato et al., pp. 20983–20994 (cited on page 23).

[107] Jie Zhou et al. 'Graph Neural Networks: A Review of Methods and Applications'. en. In: *AI Open* 1 (2020), pp. 57–81. DOI: 10.1016/j.aiopen.2021.01.001 (cited on page 23).

[108] Ettore Randazzo et al. 'Self-Classifying MNIST Digits'. In: *Distill* (2020). DOI: 10.23915/distill.00027.002 (cited on page 24).

[109] 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). DOI: 10.48550/arXiv.2106.15240 (cited on page 24).

[110] Volodymyr Mnih et al. 'Playing Atari with Deep Reinforcement Learning'. In: arXiv:1312.5602 (Dec. 2013). DOI: 10.48550/arXiv.1312.5602 (cited on page 24).

[111] Elias Najarro and Sebastian Risi. 'Meta-Learning through Hebbian Plasticity in Random Networks'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 20719–20731 (cited on pages 24, 85).

[112] Joachim W. 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*. June 2021, pp. 892–900. DOI: 10.1145/3449639.3459317 (cited on page 24).

[113] Louis Kirsch and Jürgen Schmidhuber. 'Meta Learning Backpropagation And Improving It'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. 2021, pp. 14122–14134 (cited on pages 24, 34).

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

[115] 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).

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

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

[118] 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).

[119] 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).

[120] 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).

[121] 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).

[122] 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).

[123] Timothée Masquelier and Simon J. Thorpe. 'Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity'. en. In: *PLoS Computational Biology* 3.2 (Feb. 2007). Ed. by Karl J Friston, e31. DOI: 10.1371/journal.pcbi.0030031 (cited on page 25).

[124] 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).

[125] Milad Mozafari et al. 'Bio-Inspired Digit Recognition Using Reward-Modulated Spike-Timing-Dependent Plasticity in Deep Convolutional Networks'. en. In: *Pattern Recognition* 94 (Oct. 2019), pp. 87–95. DOI: 10.1016/j.patcog.2019.05.015 (cited on page 25).

[126] Peter U. Diehl and Matthew Cook. 'Unsupervised Learning of Digit Recognition Using Spike-Timing-Dependent Plasticity'. In: *Frontiers in Computational Neuroscience* 9 (Aug. 2015). DOI: 10.3389/fncom.2015.00099 (cited on page 25).

[127] Yongqiang Cao, Yang Chen, and Deepak Khosla. 'Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition'. en. In: *International Journal of Computer Vision* 113.1 (May 2015), pp. 54–66. DOI: 10.1007/s11263-014-0788-3 (cited on page 25).

[128] Amirhossein Tavanaei and Anthony S. Maida. 'Bio-Inspired Spiking Convolutional Neural Network using Layer-wise Sparse Coding and STDP Learning'. In: arXiv:1611.03000 (June 2017). DOI: 10.48550/arXiv.1611.03000 (cited on page 25).

[129] 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)*. Killarney, Ireland: IEEE, July 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280696 (cited on page 25).

[130] Friedemann Zenke and Surya Ganguli. 'SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks'. en. In: *Neural Computation* 30.6 (June 2018), pp. 1514–1541. DOI: 10.1162/neco_a_01086 (cited on page 25).

[131] Paul Ferré, Franck Mamalet, and Simon J. Thorpe. 'Unsupervised Feature Learning With Winner-Takes-All Based STDP'. In: *Frontiers in Computational Neuroscience* 12 (Apr. 2018), p. 24. DOI: 10.3389/fncom.2018.00024 (cited on page 25).

[132] Milad Mozafari et al. 'First-Spike-Based Visual Categorization Using Reward-Modulated STDP'. In: *IEEE Transactions on Neural Networks and Learning Systems* 29.12 (Dec. 2018), pp. 6178–6190. DOI: 10.1109/TNNLS.2018.2826721 (cited on page 25).

[133] Guruprasad Raghavan, Cong Lin, and Matt Thomson. 'Self-Organization of Multi-Layer Spiking Neural Networks'. In: arXiv:2006.06902 (June 2020). DOI: 10.48550/arXiv.2006.06902 (cited on page 26).

[134] Guruprasad Raghavan and Matt Thomson. 'Neural Networks Grown and Self-Organized by Noise'. In: vol. 32. 2019 (cited on page 26).

[135] 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, 39).

[136] 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, 39).

[137] Jingzhao Zhang et al. 'Why Gradient Clipping Accelerates Training: A Theoretical Justification for Adaptivity'. In: arXiv:1905.11881 (Feb. 2020). DOI: 10.48550/arXiv.1905.11881 (cited on page 27).

[138] Chen-Yu Lee et al. 'Deeply-Supervised Nets'. In: *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).

[139] 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).

[140] 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).

[141] Read P. 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).

[142] 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).

[143] 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).

[144] 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).

[145] 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).

[146] 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).

[147] 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).

[148] Yulin Wang et al. 'Revisiting Locally Supervised Learning: an Alternative to End-to-end Training'. In: arXiv:2101.10832 (Jan. 2021). DOI: 10.48550/arXiv.2101.10832 (cited on page 27).

[149] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. 'Greedy Layerwise Learning Can Scale to ImageNet'. In: *International Conference on Machine Learning (ICML)*. PMLR. 2019, pp. 583–593 (cited on pages 27, 28).

[150] 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).

[151] 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).

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

[153] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. 'Decoupled Greedy Learning of CNNs'. In: *International Conference on Machine Learning (ICML)*. ICML'20. JMLR.org, 2020, pp. 736–745 (cited on page 28).

[154] 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, 54, 74).

[155] Geoffrey Hinton. 'How to Represent Part-Whole Hierarchies in a Neural Network'. In: arXiv:2102.12627 (Feb. 2021). DOI: 10.48550/arXiv.2102.12627 (cited on page 28).

[156] Yoshua Bengio. 'How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation'. In: arXiv:1407.7906 (Sept. 2014). DOI: 10.48550/arXiv.1407.7906 (cited on page 29).

[157] 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 page 30).

[158] Alexander Meulemans et al. 'A Theoretical Framework for Target Propagation'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. NeurIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020 (cited on page 30).

[159] Max Jaderberg et al. 'Decoupled Neural Interfaces using Synthetic Gradients'. In: *International Conference on Machine Learning (ICML)*. 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).

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

[161] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. 'Learning to Solve the Credit Assignment Problem'. In: arXiv:1906.00889 (Apr. 2020). DOI: 10.48550/arXiv.1906.00889 (cited on page 30).

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

[163] 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 pages 30, 31, 39).

[164] Will Xiao et al. 'Biologically-Plausible Learning Algorithms Can Scale to Large Datasets'. In: *International Conference on Learning Representations, (ICLR)*. 2019 (cited on page 30).

[165] David Balduzzi, Hastagiri Vanchinathan, and Joachim Buhmann. 'Kickback Cuts Backprop's Red-Tape: Biologically Plausible Credit Assignment in Neural Networks'. In: *AAAI Conference on Artificial Intelligence*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, 2015, pp. 485–491 (cited on page 30).

[166] Qianli Liao, Joel Z. Leibo, and Tomaso Poggio. 'How Important is Weight Symmetry in Backpropagation?' In: *AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, 2016, pp. 1837–1844 (cited on pages 30, 31).

[167] Sergey Bartunov et al. 'Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures'. In: *Advances in Neural Information Processing Systems (NeurIPS)* 31 (2018) (cited on pages 30, 31, 39, 47, 55).

[168] Miguel Carreira-Perpinan and Weiran Wang. 'Distributed Optimization of Deeply Nested Systems'. In: *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 31).

[169] Gavin Taylor et al. 'Training Neural Networks without Gradients: A Scalable ADMM Approach'. In: *International Conference on Machine Learning (ICML)*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 2722–2731 (cited on page 31).

[170] Ziming Zhang and Matthew Brand. 'Convergent Block Coordinate Descent for Training Tikhonov Regularized Deep Neural Networks'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1719–1728 (cited on page 31).

[171] Tim Tsz-Kit Lau et al. 'A Proximal Block Coordinate Descent Algorithm for Deep Neural Network Training'. In: arXiv:1803.09082 (Mar. 2018). DOI: 10.48550/arXiv.1803.09082 (cited on page 31).

[172] 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). DOI: 10.48550/arXiv.2101.03419 (cited on pages 31, 47).

[173] Pierre Baldi. 'Autoencoders, Unsupervised Learning, and Deep Architectures'. In: *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).

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

[175] Quoc V. Le et al. 'Building High-Level Features Using Large Scale Unsupervised Learning'. In: *International Coference on International Conference on Machine Learning (ICML)*. ICML'12. Edinburgh, Scotland: Omnipress, 2012, pp. 507–514 (cited on pages 32, 48, 93).

[176] Pascal Vincent et al. 'Extracting and Composing Robust Features with Denoising Autoencoders'. In: *International Conference on Machine Learning (ICML)*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294 (cited on page 32).

[177] Salah Rifai et al. 'Contractive Auto-Encoders: Explicit Invariance during Feature Extraction'. In: *International Conference on International Conference on Machine Learning (ICML)*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 833–840 (cited on page 32).

[178] 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).

[179] 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).

[180] Kaiming He et al. 'Masked Autoencoders Are Scalable Vision Learners'. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA: IEEE, June 2022, pp. 15979–15988. DOI: 10.1109/CVPR52688.2022.01553 (cited on page 32).

[181] Yuge Shi et al. 'Adversarial Masking for Self-Supervised Learning'. In: *International Conference on Machine Learning (ICML)*. PMLR. 2022, pp. 20026–20040 (cited on page 32).

[182] Nikos Komodakis and Spyros Gidaris. 'Unsupervised Representation Learning by Predicting Image Rotations'. In: *International Conference on Learning Representations (ICLR)*. 2018 (cited on page 32).

[183] Xiaohua Zhai et al. 'S4L: Self-Supervised Semi-Supervised Learning'. In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. Seoul, Korea (South): IEEE, Oct. 2019, pp. 1476–1485. DOI: 10.1109/ICCV.2019.00156 (cited on page 32).

[184] 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).

[185] Ting Chen et al. 'A Simple Framework for Contrastive Learning of Visual Representations'. In: *International Conference on Machine Learning (ICML)*. ICML'20. JMLR.org, 2020, 149, pp. 1597–1607 (cited on pages 32, 44).

[186] Kaiming He et al. 'Momentum Contrast for Unsupervised Visual Representation Learning'. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 9729–9738 (cited on page 32).

[187] Mathilde Caron et al. 'Unsupervised Learning of Visual Features by Contrasting Cluster Assignments'. In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 9912–9924 (cited on pages 32, 44).

[188] Prannay Khosla et al. 'Supervised Contrastive Learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 18661–18673 (cited on page 32).

[189] 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 page 33).

[190] 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 33).

[191] 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 33).

[192] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 'Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks'. In: *International Conference on Machine Learning (ICML)*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 1126–1135 (cited on pages 33–35).

[193] Chelsea Finn, Kelvin Xu, and Sergey Levine. 'Probabilistic Model-Agnostic Meta-Learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. NeurIPS'18. Montreal, Canada: Curran Associates Inc., 2018, pp. 9537–9548 (cited on page 33).

[194] Yoonho Lee and Seungjin Choi. 'Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace'. In: *International Conference on Machine Learning (ICML)*. PMLR. 2018, pp. 2927–2936 (cited on page 33).

[195] Siyuan Qiao et al. 'Few-Shot Image Recognition by Predicting Parameters from Activations'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 7229–7238 (cited on pages 33, 34).

[196] Andrei A. Rusu et al. 'Meta-Learning with Latent Embedding Optimization'. In: arXiv:1807.05960 (Mar. 2019). DOI: 10.48550/arXiv.1807.05960 (cited on page 33).

[197] Tom M. Heskes. 'Empirical Bayes for Learning to Learn'. In: (2000) (cited on page 34).

[198] James Requeima et al. 'Fast and Flexible Multi-Task Classification Using Conditional Neural Adaptive Processes'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Red Hook, NY, USA: Curran Associates Inc., 2019 (cited on page 34).

[199] David Ha, Andrew Dai, and Quoc V. Le. 'HyperNetworks'. In: arXiv:1609.09106 (Dec. 2016). DOI: 10.48550/arXiv.1609.09106 (cited on page 34).

[200] Jake Snell, Kevin Swersky, and Richard Zemel. 'Prototypical Networks for Few-Shot Learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. NeurIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 4080–4090 (cited on pages 34, 35).

[201] Wei-Yu Chen et al. 'A Closer Look at Few-shot Classification'. In: arXiv:1904.04232 (Jan. 2020). DOI: 10.48550/arXiv.1904.04232 (cited on page 34).

[202] Sachin Ravi and Hugo Larochelle. 'Optimization as a Model for Few-Shot Learning'. In: *International Conference on Learning Representations (ICLR)*. 2017 (cited on pages 34, 35).

[203] Ke Li and Jitendra Malik. 'Learning to Optimize'. In: arXiv:1606.01885 (June 2016). DOI: 10.48550/arXiv.1606.01885 (cited on page 34).

[204] Zhenguo Li et al. 'Meta-SGD: Learning to Learn Quickly for Few-Shot Learning'. In: arXiv:1707.09835 (Sept. 2017). DOI: 10.48550/arXiv.1707.09835 (cited on page 34).

[205] Eunbyung Park and Junier B. Oliva. 'Meta-Curvature'. In: arXiv:1902.03356 (Jan. 2020). DOI: 10.48550/arXiv.1902.03356 (cited on page 34).

[206] Rein Houthooft et al. 'Evolved Policy Gradients'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018 (cited on page 34).

[207] Flood Sung et al. 'Learning to Learn: Meta-Critic Networks for Sample Efficient Learning'. In: arXiv:1706.09529 (June 2017). DOI: 10.48550/arXiv.1706.09529 (cited on page 34).

[208] Giulia Denevi et al. 'Learning To Learn Around A Common Mean'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018 (cited on page 34).

[209] Giulia Denevi et al. 'Online-Within-Online Meta-Learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019 (cited on page 34).

[210] 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 34).

[211] Yiying Li et al. 'Feature-Critic Networks for Heterogeneous Domain Generalization'. In: *International Conference on Machine Learning (ICML)*. PMLR. 2019, pp. 3915–3924 (cited on pages 34, 35).

[212] Antreas Antoniou and Amos J. Storkey. 'Learning to Learn By Self-Critique'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019 (cited on page 34).

[213] Rinu Boney and Alexander Ilin. 'Semi-Supervised Few-Shot Learning with MAML'. In: *International Conference on Learning Representations (ICLR)*. 2018 (cited on page 34).

[214] Barret Zoph and Quoc V. Le. 'Neural Architecture Search with Reinforcement Learning'. In: arXiv:1611.01578 (Feb. 2017). DOI: 10.48550/arXiv.1611.01578 (cited on page 34).

[215] Justin Bayer et al. 'Evolving Memory Cell Structures for Sequence Learning'. In: *International Conference on Artificial Neural Networks (ICANN)*. 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 34).

[216] Esteban Real et al. 'Regularized Evolution for Image Classifier Architecture Search'. In: *Conference on Artificial Intelligence (AAAI)*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. DOI: 10.1609/aaai.v33i01.33014780 (cited on page 34).

[217] Luca Franceschi et al. 'Bilevel Programming for Hyperparameter Optimization and Meta-Learning'. In: *International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 1568–1577 (cited on page 34).

[218] Paul Micaelli and Amos Storkey. 'Gradient-based Hyperparameter Optimization Over Long Horizons'. In: arXiv:2007.07869 (Sept. 2021). DOI: 10.48550/arXiv.2007.07869 (cited on page 34).

[219] Luca Franceschi et al. 'Forward and Reverse Gradient-Based Hyperparameter Optimization'. In: *International Conference on Machine Learning (ICML)*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 1165–1173 (cited on pages 34, 35).

[220] Ekin D. Cubuk et al. 'AutoAugment: Learning Augmentation Strategies From Data'. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019, pp. 113–123 (cited on page 34).

[221] 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 34).

[222] Cheng Zhang et al. 'Active Mini-Batch Sampling Using Repulsive Point Processes'. In: *Conference on Artificial Intelligence (AAAI)*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. DOI: 10.1609/aaai.v33i01.33015741 (cited on page 34).

[223] Yang Fan et al. 'Learning to Teach'. In: arXiv:1805.03643 (May 2018). DOI: 10.48550/arXiv.1805.03643 (cited on page 34).

[224] Tongzhou Wang et al. 'Dataset Distillation'. In: arXiv:1811.10959 (Feb. 2020). DOI: 10.48550/arXiv.1811.10959 (cited on page 34).

[225] Jonathan Lorraine, Paul Vicol, and David Duvenaud. 'Optimizing Millions of Hyperparameters by Implicit Differentiation'. In: *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 page 34).

[226] 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).

[227] Nataniel Ruiz, Samuel Schulter, and Manmohan Chandraker. 'Learning To Simulate'. In: arXiv:1810.02513 (May 2019). DOI: 10.48550/arXiv.1810.02513 (cited on page 34).

[228] 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). DOI: 10.48550/arXiv.1903.11774 (cited on page 34).

[229] 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).

[230] Yan Duan et al. 'RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning'. In: arXiv:1611.02779 (Nov. 2016). DOI: 10.48550/arXiv.1611.02779 (cited on page 34).

[231] Jürgen 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 35).

[232] 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 35).

[233] Tim Salimans et al. 'Evolution Strategies as a Scalable Alternative to Reinforcement Learning'. In: arXiv:1703.03864 (Sept. 2017). DOI: 10.48550/arXiv.1703.03864 (cited on pages 35, 85).

[234] Olga Wichrowska et al. 'Learned Optimizers That Scale and Generalize'. In: *International Conference on Machine Learning (ICML)*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 3751–3760 (cited on page 35).

[235] Antreas Antoniou, Harrison Edwards, and Amos Storkey. 'How to train your MAML'. In: arXiv:1810.09502 (Mar. 2019). DOI: 10.48550/arXiv.1810.09502 (cited on page 35).

[236] Scott J. A. Kelso. *Dynamic Patterns: The Self-Organization of Brain and Behavior*. The MIT Press, Jan. 1995 (cited on page 38).

[237] Birgitta Dresp-Langley. 'Seven Properties of Self-Organization in the Human Brain'. In: *Big Data and Cognitive Computing* 4.2 (2020), p. 10 (cited on page 38).

[238] 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 39).

[239] 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 39).

[240] Yann LeCun. '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 39).

[241] Jia Deng et al. 'ImageNet: A Large-Scale Hierarchical Image Database'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2009, pp. 248–255 (cited on page 39).

[242] 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 39, 87).

[243] Yu-An Chung et al. 'An Unsupervised Autoregressive Model for Speech Representation Learning'. In: *Interspeech*. 2019, pp. 146–150. DOI: 10.21437/Interspeech.2019-1473 (cited on page 41).

[244] 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 42).

[245] Christos Louizos, Max Welling, and Diederik P. Kingma. 'Learning Sparse Neural Networks through L_0 Regularization'. In: arXiv:1712.01312 (June 2018). DOI: 10.48550/arXiv.1712.01312 (cited on page 42).

[246] Torsten Hoefler et al. 'Sparsity in Deep Learning: Pruning and Growth for Efficient Inference and Training in Neural Networks'. In: *Journal of Machine Learning Research* 22.1, 241 (Jan. 2021), pp. 10882–11005 (cited on page 42).

[247] Konstantinos P. Panousis, Sotirios Chatzis, and Sergios Theodoridis. 'Stochastic Local Winner-Takes-All Networks Enable Profound Adversarial Robustness'. In: arXiv:2112.02671 (Dec. 2021). DOI: 10.48550/arXiv.2112.02671 (cited on page 42).

[248] 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 42).

[249] Ting Chen et al. 'Big Self-Supervised Models Are Strong Semi-Supervised Learners'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. NeurIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020 (cited on page 44).

[250] 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 44, 78).

[251] Jan Storck, Sepp Hochreiter, Jürgen Schmidhuber, et al. 'Reinforcement driven information acquisition in non-deterministic environments'. In: *International Conference on Artificial Neural Networks (ICANN)*. Vol. 2. 1995, pp. 159–164 (cited on pages 45, 80).

[252] Magdalena Klapper-Rybicka, Nicol N. Schraudolph, and Jürgen Schmidhuber. 'Unsupervised Learning in LSTM Recurrent Neural Networks'. In: *International Conference on Artificial Neural Networks (ICANN)*. 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 45).

[253] Vassileios Balntas et al. 'Learning Local Feature Descriptors with Triplets and Shallow Convolutional Neural Networks'. en. In: *Procedings of the British Machine Vision Conference*. York, UK: British Machine Vision Association, 2016, pp. 119.1–119.11. DOI: 10.5244/C.30.119 (cited on page 49).

[254] Norman Mu and Justin Gilmer. 'MNIST-C: A Robustness Benchmark for Computer Vision'. In: arXiv:1906.02337 (June 2019). DOI: 10.48550/arXiv.1906.02337 (cited on pages 55, 67, 87).

[255] Samuel R. Bowman et al. 'Generating Sentences from a Continuous Space'. In: *SIGNLL Conference on Computational Natural Language Learning (CoNLL)*. Association for Computational Linguistics (ACL). 2016, pp. 10–21 (cited on page 59).

[256] 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 59).

[257] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 'Neural Discrete Representation Learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017 (cited on page 67).

[258] Laurens van der Maaten and Geoffrey Hinton. 'Visualizing Data using t-SNE'. In: *Journal of Machine Learning Research* 9.11 (2008) (cited on page 69).

[259] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. en. The MIT Press, 2010 (cited on pages 75, 76).

[260] 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 76).

[261] Laurenz Wiskott and Christoph von der Malsburg. *Face Recognition by Dynamic Link Matching*. Ruhr-Univ., Inst. für Neuroinformatik, 1996 (cited on page 76).

[262] Wolfgang Köhler. 'Gestalt Psychology'. In: (1929) (cited on page 76).

[263] Junuk Jung et al. 'Unified Negative Pair Generation toward Well-Discriminative Feature Space for Face Recognition'. In: *British Machine Vision Conference 2022 (BMVC)*. BMVA Press, 2022, p. 421 (cited on page 78).

[264] Hamza Keurti et al. 'Homomorphism Autoencoder – Learning Group Structured Representations from Observed Transitions'. In: arXiv:2207.12067 (July 2022). DOI: 10.48550/arXiv.2207.12067 (cited on page 78).

[265] Jean Piaget. 'Part I: Cognitive Development in Children: 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 78).

[266] Yann LeCun. 'A Path Towards Autonomous Machine Intelligence'. In: *Open Review* 62 (June 2022) (cited on pages 78, 80, 93).

[267] Emmanuel Bengio et al. 'Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 27381–27394 (cited on page 80).

[268] 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 85).

[269] Tailin Liang et al. 'Pruning and Quantization for Deep Neural Network Acceleration: A Survey'. en. In: *Neurocomputing* 461 (Oct. 2021), pp. 370–403. DOI: 10.1016/j.neucom.2021.07.045 (cited on page 87).