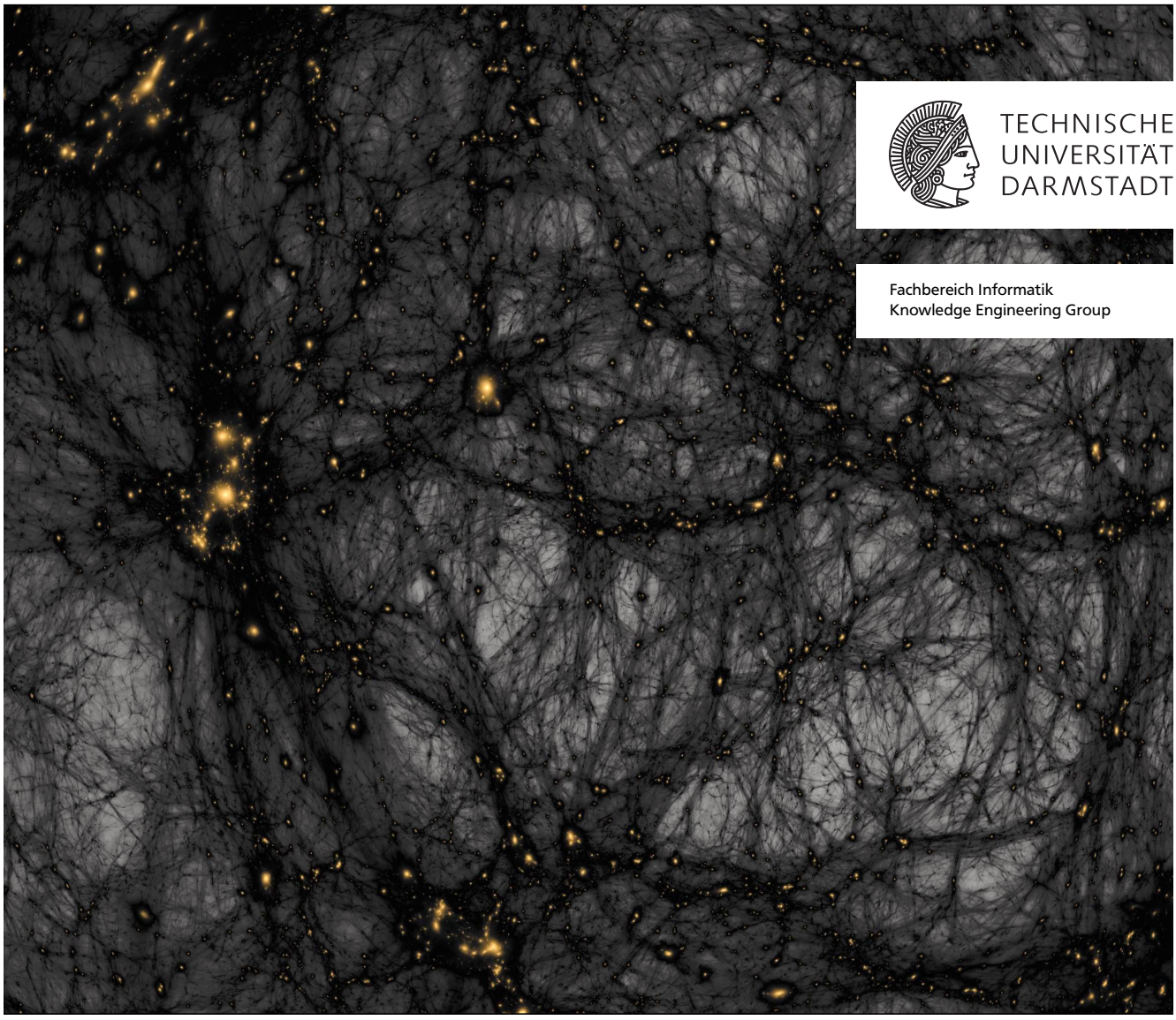
Self-Imitation Regularization: Regularizing Neural Networks by Leveraging Their Dark Knowledge

Self-Imitation Regularization:
Regularisierung Neuronaler Netze durch Verwendung ihres Dark Knowledges

Bachelor-Thesis von Jonas Jäger aus Mannheim

Juni 2019



TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

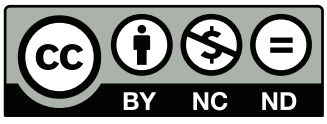
Image: Dark Matter in a Simulated Universe; Illustration Credit & Copyright: Tom Abel & Ralf Kaehler (KIPAC, SLAC), AMNH;
Published on APOD (`https://apod.nasa.gov/apod/ap171031.html`, as of February 4, 2019).
Rights of use obtained from the copyright holder, Tom Abel (on May 9, 2019). Special thanks for that.

Self-Imitation Regularization:
Regularizing Neural Networks by Leveraging Their Dark Knowledge
Self-Imitation Regularization:
Regularisierung Neuronaler Netze durch Verwendung ihres Dark Knowledges

Vorgelegte Bachelor-Thesis von Jonas Jäger aus Mannheim

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza Mencía

Tag der Einreichung:



The universe is made mostly of dark matter and dark energy,
and we don't know what either of them is.

Saul Perlmutter, 2011 Nobel laureate in Physics

Physicists will tell you that 95% of the stuff in the universe is not what they were paying attention to – it is something else.
[. . .] The point of [dark knowledge] that I want to show you [is] that 95% of what a neural network learns is not what you are trying to get it to learn and is not what you test it on – it is something else.[a]

Geoffrey Hinton, 2018 Turing recipient
for his research on artificial neural networks

[a] He began his talk with these words, in the Excellent Lecture Series at TTIC (Toyota Technological Institute at Chicago) given on October 2, 2014. `http://www.ttic.edu/dls-2014-2015/` (as of January 9, 2019)

Abstract

Deep Learning, the learning of deep neural networks, is nowadays indispensable not only in the fields of computer science and information technology but also in innumerable areas of daily life. It is one of the key technologies in the development of artificial intelligence and will continue to be of great importance in the future, e.g., in the development of autonomous driving. Since the data for learning such (deep) neural networks is clearly limited and therefore the neural network cannot be prepared for all possible data which have to be handled in real-life situations, a solid generalization capability is necessary. This means the ability to acquire a general concept from the training data, so that the task associated with the data is properly understood and the training data is not simply memorized. An essential component behind such a generalization capability is regularization.

A regularization procedure causes a neural network to generalize (better) when learning from data. Various, commonly and widespread used regularization procedures exist, which, for example, limit the weights in the neural network (parameters by whose adaptation learning takes place) or temporarily change its structure, and thus implicitly aim for the neural network to make better predictions on as yet unseen data.

Self-Imitation Regularization (SIR) is presented in this thesis and is an easy to implement regularization procedure which, in contrast to the established standard regularization procedures, explicitly addresses the actual objective – the formation of predictions – and only implicitly influences the weights/parameters in the neural network. The existing (dark) knowledge of the learning neural network is used and explicitly involved in the learning principles (i.e., the error function to be minimized) of the neural network. Since this is one's own knowledge, which, in turn, is made available during learning, this can be seen as a form of self-imitation. For a given data example, the (dark) knowledge contains, on the one hand, information about the similarities to other classes (in a classification problem, a neural network predicts classes and classifies the data). On the other hand, it quantifies (relative to other data examples) the confidence of the neural network in the prediction for this data example. Intuitively, through self-imitation, this information induces a questioning behavior regarding the correctness of the given solutions in the training data as well as deepens the understanding of correlations and similarities between the classes.

Besides the regularization ability, which has strong guarantees of success (partially under statistical significance), the use of SIR stabilizes the training, increases the data efficiency and is resistant to erroneous data labels, which was demonstrated in various experiments. It is also applicable to very deep neural network architectures and can be combined with some standard regularization methods (i.e., dropout and maxnorm regularization). The implementation and additional computation effort is very low while the hyperparameter tuning is simple.

In this thesis, experimental results on the use of SIR are analyzed and evaluated using several procedures, and the learned neural networks are examined closely in order to explain the regularization behavior along with accompanying properties.

Zusammenfassung

Deep Learning, das Lernen von tiefen Neuronalen Netzen, ist heutzutage nicht nur aus der Informatik und Informationstechnologie, sondern auch aus unzähligen Bereichen des Alltags nicht mehr wegzudenken. Es bildet eine der Schlüsseltechnologien in der Entwicklung von Künstlicher Intelligenz und wird auch für die Zukunft von großer Bedeutung sein, wie bspw. in der Entwicklung des Autonomen Fahrens. Da die Daten zum Lernen von solchen (tiefen) Neuronalen Netzen natürlich begrenzt sind und dieses somit nicht auf sämtliche mögliche Daten, die im späteren realen Einsatz verarbeiten werden müssen, vorbereitet werden kann, ist eine solide Generalisierungsfähigkeit notwendig. Also die Fähigkeit ein generelles Konzept aus den Lerndaten zu lernen, damit die mit den Daten verbundene Aufgabe wirklich verstanden wird und die präsentierten Daten nicht einfach nur auswendig gelernt werden. Hinter einer solchen Generalisierungsfähigkeit steckt eine wesentliche Technik: Regularisierung.

Ein Regularisierungsverfahren bewirkt das (bessere) Generalisieren eines Neuronalen Netzes beim Lernen aus Daten. Diverse, viel eingesetzte und weit verbreitete Regularisierungsverfahren existieren, die bspw. die Gewichte im Neuronalen Netz beschränken (Parameter durch deren Anpassung das Lernen stattfindet) oder dessen Struktur temporär verändern und damit implizit bezweckt werden soll, dass das Neuronale Netz bessere Vorhersagen auf ungesehenen Daten trifft.

Self-Imitation Regularization (SIR) wird in dieser Thesis vorgestellt und ist ein einfach zu implementierendes Regularisierungsverfahren, das im Gegensatz zu den etablierten Standardregularisierungsverfahren explizit am eigentlichen Ziel – der Formung der Vorhersagen – ansetzt und nur implizit die Gewichte/Parameter im Neuronalen Netz beeinflusst. Dabei wird das vorhandene (Dark) Knowledge des lernenden Neuronalen Netzes genutzt und explizit in die Lernprinzipien (d.h. die Fehlerfunktion, die minimiert werden soll) des Neuronalen Netzes involviert. Da es sich um das eigene Wissen handelt, was nun wiederum beim Lernen zusätzlich zur Verfügung gestellt wird, kann dies als eine Form von Selbstimitation gesehen werden. Das (Dark) Knowledge beinhaltet für ein gegebenes Datenbeispiel zum einen Informationen über die Ähnlichkeiten zu anderen Klassen (in einem Klassifizierungsproblem sagt ein Neuronales Netz Klassen vorher, um damit die Daten zu klassifizieren). Zum anderen quantifiziert dies (relativ zu anderen Datenbeispielen) darüber, wie sicher sich das Neuronale Netz beim Treffen der Vorhersage zu diesem Datenbeispiel ist. Intuitiv gesehen wird durch diese Informationen mittels der Selbst-Imitation sowohl ein hinterfragendes Verhalten bzgl. der Korrektheit der vorgegebenen Lösungen in den Lerndaten hervorgerufen, als auch das Verständnis über Zusammenhänge und Ähnlichkeiten zwischen den Klassen vertieft.

Neben der eigentlichen Regularisierungsfähigkeit, die starke Erfolgsgarantien (teilweise mit statistischer Signifikanz) aufweist, stabilisiert der Einsatz von SIR das Training, erhöht die Dateneffizienz und erweist sich als resistent gegen fehlerhafte Datenlabel, was in diversen Experimenten gezeigt werden konnte. Auch ist es auf sehr tiefen Netzarchitekturen anwendbar und kann mit manchen Standardregularisierungsmethoden (Dropout und Maxnorm) kombiniert werden. Der Implementierungs- sowie zusätzliche Berechnungsaufwand ist sehr gering und das Tunen der Hyperparameter ist einfach.

In der Thesis werden experimentelle Ergebnisse zum Einsatz von SIR mit verschiedenen Verfahren ausgewertet und evaluiert, sowie die gelernten Neuronalen Netze genauer untersucht, um das Regularisierungsverhalten samt einhergehender Eigenschaften zu erklären.

Contents

List of Figures **11**

List of Tables **13**

Notation **15**

1. Introduction **17**

2. Background **19**
2.1. Machine Learning . . . 19
2.1.1. Categories of Machine Learning . . . 19
2.1.2. Algorithms in Machine Learning . . . 23
2.1.3. Expressive power and separability . . . 24
2.1.4. Generalization . . . 25
2.2. Artificial Neural Networks . . . 26
2.2.1. The neuron . . . 28
2.2.2. Layer of neurons . . . 29
2.2.3. Multi-Layer Perceptron (MLP); or, stacking layers of neurons . . . 31
2.2.4. Gradient-based learning . . . 33
2.3. Convolutional Neural Networks (CNNs) . . . 37
2.3.1. Convolutional layer . . . 38
2.3.2. Pooling layer . . . 39
2.4. Activation functions . . . 40
2.4.1. Sigmoidal activation functions . . . 41
2.4.2. Rectifier activation function variants . . . 41
2.5. Loss functions . . . 43
2.5.1. Mean squared error . . . 44
2.5.2. Cross entropy loss . . . 44
2.6. Softmax . . . 47
2.6.1. Temperature Softmax . . . 51
2.6.2. Combination of softmax and cross entropy loss . . . 52
2.7. Regularization . . . 53
2.7.1. Parameter norm penalty regularization . . . 54
2.7.2. Maxnorm constraint . . . 55
2.7.3. Dropout . . . 56
2.8. Knowledge Distillation and Dark Knowledge . . . 57

3. Self-Imitation Regularization (SIR) **61**
3.1. Hypotheses . . . 64

3.2. Further variants of Self-Imitation Regularization (SIR) . . . 67
3.2.1. Penalty formulation . . . 67
3.2.2. Target network . . . 69
3.3. Preliminary studies in Knowledge Distillation (KD) . . . 70
3.3.1. Comparison: temperature and the chained softmax function . . . 70
3.3.2. Comparison: Symmetric and asymmetric imitation . . . 71

4. Experiments **73**
4.1. Experimental set-ups . . . 73
4.1.1. Noisy data labels . . . 73
4.1.2. Learning on a small training data subset . . . 73
4.1.3. Reduced training samples of a class . . . 74
4.1.4. Standard regularization methods . . . 74
4.2. Data sets . . . 74
4.3. Neural network architectures . . . 77
4.4. Evaluation methodologies . . . 79
4.4.1. Metrics . . . 79
4.4.2. Statistical tests . . . 83

5. Experimental results **87**
5.1. Regularization capability . . . 87
5.1.1. Regularization ability with other regularizers being present . . . 91
5.1.2. Overfitting in the later training course . . . 93
5.2. On the functioning of Self-Imitation Regularization (SIR) . . . 94
5.2.1. Predictions and soft targets investigation . . . 94
5.2.2. Parameters investigation . . . 98
5.2.3. (Ranking) quality of the soft targets/predictions . . . 101
5.3. Noisy data label compensation . . . 104
5.3.1. Robust learning despite noisy labels . . . 104
5.3.2. Correcting ability of noisy training labels . . . 110
5.4. Learning from other classes . . . 113
5.5. Training stabilization . . . 115
5.6. Depth of a neural network . . . 116
5.7. Pre-fitting . . . 119

6. Conclusion **121**
6.1. Future work . . . 124
6.1.1. Further investigations . . . 125
6.1.2. Extensions and improvements of Self-Imitation Regularization (SIR) . . . 126

Acknowledgments **129**

Bibliography **131**

A. Experimental results **139**
A.1. Statistical tests raw performance scores 139
A.1.1. Testing Self-Imitation regularized against unregularized neural networks 139
A.1.2. Testing neural networks with SIR combined with a standard regularizer against this standard regularizer on its own 141
A.2. Noisy data labels results 145

List of Figures

2.1. Separability in classification problems . . . 24
2.2. Generalization behavior in learning curves . . . 25
2.3. Artificial neuron: a schematic visualization . . . 30
2.4. Artificial neural network or Multi-Layer Perceptron (MLP): a schematic visualization . . . 31
2.5. Exemplary application of the backpropagation algorithm on a computational graph . . . 36
2.6. Exemplary visualization of the convolution operation . . . 38
2.7. Exemplary visualization of max pooling . . . 39
2.8. Activation functions . . . 40
2.9. Plot of the negative log probability mapping $p \mapsto -\log(p)$ . . . 45
2.10. Softmax function mapping properties . . . 48
2.11. Temperature softmax function mapping properties (on different temperatures) . . . 52
2.12. Knowledge Distillation (KD) framework diagram . . . 59

3.1. Self-Imitation Regularization (SIR) framework diagram . . . 62
3.2. ζ-chain softmax function mapping properties (on different values for ζ) . . . 63
3.3. Preliminary KD experiments: comparison between temperature and chained softmax function . . . 71

4.1. Exemplary samples from the (visual) training data sets . . . 78

5.1. Differences in test accuracy on multiple data sets between self-imitation regularized neural networks compared to unregularized ones . . . 89
5.2. Mean test accuracy of different SIR settings on multiple data sets plotted over the imitation rate π . . . 90
5.3. Learning curves on the CIFAR-10 data set and differently regularized neural networks . . . 93
5.4. Best test accuracy achieved while learning compared to the final test accuracy . . . 94
5.5. Mean normalized entropy of (prediction) distributions from learned neural networks . . . 95
5.6. Exemplary prediction distributions of two learned neural networks (SIR vs. unregularized) 97
5.7. Mean lengths (L^1-norm and L^2-norm) of parameter vectors . . . 99
5.8. Average softmax distributions per class in heat maps on the CIFAR-10 test data set . . . 101
5.9. Test accuracy plotted over different amounts of label noise . . . 104
5.10. Confusion matrices on the CIFAR-10 test data set after training with 50% label noise . . . 105
5.11. Mean test accuracy (to different degrees of label noise) plotted over the imitation rate π . 106
5.12. Best test accuracy achieved while learning on different amounts of label noise . . . 107
5.13. Mean test accuracy plotted to different degrees of label noise in comparison with standard regularization methods . . . 108
5.14. Peak test accuracy plotted to different degrees of label noise in combination with standard regularization methods . . . 109
5.15. Correction ability of the noisified training data samples (in accuracy) . . . 110
5.16. Correction ability of the noisified training data samples (in position error) . . . 111

5.17. Incorrect label uncertainty factors on training data with label noise 112
5.18. Performance for class *dog* whose samples had been reduced in the training set to different degrees 114
5.19. SIR on the CIFAR-10 data set with the `ResNet` architecture at different depths 117
5.20. SIR on the CIFAR-10 data set (with label noise) with the `ResNet` architecture at different depths 118

List of Tables

3.1. Preliminary KD experiments: comparison between symmetric and asymmetric imitation . 72

4.1. Information about the data sets 75
4.2. Information about the experimental neural network architectures 77

5.1. Results from the statistical tests for different experimental configurations over multiple data sets comparing the performance of self-imitation regularized neural networks to unregularized ones 88
5.2. Further statistical tests with a different SIR hyperparameter setting for different experimental configurations over multiple data sets comparing the performance of self-imitation regularized neural networks to unregularized ones 90
5.3. Results from the statistical tests for different experimental configurations over multiple data sets comparing the performance of neural networks with SIR and a standard regularizer to a neural network regularized solely with this standard regularizer 92
5.4. Evaluation metrics regarding the ranking quality of soft predictions 103

A.1. Performance scores and pair-wise differences used for statistical testing (SIR with target gradients) 139
A.2. Performance scores and pair-wise differences used for statistical testing (SIR without target gradients) 140
A.3. Performance scores and pair-wise differences used for statistical testing comparing SIR combined with L^1-norm reg. to solely L^1-norm reg. (SIR with target gradients) 141
A.4. Performance scores and pair-wise differences used for statistical testing comparing SIR combined with L^2-norm reg. to solely L^2-norm reg. (SIR with target gradients) 141
A.5. Performance scores and pair-wise differences used for statistical testing comparing SIR combined with maxnorm to solely maxnorm (SIR with target gradients) 142
A.6. Performance scores and pair-wise differences used for statistical testing comparing SIR combined with dropout to solely dropout (SIR with target gradients) 142
A.7. Performance scores and pair-wise differences used for statistical testing comparing SIR combined with L^1-norm reg. to solely L^1-norm reg. (SIR without target gradients) 143
A.8. Performance scores and pair-wise differences used for statistical testing comparing SIR combined with L^2-norm reg. to solely L^2-norm reg. (SIR without target gradients) 143
A.9. Performance scores and pair-wise differences used for statistical testing comparing SIR combined with maxnorm to solely maxnorm (SIR without target gradients) 144
A.10. Performance scores and pair-wise differences used for statistical testing comparing SIR combined with dropout to solely dropout (SIR without target gradients) 144
A.11. Label noise performance scores 145
A.12. Label noise performance scores 146

Notation

Math notation:

The math notation does mainly follow the notation used in Petersen et al. (2004) and Goodfellow et al. (2016).

$\boldsymbol{A}$	Matrix
$\boldsymbol{A}_i$	Matrix indexed for some purpose
$\boldsymbol{A}^{(i)}$	Matrix indexed for some purpose
$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{I}$	Identity matrix (i.e., ones on the diagonal and zeros elsewhere)
$\boldsymbol{a}$	Vector
$\boldsymbol{a}_i$	Vector indexed for some purpose
$\boldsymbol{a}^{(i)}$	Vector indexed for some purpose
$\boldsymbol{a}^\top$	Transpose of vector $\boldsymbol{a}$
$\mathbb{1}$	All-ones vector (i.e., every element is equal to one)
$\|\boldsymbol{a}\|_a$	a-norm of vector $\boldsymbol{a}$
a	Scalar
$a^{(i)}$	Scalar indexed for some purpose
a_{ij}	The element in the ith row and jth column of the matrix $\boldsymbol{A}$
$a_{i,j}$	The element in the ith row and jth column of the matrix $\boldsymbol{A}$
a_i	The ith element of vector $\boldsymbol{a}$ (or taken from a set)
$f_i(a)$	The ith element of the output vector of the function $\boldsymbol{f}$, i.e., equal to y_i for $\boldsymbol{y} = \boldsymbol{f}(a)$
$\mathcal{A}$	Set
$\mathbb{N}$	Natural numbers
$\mathbb{R}$	Real numbers
$\mathbb{R}^+$	Positive real numbers (exclude zero)
$\mathbb{R}^n$	n-dimensional (real valued) vector space
$\mathscr{P}(\mathcal{A})$	Power set of set $\mathcal{A}$
$P(X \geq a)$	Probability that random variable X is greater than or equal to a
$\mathop{\mathbb{E}}_{X \sim p(\cdot)}\{\cdot\}$	Expected value of $\{\cdot\}$ for the random variable X distributed according to $p(\cdot)$
f^{L}	Function f labeled for some purpose
$f \circ g(\cdot)$	The composition of functions f and g, i.e., $f(g(\cdot))$
$\nabla_{\boldsymbol{a}} f(\cdot)$	Gradient of function f with respect to to vector $\boldsymbol{a}$
$\frac{\partial f(\cdot)}{\partial a}$ or $\frac{\partial}{\partial a} f(\cdot)$	Partial derivative of function f with respect to scalar a
$\exp a$	Exponential function of a, i.e., e^a
$\log a$	The natural logarithm of a, i.e., equal to $\ln a$ or $\log_e a$

Abbreviations:

ANN Artificial Neural Network
CNN Convolutional Neural Network
KD Knowledge Distillation
ML Machine Learning
MLP Multi-Layer Perceptron
MSE mean squared error
NN Neural Network
RL Reinforcement Learning
RNN Recurrent Neural Network
SIR Self-Imitation Regularization

1 Introduction

When Geoffrey Hinton, Oriol Vinyals, and Jeff Dean introduced a method they called "Knowledge distillation" in 2014 (Hinton et al., 2015), they shed light upon an intriguing property of artificial neural networks that they referred to as dark knowledge; or at least Geoffrey Hinton did so in a talk named "Dark knowledge" in the Excellent Lecture Series at Toyota Technological Institute at Chicago (TTIC) given on October 2, 2014.

Clearly, this term is more metaphoric than scientific but gives a vivid description of the subject. As quoted on the opening pages, the term dark knowledge is derived from the phenomena of dark matter and dark energy (Abbott et al., 2016) used in astrophysics and yields a description for the material that the universe is mostly made of. To be more precise, dark matter makes up 27% and dark energy 68% of the universe which results in the impressive total portion of 95%. Although the effects and necessity of the existence of dark matter and dark energy were already demonstrated in the 20th century as the reasons why rotating galaxies do not fly apart and the universe is permanently expanding, respectively. However, it is astounding that it is entirely unknown what dark matter and dark energy is made of. Therefore, it seems that over the past decades, most of the research that has been highly successful in physics has been based on areas dealing with the remaining 5% of ordinary matter. The visualization of a dark matter simulation on the title page of this thesis vividly shows a possible segmentation of the universe into complex filaments of dark matter (in black) and rare clusters of ordinary matter (orangeish glowing), and demonstrates how small the ordinary matter share of 5% is.

Returning to computer science, machine learning, and the explanation of Geoffrey Hinton's dark knowledge term in artificial neural networks: In addition to learning the ability to make an accurate prediction based on a prediction task, such as a classification problem – which is a typical setting artificial neural networks are deployed in – artificial neural networks generally acquire much more knowledge as a side product of generalization during the course of learning. For example, in this additional knowledge lies the understanding of similarity and correlation between the different outcomes to be predicted (i.e., the different classes in a classification setting). However, since in most cases of application the final prediction is of interest only, the research community has mainly paid attention to only a small part of knowledge an artificial neural network acquires which is measured by the final prediction performance. Again, note that the given information in the presented quote expound dark knowledge as an analogy of dark matter and dark energy serves more motivational purposes than scientific significance. Especially the percentage of 95% is not related to any portion of knowledge within an artificial neural network in general. However, this dark knowledge phenomenon is clarified in Hinton et al. (2015) explaining the method of knowledge distillation as a concrete procedure of training artificial neural networks, which is going to be explained in more detail in Sec. 2.8. Roughly, knowledge distillation can be described as a technique that allows distilling the (dark) knowledge in a large and deep neural network into another much smaller neural network, which would not be able to acquire that knowledge from the data on its own. These ideas and concepts were taken up to develop a novel regularization method, which is introduced in this thesis.

Regularization encompasses all the techniques supporting a neural network learning the general concept from the data better and to prevent the neural network from overfitting on the (training) data. Overfitting describes the memorization of idiosyncrasies in this data or even of all data examples. If

overfitting cannot be prevented and such generalization cannot be achieved (e.g., by missing or using unsuitable regularization methods), this can have severe consequences in later use even leading to a complete uselessness of the learned neural network, and also it will not perform well on data that were not available in the learning phase. Various regularization methods have been developed that became increasingly often used and popular. Probably the most famous example is the method of dropout (Srivastava et al., 2014), where a random selection of neurons within the neural network is temporally dropped. Now, for a few examples, the neural network gets by with fewer neurons and must continue to learn until some other neurons are picked instead and turned off. Many other well-known regularization methods address the problem of overfitting by directly constraining the weights (parameters within the neural network by whose adaptation the learning takes place). It can be criticized that these methods often approach the problem very indirectly in the form of parameter limitations or changes in the neural network architecture but do not interact directly with the knowledge of the neural network.

Self-Imitation Regularization (SIR), introduced in this thesis, is a regularization method, which uses the (dark) knowledge within the neural network that is currently undergoing learning and makes it explicitly available to the learning principles (i.e., the loss that is aimed to minimize) of the same neural network. The regularization through such a form of self-imitation is convenient and straightforward to implement into an existing learning framework, incurs hardly any additional computational effort, and requires barely any hyperparameter tuning (i.e., particular hyperparameter setting are suggested, which turned out to be robust and able to achieve improvements in many different tasks). Through a large number of experiments conducted within the context of this thesis, SIR demonstrated regularizing abilities (partially with guarantees of success under statistical significance) on multiple data sets and across diverse architectures. Even very deep neural networks with over 100 hidden layers constituted no hurdle compared to other regularizers. In addition to regularizing in terms of minimizing the amount of overfitting, SIR was also successfully used to increase the data efficiency or to stabilize the training (that a neural network learns something meaningful at all; underfitting problem) where other regularization methods were not able to help. Furthermore, in experiments in which many data labels were intentionally incorrectly specified, good results were nevertheless achieved when using SIR. In the experiments, it was often observed that SIR could be beneficially combined with standard regularizers to improve the result much more than if each of the regularization methods had been used separately. In order not to rely solely on empirical results regarding the performance scores of the learned self-imitation regularized neural networks, the learned neural networks are examined more closely and interpreted so as to explain how the regularization is achieved as well as pointing out the effectiveness of different SIR settings.

Regarding the structure of the thesis, the method of Self-Imitation Regularization (SIR) is introduced in Sec. 3 after a detailed introduction into the background topics (Sec. 2) about neural networks and the concept of dark knowledge as well as the methods on which SIR is based. Subsequently, the experiments conducted within the context of this thesis will be explained in Sec. 4 and their results are thoroughly discussed and interpreted in Sec. 5. Here, the regularization ability of SIR is verified (Sec. 5.1) as well as examined (Sec. 5.2), and the behavior of SIR in special problems is further investigated (Sec. 5.3 - 5.7). This is rounded off with summarizing the characteristics of SIR and investigational findings in a conclusion (Sec. 6), whereby the initial working hypotheses are confirmed or rejected.

2 Background

2.1 Machine Learning

Since Neural Networks (NNs) are the subject of interest in this thesis and belong to the field of Machine Learning (ML), the basic concepts and definitions of ML are introduced in this section.

Abstractly, a computer program which takes an input x, somehow manipulates it and then produces an output y which can be considered as a function $y = f(x)$ in a mathematical sense. Therefore, a computer programmer aims to design such a function by programming a computer in order to solve a specific task what means explicitly defining the function f.

As computational resources and performance have increased in recent decades, the demands on computer programs have increased to solve more complex and sophisticated tasks. Furthermore, tasks that are easily manageable by humans such as visual perception or dealing with natural language in both written and spoken form cannot be easily implemented on a computer system since in most cases neither an algorithmic description nor even a mathematical notion exists to specify computer programs (viz. the function f) to cope with these tasks to at least an extent a human being does. Nevertheless, hope is not yet lost, since, for most of these tasks, at least a set of data D is available, which exemplarily shows mappings from the input x to the output y according to the underlying, unknown function f or at least consists of input data. Related to the previous examples, it is relatively easy to provide a set of images along with matching labels describing the object pictured on those or audio snippets of speech with complementary written transcriptions. This is exactly the point where ML begins.

Briefly in conclusion and according to Géron (2017, ch. 1): ML is the science (and art) of programming computers so they can learn from data. This means that the function f is not needed to be specified manually but providing data D to a learning algorithm that manages to learn (or approximate) the desired function f with $\hat{f}$ instead. Another benefit in finding powerful learning algorithms lies in gaining a more general concept of solving a large number of comparable problems by only exchanging the data instead of manually specifying a program (i.e., function f) for each task.

Putting this into perspective, according to today's ML research findings, the principle of learning is split up into different types of problem, and task settings and a variety of learning techniques and algorithms exist instead of having one all-encompassing, universal, and potentially cumbersome learning algorithm. Consequently, in order to apply ML one have to categorize the task roughly according to the following points successively:

$$\text{Problem class (Degree of supervision} \rightarrow \text{Task specification)} \rightarrow \text{Learning algorithm}$$

These points are further explained and defined in the following section.

2.1.1 Categories of Machine Learning

The problem classes of ML can be distinguished into three classes: supervised learning, unsupervised learning and reinforcement learning. This distinction is based on the degree of supervision. Supervision

in this context is essentially describing the further knowledge and information the raw input data is accompanied with. Imagine, when we have the correct output of the unknown function f instead of only having a measure that assesses the quality of the outputs of a (partly) learned function $\hat{f}$, or, even worse, having no information except for the input data, this represents a higher degree of supervision. In addition to the degree of supervision, the different problem classes yield interpretations for the different goals of ML: Unsupervised learning describes the data, supervised learning makes predictions whereas reinforcement learning can be seen as going beyond and making decisions based on the data. The terminology of making *predictions* based on the input as used in supervised learning can also be rephrased as directly mapping inputs to *outputs* for what reason supervised learning fits most with the concept of learning a function $y = \hat{f}(x)$.

After deciding on the class based on the degree of supervision, we have to specify the task further. Any task specifications within these are further explained in the following but note that the focus lies on supervised learning since the method introduced in this thesis is mostly investigated in terms of this problem class.

Supervised learning

As the name suggests, supervised learning has a high degree of supervision. In particular, our data includes true outputs of the unknown function f. First of all, it is more common to refer to the (true) outputs of the unknown function f taken from a data set $\mathcal{D}$ as (true) data targets (or also as labels). Formally, our data set consisting of inputs $x \in \mathcal{X}$ and targets $y \in \mathcal{Y}$ is defined as

$$\mathcal{D} = \{(x^{(d)}, y^{(d)}) \in \mathcal{X} \times \mathcal{Y} \mid d = 1, 2, \ldots, D\} \quad \text{with} \quad D > 0.$$

Note that $\mathcal{X}$ and $\mathcal{Y}$ can indeed be infinite sets, but the amount of data we can collect and process (data set $\mathcal{D}$) is finite. Therefore, the number of data samples $(x^{(d)}, y^{(d)}) \in \mathcal{D}$ (also called data examples or data instances) is denoted with D.

Depending on the nature of the targets $y \in \mathcal{Y}$ different task/problem specifications do exist in supervised learning:

- **Classification:**
  For *classification* problems, the target set $\mathcal{Y}$ is a discrete set ($|\mathcal{Y}| \neq \inf$) and the possible n target values $\{1, 2, \ldots, n\} = \mathcal{Y} \subset \mathbb{N}$ are the indices referring to the elements of a set of classes $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$. A classification task can be further discriminated into a binary ($n = 2$), or a multi-class or n-class ($n > 2$) classification which are also called logistic regression and multinomial logistic regression respectively. In practice, it is often the case that the indices are further represented via a one-hot-encoded vector. A one-hot-encoded vector is a special case of a binary-encoded vector, where each component can either be one or zero with the further restriction of only one component is allowed to be one ("hot") per encoding vector. Formally, this results in a new target set of $\mathcal{Y} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\} \subset \{0, 1\}^n$ denoting the set of the unit vectors of the standard basis in an n-dimensional Euclidean space. One hot encoding is being done since using the raw index values as target representation might entail the risk of some learning algorithms falsely inferring that classes with a smaller quantitative difference between their index values might be semantically more related than classes with a higher difference between their index values.
  Example: Classify images according to the animal depicted on them. Having the images (represented by raw RGB pixel values) as input data samples $x^{(i)}$ of cats and dogs, subsequently, with the classes

$\mathcal{C} = \{cat, dog\}$, *and trying to learn a function discriminating between cat and dog images is a binary classification problem.*

- **Multi-label classification**:
  Multi-label classification is highly similar to standard classification problems and, therefore, often misunderstood. We also have a discrete target set $\mathcal{Y}$, but it does not consist of one-element labels any more. Instead, multiple labels can be assigned to one input instance x at once. Thus, a discrete set of different labels (similarly to the set of different classes in standard classification problems) is defined as $\mathcal{L} = \{l_1, l_2, \ldots, l_n\}$. By allowing all possible combinations of labels as targets for this problem, the target set can be expressed as the power set of the label set $\mathcal{Y} = \mathcal{P}(\mathcal{L})$ inducing 2^n possible combinations of labels (or analogously doing this with the indexes of the labels in $\mathcal{L}$). The targets are often represented as a binary-encoded vector in practice by each component is indicating the presence and absence of a specific label via a component of one and zero, respectively. It is important to not confuse the terms labels and classes in the context of multi-label and multi-class classification since multi-class classification indicates the standard classification problem from above with $n > 2$ classes.
  Example: Consider the problem of classifying images from above but with multiple animals depicted per image. While a trained multi-class model is only capable of predicting one of the shown animals, a multi-label model can accomplish this task better by predicting labels for all shown animals.

- **Preference learning or label ranking**:
  Preference learning or label ranking is similar to multi-label classification, but instead of only predicting the presence of a set of labels for each input instance, a ranking of these labels is predicted (for example through numeric scores $\boldsymbol{y}$). This ranking represents a (partial) ordering of the labels from which the preference of arbitrary two labels ($L_i \succeq L_j \Leftrightarrow y_i \geq y_j$) can be derived. Multi-label classification can be seen as a special case of preference learning which is why a ranking can be used to predict a subset of labels, e.g., by introducing an artificial calibration label that separates the relevant from the irrelevant labels within each label ranking (Fürnkranz et al., 2008).
  Example: Recommendation systems based on user data can be understood as a preference learning task, where not only a subset of the available labels should be displayed but also in an order that the labels displayed first fitting the user's preferences best.

- **Regression**:
  In contrast to the problem specifications mentioned above, the target set $\mathcal{Y}$ for a *regression* problem is continuous ($|\mathcal{Y}| \to \infty$) and can be an arbitrary n-dimensional space $\mathcal{Y} \subseteq \mathbb{R}^n$.
  Example: Given records of stock price developments as training data can be used to train a model predicting the expected stock price for the next hour.

Note, that more supervised learning tasks are present, but that would go beyond the scope of this thesis.

Unsupervised learning

In this regime, we have a total lack of supervision. More detailed, contrary to supervised learning, we have no targets besides the input data $x \in \mathcal{X}$ which results in a thinner, unlabeled data set

$$\mathcal{D} = \{(x^{(d)}) \in \mathcal{X} \mid d = 1, 2, \ldots, D\} \quad \text{with} \quad D > 0.$$

Typical tasks in unsupervised learning are, for example, *clustering* where data samples are divided into clusters based on discovered similarities of data features and properties, or *density estimation* where the underlying distribution (or function) the input data was drawn from (or what produced the data) is learned by estimating the density function of this distribution.

Reinforcement learning

In the *reinforcement learning* arrangement, an agent interacts with an environment over (discrete) time steps by taking actions based on perceived observations and rewards, which is the only supervision in this setting, to reach a specific goal. See Sutton and Barto (2018) for more details. Formally, this can be modeled as a Markov Decision Process (MDP) where the actions for a given state the agent selects are sampled from a policy (distribution or function). For an optimal behavior, the agent is aimed to maximize the expected (discounted) return, which is the (geometrically weighted) sum of the rewards collected within an episode.

One can consider learning an approximation of the value function, e.g., the Q-function, estimating this return to plan the actions. The *Q-function* provides an estimate about the expected return for the rest of the episode if the current policy is followed (given the current state and an action that could be executed next). This, coupled with *Bellman's principle of optimality* (Bellman et al., 1957), allows composing an optimal policy from the Q-function by greedily selecting the action for the given state for which the Q-function estimate is maximal. *Q-learning* (Watkins and Dayan, 1992) is a technique to learn an approximator of the Q-function where the estimate is adjusted according to the *temporal difference error* (Sutton and Barto, 2018, ch. 6) measuring the error between the Q-function estimate for the current state and the next state by incorporating the reward perceived in the current state.

Semi-supervised learning

Apart from the three problem classes stated previously, one might consider additional problem classes. However, the central problem classes of ML are remaining the three ones since other problem classes can primarily be expressed either as combinations or simplifications of these major three problem classes without the loss of generality. For example, *semi-supervised learning* can be formulated as a combination of supervised and unsupervised learning or *active learning* can be obtained by a simplification of Reinforcement Learning (RL). However, the principle behind semi-supervised learning is roughly outlined since semi-supervised learning plays an increasingly important role in ML. Furthermore, the possible application of the introduced method, Self-Imitation Regularization (SIR), will be discussed in this thesis.

As mentioned above, semi-supervised learning can be seen as a combination of both supervised and unsupervised learning. Accordingly, the data set consists of both labeled and unlabeled data

$$\mathcal{D} = \mathcal{D}_{labeled} \cup \mathcal{D}_{unlabeled}$$
$$\text{with } \mathcal{D}_{labeled} = \{(x^{(d)}, y^{(d)}) \in \mathcal{X} \times \mathcal{Y} \mid d = 1, 2, \ldots, D\},\ \mathcal{D}_{unlabeled} = \{(x^{(d)}) \in \mathcal{X} \mid d = 1, 2, \ldots, D_{unlabeled}\},$$
$$\text{and } D_{labeled} + D_{unlabeled} > 0.$$

From another point of view and more obvious when considering the formal definition, semi-supervised generalizes supervised and unsupervised learning. Therefore, $\mathcal{D}_{labeled} \neq \emptyset$ and $\mathcal{D}_{unlabeled} = \emptyset$ or alternatively $D_{labeled} > 0$ and $D_{unlabeled} = 0$ results in supervised learning whereas $\mathcal{D}_{unlabeled} \neq \emptyset$ and $\mathcal{D}_{labeled} = \emptyset$ or alternatively $D_{unlabeled} > 0$ and $D_{labeled} = 0$ yields unsupervised learning as special cases of semi-supervised learning.

Before finishing this section about semi-supervised learning, it is worthwhile mentioning that the techniques of semi-supervised learning are very useful and would improve the learning on any problem class with a higher level of supervision than unsupervised learning, e.g., supervised learning or RL, if additional data without supervision is around. This is a widespread case in practice since labeling the data is often associated with high expenses and costs, e.g., when human experts are needed to label the data, and some unlabeled data is left. The reason why we can profit from unlabeled data can be explained by the fact that unsupervised learning has the interpretation of learning to describe the data as mentioned earlier. Intuitively, by having a more accurate description of the data, a prediction task might be learned more easily and better due to an improved understanding of the input data and its patterns. Considering this in two steps of understanding the data first and solving the prediction task based on this improved understanding later, the first step can be seen as a way of representing the data in order to make it more informative for a prediction base, which is commonly referred to as (automatic) feature extraction. As a result, algorithms that automatically generate features from the input might benefit from processing more (unlabeled) input data while learning. E.g., in artificial neural networks using a multi-layered network architecture, features are generated across the first layers and down-stream layers are making use of it to compute the prediction.

2.1.2 Algorithms in Machine Learning

There are many learning algorithms and types of models that are designed for specific problem classes and tasks. A rough distinction can be made between parametric and non-parametric learning algorithms (or models).

Parametric models contain parameters and learn through the learning algorithm adjusting these parameters. The vector containing all parameters of the model is denoted as $\boldsymbol{\theta}$ and the model as $\hat{f}(x) = f(x; \boldsymbol{\theta})$. A (deep) neural network is a typical example of parametric models with millions of parameters in some cases.

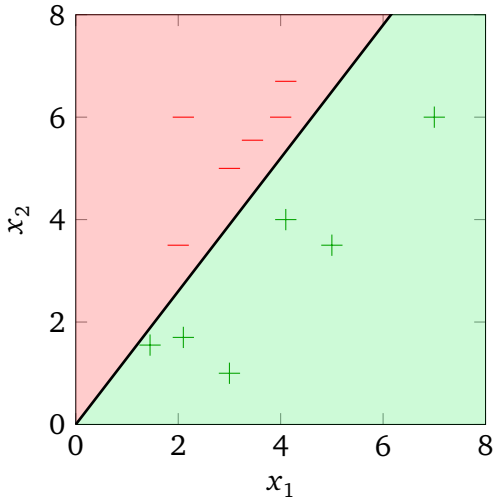
Non-parametric learning algorithms do not adjust the parameters of a model in order to learn or frequently do not learn a model at all. A well-known example is the *k nearest neighbor* algorithm, which stores the training data and predicts the target appearing most within the k nearest neighbors (training data instances) for a query at prediction time. To achieve good results, a proper distance metric for the input data space is needed, which considers semantic similarities of the data. That is hard to obtain in settings with high-dimensional input data with a proportionally low degree of useful information as it is the case for image data (Karpathy and Li, 2015, ch. 1: Image Classification).

Once a suitable learning algorithm has been chosen, the hyperparameters must be set properly.

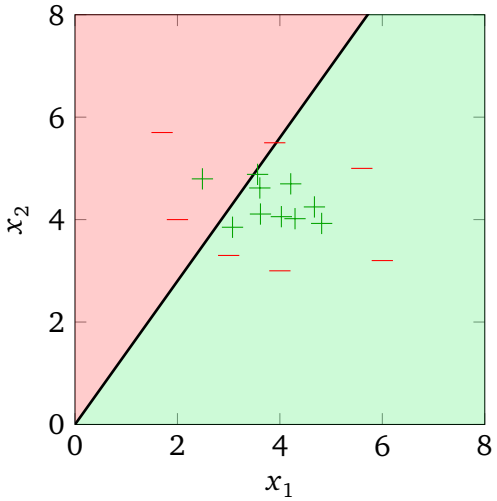
Hyperparamters:

Hyperparameters are parameters in a learning algorithm which are not being learned but set beforehand by the expert/programmer. For example, k of the k nearest neighbor algorithm is a hyperparameter or in the regime of (artificial) neural networks a high number of hyperparameters is introduced. For example, the number of neurons the neural network is meant to contain, the number of layers in which these are organized, or the learning rate.

Tuning these hyperparameters so that the learning algorithm performs well is tricky since this suffers from the *curse of dimensionality* (Bellman et al., 1957). Moreover, high-dimensional spaces of hyperparameters do commonly have low effective dimensionality, which means that the outcome of a hyperparameter search is more sensitive to changes in some particular dimensions than in other ones (Caflisch et al., 1997) since some hyperparameters affect the outcome of the learning algorithm more



(a) Linearly separable classification problem



(b) Non-linearly separable classification problem

Figure 2.1.: Example of binary classification problems (data samples (x_1, x_2) of the positive class are denoted with + and of the negative class are denoted with −) where the data is linearly separable in **(a)** and non-linearly separable in **(b)**. The decision boundary (separation) of the linear model is depicted with a black line where data samples in the green sub-space will be predicted as positive and data samples in the red sub-space will be predicted as negative. The linear model can successfully separate **(a)** while this is not possible for **(b)**.

notable than others. Bergstra and Bengio (2012) showed that random search is therefore more effective than grid search in practice since with the latter approach much computation time is wasted in examining hyperparameters which might be unimportant while holding all the other hyperparameters fixed.

2.1.3 Expressive power and separability

By choosing a concrete model in ML, we make non-negligible assumptions about the problem-solving approach and the hardness and idiosyncrasies of the given task, which is aimed to be learned.

The expressive power refers to the capability of the model to express a solution for the given problem. If the expressive power of a model is too low, the task cannot be solved. For example, if we want to fit a linear model in a regression problem (i.e., fitting the data with a line) where the underlying function of the training data is non-linear, the result will be that our model insufficiently expresses or fits the data.

For classification problems this is pretty much the same in that the chosen model has to be rich enough to be capable of separating the space of input samples and assigning these sub-spaces to particular classes. Mainly, a rough distinction is made between linearly and non-linearly separating models and linearly and non-linearly separable problems (Minsky and Papert, 1988, ch. 12). See linear and non-linear separability in an example of a binary classification problem visualized in Fig. 2.1.

Further statements can also be made if a model has to solve more complex sequences of tasks. The classes from computational complexity theory are used for this purpose. For example, a model is Turing-complete if it has the expressive power of an arbitrary computer program .

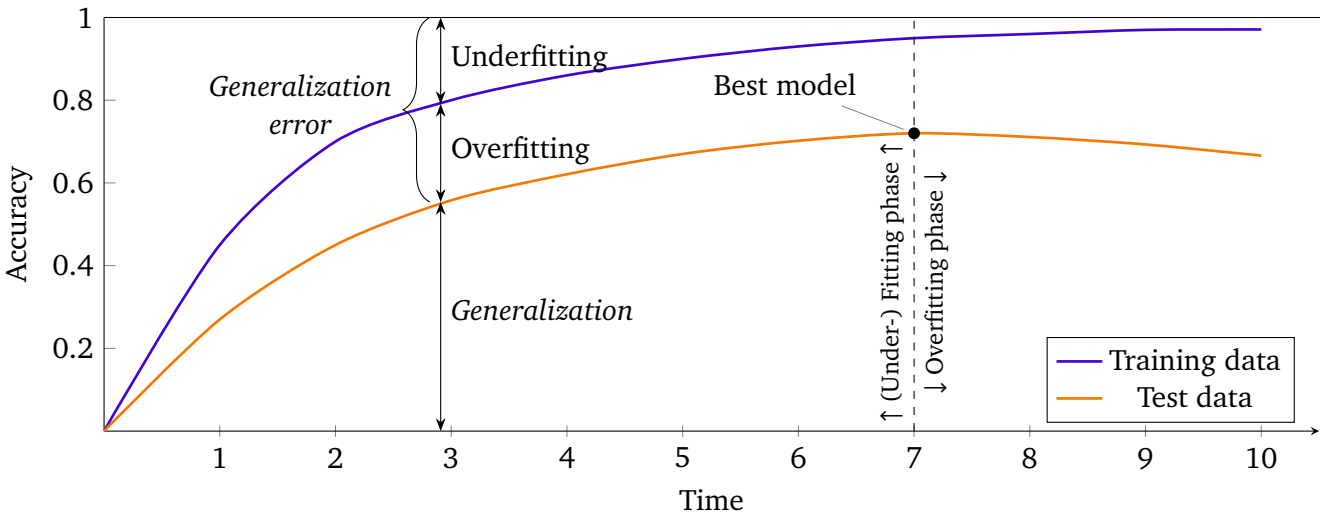


Figure 2.2.: Generalization behavior visualized in the learning curves regarding the accuracy metric on the training data (purple curve) and test data (orange curve) over the course of training. Additionally, the gaps between curves and axes are labeled with the common interpretation terminology and the (under-) fitting and overfitting phases are depicted, which are oriented to the iteration where the best model was achieved.

2.1.4 Generalization

The original goal of ML was to learn a function or task automatically without providing further information about the problem-solving approach. However, this goal is aimed at being achieved implicitly by letting an algorithm learn from (training) data involving the solution (i.e., the desired model output) for an input but no further solving approaches. This inevitably leads to a small but not negligible shift in the formulation of the goal away from the original one. Namely, learning concepts from the (training) data in order to perform well on this very same data whereby we assume that this good performance will remain on unseen data (meaning data which was not involved during training) and consequently solving the original goal implicitly. This capability of transferring the knowledge to unseen data is referred to as *generalization*. To assess the degree of generalization another data set called the test data set, which is independent of the training data, can be used to measure the performance of the model on this data. A typical generalization behavior is plotted in Fig. 2.2 with the learning curves regarding the accuracy metric on the training as well as on the test data over the course of training (e.g., ten training epochs, which means that the training algorithm has processed the training set ten times). The accuracy metric measures the relative amount (in relation to the number of samples in the data set) of correct classifications on the given data set.

The generalization (of the model), depends on (a) the model, (b) the learning algorithm and (c) the training data. Concerning the model (a), the capacity and expressive power of the model is crucial to capture the concept and patterns of the training data in order to achieve the task. The capacity of a model is usually expressed in the number of parameters and whenever the capacity is too low or the expressive power is improper (e.g., trying to separate the data by a linear model but the data is non-linearly separable only) in order to capture the concept the problem of *underfitting* occurs. Apart from that, the problem of *underfitting* can also occur when iterative learning algorithms, such as those for learning neural networks, are stopped prematurely whereby the model has not run sufficiently many iterations to fit on the training data properly. Underfitting can be seen in the gap between the accuracy

on the training data and the optimal accuracy value (commonly 1.0). Ensuring a proper expressive power and increasing the capacity of the model addresses the problem of underfitting in most cases but depending on the choice of the learning algorithm (b) this is prone to *overfitting*. Overfitting can be seen as the gap between the performance on the training data and the test data. This is very dependent on the chosen learning algorithm as, for example, if the model fits the training data (which shows that the capacity is high enough) the model can entirely suffer from overfitting and barely achieves generalization if the learning algorithm encourages rote learning leading to the pure memorization of the training data samples, which is low risk in terms of a low training error but obviously does not serve to generalize the model. Nevertheless, it is common practice especially in *deep learning* to choose high capacity models while addressing the problem of overfitting with the learning algorithm, i.e., with appropriate regularization techniques (Goodfellow et al., 2016, ch. 7, p. 225) as described in Sec. 2.7 (standard regularization methods) or the novel regularization method introduced in this thesis (see Sec. 3). The suitability of the training data (c) should not be neglected since the success of generalization or even sufficient fitting on the training data hinges with this. Otherwise, the model might tend to overfit to idiosyncrasies of the training data. This is dependent on the amount of data available for training, on the one hand, since a small training set might not be sufficient to recognize general patterns and structures, as well as dependent on the quality of the training data, on the other hand, especially in the case of supervised learning regarding the correctness of the target labels, which were set by humans or even human experts beforehand. Additionally, it is essential to ensure that the distribution and characteristics of the training data match those of the test data or, more generally, match those of the data the learned model is planned to be used on in the future. Furthermore, the samples of the training data have to be drawn independently from the same distribution (in which case the training data and training samples are said to be independent and identically distributed, which is often abbreviated to i.i.d.) (Bishop, 2006, ch. 1.2.4, p. 26).

As one can see in the learning curves of Fig. 2.2, the test accuracy (representing the generalization) is not necessarily asymptotic during learning and can decrease again in a later phase of training. This phase can be denoted as the *overfitting phase* whereas the phase before the maximum in the test curve is reached can be called *(under-) fitting phase*. This is because the model needs some time to learn the general concept and patterns of the data at the beginning of training (fitting phase) while starting to gradually overfit to special samples and idiosyncrasies of the training data later on (overfitting phase). This is the reason why it is advisable to take a snapshot of the best model according to the accuracy (or a similar metric) on some test data over the course of training. After training, this snapshotted model is the *best or most generalized model* achieved during training and is represented by the maximum of the test set learning curve. Within the underfitting and overfitting phases the problem of underfitting and overfitting respectively, is the predominant factor hindering the model from generalizing.

2.2 Artificial Neural Networks

So-called Artificial Neural Networks (ANNs), or also Neural Networks (NNs) for short, can be modestly described as universal, non-linear, and parametric function approximators (Hornik et al., 1989) in mathematical terms. Since this does not provide an easily understandable introduction to this topic and the name itself is inspired from an entirely different scientific field anyway, we are going to start with a more intuitive perspective: namely from the field of biology or preferably, neuroscience.

Neuroscientific inspiration:

The (human) brain is a huge network consisting of over 120 billion neurons (Herculano-Houzel, 2009) which receive electrical and chemical signals from adjoining neurons and combine those signals by temporal and spatial summation. If a unique location, the axon hillock, a swollen part of the cell body of a neuron, is triggered by a charge exceeding a particular threshold a signal is fired on its way to a large number of subsequent neurons (Kandel et al., 2013, ch. 2) known as the *all-or-none* principle. By doing so, all of these neurons are solving specific tasks by propagating information encoded in the electrochemical signals step by step where each neuron solves a particular and more simple subtask. Due to the hierarchical arrangement, the brain (a network of neurons) is able to accomplish the whole and more complex task.

The connections between neurons where the transmission of the signals happens are called synapses. The fact that these synapses can transmit the signals to different degrees of effectiveness and, moreover, this modulation behavior of synaptic transmission can change over time, leads to the underlying mechanism of storing (implicit long-term) memory: a from of learning. This implicit memory storage covers the unconscious and automatic memorization of perceptual and motor skills or habits through habituation, sensitization, and conditioning, for example. Habituation involves activity-dependent presynaptic depression of synaptic transmission whereas sensitization and conditioning result from presynaptic facilitation or long-term potentiation of synaptic transmission (Kandel et al., 2013, ch. 66; Chapman et al., 1990). In a nutshell, this type of learning involves changes in the strength of the synaptic signal transmissions to trigger reactions and behavior that might be advantageous according to the experienced and recurring stimuli.

Note that another type of learning in the human brain exists: the long-term explicit memory storage. This happens consciously like remembering places, objects or people and begins in the hippocampus and medial temporal lobe of the neocortex, which is different from the implicit memorization. Interestingly, many skills are initially stored as explicit memory, however, through repetition become ingrained and stored as implicit memory. For example, gradually automating particular finger movements while practicing a piano piece over time (Kandel et al., 2013, ch. 67).

Artificial Neural Networks in computer science:

Now, with this brief biological illustration in mind, we can acquire a mathematical model to approximate arbitrary functions by having small units also called neurons doing simple calculations based on numerical input signals with each solving a task partially. They are arranged in a hierarchically structured network by each passing a numerical output signal to subsequent neurons. To achieve an analogy for the modulation behavior of the synaptic transmission effectiveness, real-valued weights are used to scale each input signal. Obviously, by adjusting those weights adequately, the artificial neural network is able to accomplish specific tasks. In order to fit the weights accordingly, automated and data-driven learning algorithms are used in most cases. This process is referred to as learning or, alternatively, as training, optimizing, or fitting the artificial neural network. All of this is explained in more detail in the following sections. However, this is basically the composition of artificial neural networks and holds for many different variants and variations of artificial neural networks developed in the past years.

At this point, it has to be emphasized that – contrary to popular belief often found in the media or popular science – with the use of artificial neural networks we claim neither to simulate the biological brain which is impossible as its functioning is far from being fully discovered yet, nor to offer an adequate or correct abstract model allowing us to draw any conclusions about the brain. By doing so, we would neglect important and strong assumptions since many choices of design and construction are made for mathematical and computational reasons. So the brain should just be seen as an inspiration. In fact,

this is a highly valuable inspiration from which many brilliant ideas and methods have been derived in the past and which drives the research of neural networks further, however it is meant to be no more than an inspiration. Subsequently, and as mentioned above, many techniques were often deduced from mathematics and computer science, making these both essential sources of inspiration and indispensable "toolboxes" in order to get all of this working or even tractable.

The following will examine the concept of artificial neural networks in more detail by starting with the smallest components and ending with advanced architectures.

Early history, in a nutshell:

It all began back to the 1940s with the first theories of biological learning as well as mathematical representations and formal descriptions of neural signal processing (McCulloch and Pitts, 1943; Hebb, 1949; Widrow and Hoff, 1960) moreover with the first breakthrough by Frank Rosenblatt who designed an algorithm called *perceptron* (Rosenblatt, 1958, 1962) and implemented this on a machine to learn image recognition tasks. 20x20 photocells of a camera were connected randomly to multiple neurons building the single-layer *perceptron* which was aimed to learn a binary classification of the perceived visual content. As common in ML, in order to learn the task, the *perceptron* was shown several visual inputs. Whenever the *perceptron* misclassified such an input, the weights were adjusted based on *Hebb's rule* (Hebb, 1949). After the summation of the weighted inputs, the *Heaviside step function*, which maps negative arguments to zero and non-negative arguments to one (see Fig. 2.8b), was used as activation function in order to produce the final output which was obviously inspired by the *all-or-none principle* of nervous activity.

Overall, this outstanding work caused a great interest in the field of ML but was later superseded by Marvin Minsky and Seymour Papert in 1969 who showed that a *perceptron* is not able to represent the logical XOR function (Minsky and Papert, 1969) due to the capability of learning linearly separable patterns only. Later on, it was shown that the logical XOR function can still be learned with a different non-linear choice of activation function (Melnychuk et al., 2017) or the use of a so-called Multi-Layer Perceptron (MLP) (with a single hidden layer and non-linear activation functions) which will be explained in Sec. 2.2.3. However, Minsky and Papert (1969) had such a massive impact on neural network research that this stagnated until the 1980s and revived with the invention of the backpropagation algorithm (Rumelhart et al., 1986), explained in Sec. 2.2.4.

2.2.1 The neuron

Nevertheless, the formulation of a neuron as explained in the *perceptron* algorithm is still present and is still used in a slightly generalized form in today's definition of artificial neural networks.

In Fig. 2.3, the composition of an artificial neuron is schematically visualized. Each of the I inputs (also referred to as input activations) the neuron perceives $(x_1, x_2, \ldots, x_I)$ are element-wisely multiplied by the weights $(w_1, w_2, \ldots, w_I)$ in the neuron first. Each weight is exactly assigned to one input at a time which results in the requirement that we always must have as many weights in a neuron as incoming input activations. Inputs, as well as weights, are real numbers with no further restrictions in general.

The weighted input activations are summed up including an additional parameter: the bias b. The bias is independent of the inputs or can also be seen as an additional weight $w_0 = b$, which is dedicated to a constant input activation of 1. Obviously, a neuron is a parametric model and the vector of parameters $\boldsymbol{\theta}$ incorporates all the weights and the bias of the neuron. Formally, $\boldsymbol{\theta} = (w_1, w_2, \ldots, w_I, b)^\top$. Throughout the training, these parameters are optimized to improve the neuron's capability to capture the concept of the input and produce the desired output.

To obtain a better understanding of the summation z (called pre-activation) of the weighted inputs it should be mentioned that this weighted sum can also be seen as linear combination of weights and inputs or as the dot product between the vector of the weights $\boldsymbol{w} = (w_1, w_2, \ldots, w_I)^\top$ and the vector of the inputs $\boldsymbol{x} = (x_1, x_2, \ldots, x_I)^\top$. Its calculation can be described in matrix-from as follows

$$z = \sum_{i=1}^{I} w_i \cdot x_i + b = \boldsymbol{w}^\top \boldsymbol{x} + b \tag{2.1}$$

Up to this point, a neuron is clearly nothing more than a simple *linear model* used in different areas of ML. For example, when solving linear regression each explanatory variable has its own slope coefficient (Freedman, 2009, p. 26) which is equivalent to scaling each input activation x_i by the corresponding weight w_i in a neuron. As the name *linear model* suggests, this type of model is limited to solve linearly separable problems only. Therefore, we do not use z as the final output of the neuron but apply an activation function (also referred to as transfer function) φ first.

Historically, the *Heaviside step function* as the activation function in the perceptron mimics the *all-or-none* principle of a biological neuron and the fact that this provides binary instead of real values as output is preferable for binary classification problems. However, the activation function is furthermore of crucial importance in a mathematical context since an appropriate choice serves the capability of a neuron to no longer being limited to separate linearly. Thus, the activation function has to be non-linear, e.g., of sigmoidal shape, a step function, or piecewise linear. The activation function is single-variable and scalar-valued and is commonly of (strictly) monotonically increasing, differentiable, and bounded nature but many well-functioning and widely used activation functions do not meet all of these criteria. Some common activation functions are presented in Sec. 2.4. Note that choosing the identity function $\varphi(z) = z$ as activation function yields a standard linear model as a special case of an artificial neuron. Conversely, an artificial neuron is a generalization of a linear model.

Finally, rather than having multiple input activations, each neuron provides exactly one output activation o. This is merely the result of the activation function $o = \varphi(z) = \varphi(\boldsymbol{w}^\top \boldsymbol{x} + b)$. A more general formulation denoting the output of a parametric model is $o = f(\boldsymbol{x}; \boldsymbol{\theta})$.

2.2.2 Layer of neurons

Up to now, using a single neuron is only capable of producing one output by definition, which at first glance seems sufficient for classification problems or regression problems in ML since there is only one class $c \in \mathcal{C}$ or one value $y \in \mathcal{Y}$, respectively, per input sample $\boldsymbol{x}$ desired to be predicted. However, a single output value is not sufficient for regression problems with multiple outputs $\boldsymbol{y} \in \mathcal{Y} \subseteq \mathbb{R}^n$ or when the classes in a multi-class classification problem are *one-hot encoded*. One-hot encoding is always suggested since predicting the class index with a single output value would implicitly lead to learning the misconception about classes whose index values are near to each other that they would be more (semantically) similar to each other than to other ones with more distant index values (as mentioned in Sec. 2.1.1).

For that reason, using several neurons is the most straightforward approach to obtain a model which incorporates the prediction of multiple outputs or, in other words, a vector-valued output $\boldsymbol{o} = (o_1, o_2, \ldots, o_n)^\top$ instead of a scalar one o. Each neuron receives exactly the same input activations $\boldsymbol{x}$ but due to the different weight, as well as bias, values between the neurons, each neuron is able to produce an individual output (activation) o_j. Such a setting is called a layer (of neurons) or also

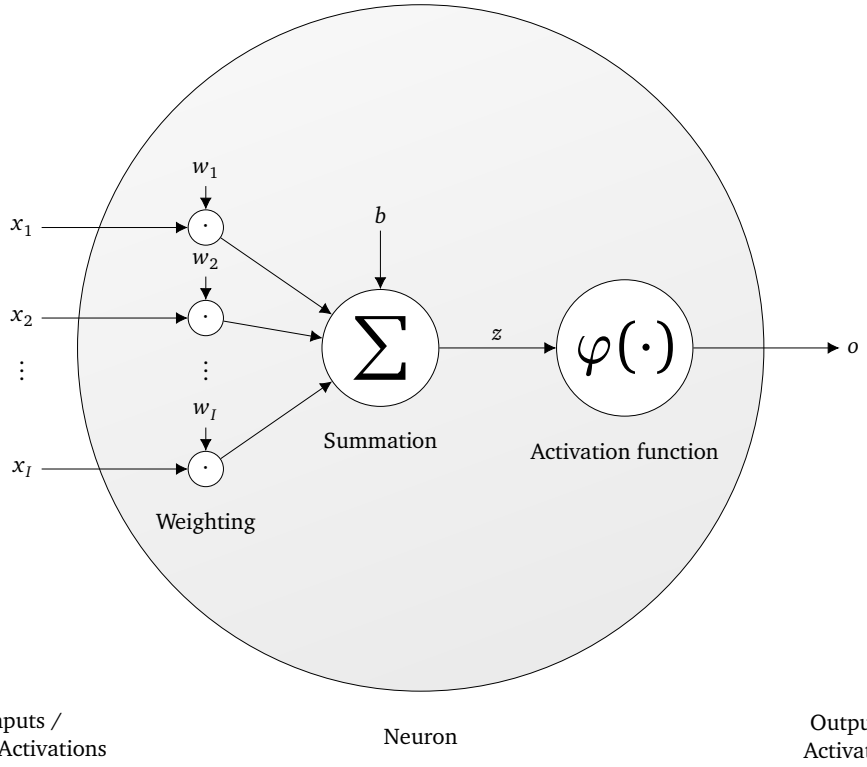


Figure 2.3.: Schematic structure and computation flow of an artificial neuron. I denotes the number of input activations (e.g., the dimension of a data sample used as input).

single layer perceptron. The notation for a single neuron of the preceding section (Sec. 2.2.1) is extended for a layer of J neurons in the following. The weights of the jth neuron within the layer are denoted with $w_{j1}, w_{j2}, \ldots, w_{jI}$ and are gathered in the weight vector $\boldsymbol{w}_j = (w_{j1}, w_{j2}, \ldots, w_{jI})^\top$ where each neuron perceives I input activations. Furthermore, the bias, the pre-activation, and the output (activation) are indexed by j in the jth neuron being b_j, z_j, and o_j, respectively.

Instead of computing the outputs of the neurons in the layer separately, the dot product formulation from Eq. 2.1 computing the pre-activation z_j for a single neuron is extended to a matrix-vector product formulation in order to obtain the pre-activations $\boldsymbol{z} = (z_1, z_2, \ldots, z_J)^\top$ of all J neurons through one operation

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \quad \text{where} \quad \boldsymbol{W} = \begin{pmatrix} w_{11} & w_{12} & \ldots & w_{1I} \\ w_{21} & w_{22} & \ldots & w_{2I} \\ \vdots & \vdots & \ddots & \vdots \\ w_{J1} & w_{J2} & \ldots & w_{JI} \end{pmatrix} = (w_{ji})_{j=1,\ldots,J;i=1,\ldots,I} \,.$$

As a consequence the vector of parameters for the layer has to be extended and can be achieved by the vectorization of the block matrix containing the weight matrix $\boldsymbol{W}$ and the bias vector $\boldsymbol{b}$ of the layer $\boldsymbol{\theta} = \mathrm{vec}((\boldsymbol{W} \quad \boldsymbol{b}))$ in order to capture all learnable parameters present in the layer.

Note that it is common to use the same activation function φ for all neurons in one layer. For that reason, the output activations the layer of neuron predicts can be stated as follows

$$\boldsymbol{o} = \boldsymbol{\varphi}(\boldsymbol{z}) = \boldsymbol{\varphi}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$$

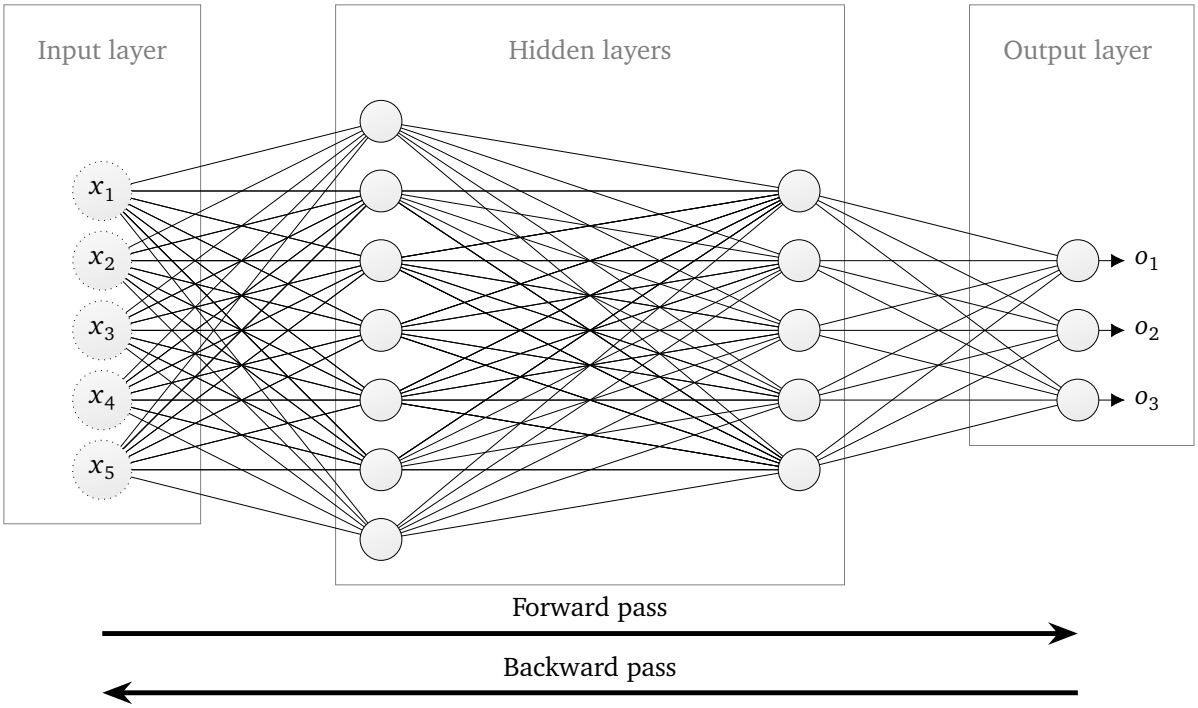


Figure 2.4.: Schematic structure of a Multi-Layer Perceptron (MLP) consisting of four layers ($L = 4$) where two hidden layers exist. The inputs are five-dimensional, and the output is three-dimensional (e.g., represents three classes). The directions of the computation flow for both forward (computing the output of the neural network) and backward pass, e.g., learning the parameters with the backpropagation algorithm (see Sec. 2.2.4), are depicted.

where $\boldsymbol{\varphi}$ simply denotes the element-wise application of the activation function φ whereby $o_j = \varphi(z_j)$ applies.

2.2.3 Multi-Layer Perceptron (MLP); or, stacking layers of neurons

By combining several layers of neurons sequentially, an ANN is obtained. This means that the output activations of a layer are routed to the subsequent layer as input activations. Such stacking of layers is referred to as Multi-Layer Perceptron (MLP) since the terminology ANN is an umbrella term comprising several different network architectures where layers are not required to be combined sequentially only, for example. MLPs are commonly schematically visualized as depicted in Fig. 2.4 where one can see an MLP with four layers ($L = 4$).

The first layer, on the left of Fig. 2.4, called the *input layer*, is not a real layer of neurons since the neurons in the input layer do not contain any learnable parameters (weights or biases) but represent the perception of the input data vector $\boldsymbol{x}$ where the jth neuron of the input layer transmits the jth component x_j of the input data vector $\boldsymbol{x}$ to all of the down-stream neurons in the following layer. For that reason, the neurons in the input layer are visualized differently from the other neurons of the network through a dotted boundary in Fig. 2.4. The last layer (or Lth layer) on the right side of Fig. 2.4 is called the *output layer* since the output activations of this layer are treated as the output vector of the whole network or model $\boldsymbol{o} = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$. The remaining intermediate layers, which are indexed with $l = 2, \ldots, L-1$, are referred to as the *hidden layers* since they do not have a direct connection to the "outside world" of the model unlike the input and output layer dealing either with the inputs or outputs. The (output) activations of the hidden layers are hidden within the model.

From a functional point of view, the model (i.e., an MLP consisting of L layers) is described by

$$\boldsymbol{o} = \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}) = (\boldsymbol{f}_L \circ \ldots \circ \boldsymbol{f}_2 \circ \boldsymbol{f}_1)(\boldsymbol{x};\boldsymbol{\theta})$$

where f_l denotes the function producing the output activations of the lth layer and $\circ$ the composition operator of two functions $(f \circ g)(x) = f(g(x))$. Furthermore, the notation of the constituents of a single layer of neurons as stated in Sec. 2.2.2 is extended to the multi-layer setting by indexing these with the index of the layer they are contained in. Hence, the weight matrix, bias vector, parameter vector, pre-activation vector, and output activation vector of the lth layer where $l = 2,\ldots,L$ is denoted with $\boldsymbol{W}_l$, $\boldsymbol{b}_l$, $\boldsymbol{\theta}_l$ $\boldsymbol{z}_l$, and $\boldsymbol{o}_l$, respectively, which must not be confused with the lth component of some vector therefore it is printed in boldface. For the input layer ($l = 1$) any notation for the weight matrix, bias vector or pre-activation vector is superfluous because of the lack of parameters within this layer, but the output activation of this layer can be expressed with $\boldsymbol{o}_1 = \boldsymbol{x}$. Another notational equivalence exists in the output layer with $\boldsymbol{o} = \boldsymbol{o}_L$ denoting the output of the entire model. Finally, the extension of notation allows for a model formulation in matrix notation

$$\boldsymbol{o} = \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{o}_L \quad \text{with} \quad \boldsymbol{o}_l = \varphi(\boldsymbol{W}_l\boldsymbol{o}_{l-1} + \boldsymbol{b}_l) \quad \text{where} \quad l = 2,\ldots,L \quad \text{and} \quad \boldsymbol{o}_1 = \boldsymbol{x}.$$

Again, the parameter vector $\boldsymbol{\theta}$ has been extended to include all learnable weights and biases of the whole MLP, e.g., by the vectorization of the block matrix incorporating parameter vectors of all layers $\boldsymbol{\theta} = \text{vec}((\boldsymbol{\theta}_1 \quad \boldsymbol{\theta}_2 \quad \ldots \quad \boldsymbol{\theta}_L))$.

Through stacking multiple layers, a hierarchical order is achieved which is not the case in the regime of flat neural networks or single layer perceptrons. A hierarchical structure facilitates to learn more expressive representations of the input data and consequently obtains a more powerful model in contrast to the flat structured models.

This can be motivated from the neuroscientific perspective where hierarchical structures are present throughout the brain. Brodmann (1909) showed that the neocortex develops from six layers in the mammalian brain, which is the part of the brain responsible for the execution of higher-order brain functions, including cognition, sensory perception, and sophisticated motor control (Lodato and Arlotta, 2015). For example, it requires approximately 0.1 seconds for someone to visually recognize their mother. The sequence of neuron firings that can take place during this short time interval cannot possibly be longer than a few hundred steps due to the switching speed of biological neurons where the fastest neuron switching times are known to be on the order of 10^{-3} seconds (Mitchell, 1997, p. 82). These facts vividly demonstrate the effectiveness of sequentially combining layers of neurons in a hierarchical order so that particular groups of neurons are dedicated to solving specific subtasks in order to make complex decisions.

From another perspective, stacking multiple layers can be motivated by aiming to maximize the expressive power of the network from a theoretical point of view. Cybenko (1989) and Hornik et al. (1989) showed that approximating arbitrary but continuous functions with a neural network architecture with two layers of neurons is possible up to an arbitrarily small error (under a finite norm) (e.g., one hidden and the output layer in an MLP with three layers). The same holds for arbitrary functions (then including the non-continuous ones, too), which can be approximated with neural networks of three layers of neurons (e.g., a four-layer MLP) (Cybenko, 1988). For both proofs, the required number of neurons in the hidden layers depends on the function to be approximated (Mitchell, 1997, p. 105). Moreover, Siegelmann and Sontag (1992) even proved the *Turing completeness* of finite Recurrent Neu-

ral Networks (RNNs) (Goodfellow et al., 2016, ch. 10; Sherstinsky, 2018), which are neural networks architectures that are capable of processing sequences with varying length of consecutive inputs such as sentences in natural language and holding an internal state over the course of processing a single sequence, and such concrete ones with a surprisingly low number of units (about a thousand units only).

The general definition of an MLP allows choosing the number of hidden layers $L-2 \geq 0$ arbitrarily to construct "deeper" neural network architectures. Learning deep neural network architectures motivates the term *deep learning* (LeCun et al., 2015), which earned widespread popularity over the past years due to the tremendous successes in various fields. However, the question arises whether to go beyond shallow networks like MLPs with one or two hidden layers since, according to the theoretical findings of the expressive strength as described above, this seems to nullify any reason for using deeper neural networks. In fact, it has to be distinguished between the capability of representing a function with a neural network (or the approximation quality), which is in line with the expressive strength of the neural network, and learning a function from a finite (training) data set of data samples associated with this underlying function. Here, other difficulties might occur such as an insufficient amount of data or noisy data targets that can evoke optimization issues such as over- or underfitting (Goodfellow et al., 2016). Therefore, for training neural networks, deeper neural networks should be preferred over shallow ones to achieve better generalization (Bengio, 2009) and, moreover, deep networks can be more effective than shallow ones even with a comparably lower number of neurons as empirically shown by Goodfellow et al. (2014), for example.

2.2.4 Gradient-based learning

After presenting the construction of neurons and their function of producing outputs based on the inputs and learned weights, it remains to answer the question of how the neurons are able to learn such weights in order to produce the desired outputs. As mentioned earlier, Hebb's rule (Hebb, 1949) inspired the update rule in the perceptron algorithm, and this approach can indeed train a single neuron or a single layer of neurons whereas this does not generalize to neural networks with multiple layers of neurons (e.g., MLPs). To address this incapability another and more general optimization strategy is commonly used: *gradient descent optimization*. Combined with the *backpropagation* algorithm (described later), this caused the *second neural network hype* back in the 1980s by facilitating the learning of MLPs. Furthermore, its general formulation allows for the appliance on more sophisticated neural network architectures such as CNNs (see Sec. 2.3) and RNNs (Goodfellow et al., 2016, ch. 10; Sherstinsky, 2018), which are a significant contribution to the current success of deep learning over various areas. So, the simplicity of this optimization approach is even more surprising.

Gradient Descent Optimization:

Gradient descent optimization method, which is an iterative first-order optimization method, which gradually adjusts the model's parameters $\boldsymbol{\theta}$ through first-order information by determining the gradient of the error (or loss) on the training data $\mathcal{D}$ with respect to the parameters over several iterations, which are commonly referred to as epochs, in order to reach a local minimum.

As a short example, optimizing according to the gradient descent method can be compared to finding oneself in the mountains (representing the shape of the loss function) blindfolded and trying to find a valley (a local minimum). If you now determine the gradient under your feet at your current location (which represents the current parameter vector $\boldsymbol{\theta}$) and follow this gradient downhill when taking the next step, you will reach a valley after some time since at a valley there is no gradient which can be

followed for a successive step. Note that this valley/minimum has to be neither a global minimum nor a good minimum at all, nor does it actually have to be a minimum, since it could also be a saddle point.

In the supervised learning setting the loss for a data sample $(\boldsymbol{x}^{(d)}, y^{(d)}) \in \mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$ expresses the error between the model's output $o^{(d)} = f(\boldsymbol{x}^{(d)}; \boldsymbol{\theta})$ given input $\boldsymbol{x}^{(d)}$ and the data target $y^{(d)}$ and is denoted as $l(o^{(d)}, y^{(d)}; \boldsymbol{\theta})$. Considering a batch of data samples (e.g., the whole training data set $\mathcal{D}$) the losses for each data sample have to be aggregated (often via an average over the individual data sample losses) in order to obtain a single scalar loss representing the performance on the particular sample batch. The notation of a parametric model $o = f(\boldsymbol{x}; \boldsymbol{\theta})$ (such as a neural network) can be extended to process a batch of input samples $\boldsymbol{X} = (\boldsymbol{x}^{(1)} \quad \boldsymbol{x}^{(2)} \quad \dots \quad \boldsymbol{x}^{(D)})^\top$ of D data samples (as a block matrix) through the model yielding a vector of outputs $\boldsymbol{o} = (o^{(1)}, o^{(2)}, \dots, o^{(D)})^\top$ instead of a single scalar output as follows

$$\boldsymbol{o} = \boldsymbol{f}(\boldsymbol{X}; \boldsymbol{\theta}) = \begin{pmatrix} f(\boldsymbol{x}^{(1)}; \boldsymbol{\theta}) \\ f(\boldsymbol{x}^{(2)}; \boldsymbol{\theta}) \\ \vdots \\ f(\boldsymbol{x}^{(D)}; \boldsymbol{\theta}) \end{pmatrix}$$

for the sake of conciseness through such notation in matrix-from.

To minimize the error being made on the entire training data set $\mathcal{D}$, the aggregated loss function, which the gradient descent optimization is applied on, is defined as the error between the (vector of the) model's outputs $\boldsymbol{o} = \boldsymbol{f}(\boldsymbol{X}; \boldsymbol{\theta})$ regarding the (block matrix of the) training input samples $\boldsymbol{X} = (\boldsymbol{x}^{(1)} \quad \boldsymbol{x}^{(2)} \quad \dots \quad \boldsymbol{x}^{(D)})^\top$ and the corresponding (vector of the) data targets $\boldsymbol{y} = (y^{(1)}, y^{(2)}, \dots, y^{(D)})^\top$. Formally, as follows

$$L(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) = \sum_{d}^{D} \frac{1}{D} l\left(f(\boldsymbol{x}^{(d)}; \boldsymbol{\theta}), y^{(d)}; \boldsymbol{\theta}\right). \tag{2.2}$$

As the gradient is utilized to adjust the parameters so that the loss decreases, the weights are updated in the opposite direction of the gradient, i.e., the negative of the gradient, since the gradient of the loss with respect to the parameters indicates the direction in which the loss increases. This leads to the gradient descent update rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \cdot \nabla_{\boldsymbol{\theta}_t} L(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}_t)$$

where $\boldsymbol{\theta}_{t+1}$ denotes the updated parameters, which are used in the successive training epoch, $\boldsymbol{\theta}_t$ denotes the parameters of the current epoch the gradient has been computed with respect to, and the learning rate (or step size) hyperparameter α is a positive real-valued scalar which is usually chosen to be much less than 1, i.e., $0 < \alpha \ll 1$, in order to limit the magnitude of the gradient leading to a restricted change in the parameter update. A low learning rate addresses the problem of overshooting or oscillating around a local minimum in the loss function where gradient descent with a high learning rate suffers from. However, this has the disadvantage that the convergence speed might be lower due to the small update steps in the parameters. To ensure convergence to a local optimum the sufficient condition

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty \tag{2.3}$$

must be met, e.g., by using a learning rate schedule which decays the learning rate over time where α_t denotes the learning rate applied in the tth epoch of learning (Goodfellow et al., 2016, ch. 8.3.1, p. 291).

In practice, more sophisticated optimizers build upon the method of gradient descent and are applied because of faster convergence and reduced effort in learning rate schedule tuning (since some implicitly ensure the conditions from Eq. 2.3). These optimizers store and (mostly exponential moving) averaging the previous gradients. Further, they derive their improvements from this additional information by scaling the size of the current gradient (which is directly equivalent to scaling the learning rate) to, for example, finely approach to a minimum and prevent oscillating around it, and add the current momentum to the gradient in order to move faster (i.e., performing larger update steps) when the previous gradients point in a similar direction. For example, such sophisticated and well-known optimizers are RMSProp (Tieleman and Hinton, 2012), AdaDelta (Zeiler, 2012), Adam (Kingma and Ba, 2014), or AMSGrad (Reddi et al., 2018).

Backpropagation:

Considering the appliance of the *gradient descent* method on a deep neural network, there are some major difficulties arising from the depth of the network architecture. On the one hand, determining the gradient analytically with respect to the parameters $\boldsymbol{\theta}$ in the network might be error prone and have to be redone whenever the network's architecture is slightly changed. On the other hand, computing the gradient with respect to the parameters $\boldsymbol{\theta}$ can be computationally demanding when the computation of the partial derivative is performed for each parameter separately, and intermediate results might be calculated redundantly. Fortunately, these matters are being addressed by the *backpropagation* (or backprop) algorithm (Rumelhart et al., 1986). The backpropagation algorithm is often claimed to be an efficient application of the *chain rule* from calculus on neural networks or, more generally, on *computational graphs*. First of all, a computational graph is a directed acyclic graph whose nodes denote – mostly simple – mathematical operations (such as addition, multiplication, etc.) and the inflowing edge(s) provide the values of the operands whereas outgoing edge(s) yield the result(s) of the computation being expressed by this node. See Fig. 2.5 for an example of a computational graph. A computational graph can express arbitrarily complex computations by breaking them down into simple, immediate subcomputations what applies for ANNs. By applying input values (e.g., the input data and the current parameters in the case of a neural network) on the edges of the first/input nodes of the computational graph and propagating the values through the graph (similar to propagating an input through a neural network) leads to the result at the edges of the last nodes of the graph. This is called a forward pass or forward propagation. If the intermediate results of each node in the computational graph are stored in the forward pass and the partial derivatives of the output edges(s) of a node with respect to all input edges are known for all the operations/nodes used throughout the graph, we are ready to use the backpropagation algorithm to determine the gradient of the computational graph's output with respect to particular graph inputs. An example of the backpropagation algorithm applied to a computational graph is shown in Fig. 2.5. The algorithm is initialized with storing a gradient of 1 of the output node (with respect to the output) obviously since generally $\partial o / \partial o = 1$ applies. Then, the algorithm computes the gradients with respect to the inflowing edges beginning with the last nodes of the computational graphs all the way up to the input edges the gradient with respect to is desired. This takes place by propagating and storing the gradients of the intermediate nodes in a backward manner, which are efficiently reused without the need of reevaluating the partial derivatives of any node. According to the *chain rule*, the gradients are successively built by calculating the gradient of the currently considered node with respect to the input edges and multiplying this intermediate gradient with the gradient of the downstream node which yields the gradient of the graph('s outputs) with respect to the input edges of the currently considered node. This step in the backpropagation algorithm is readily apparent in the definition of the chain

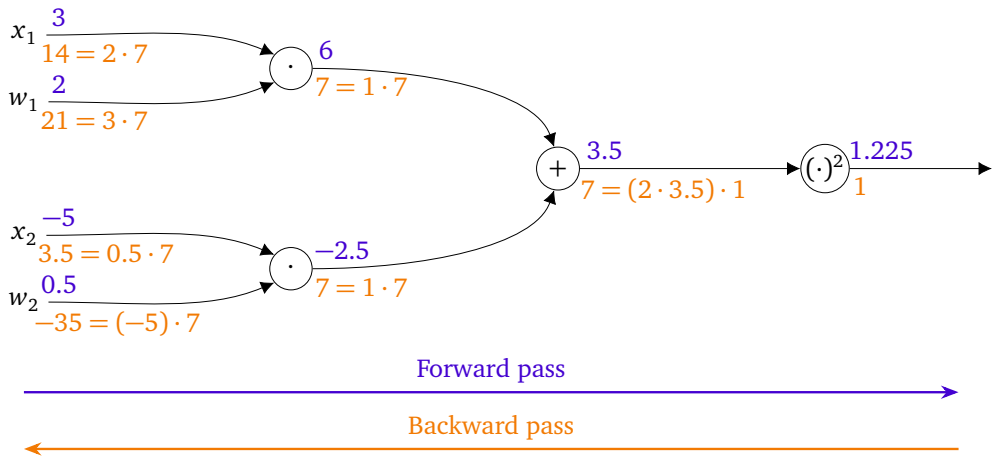


Figure 2.5.: Visualization of the backpropagation algorithm applied on a computational graph with the inputs $x_1 = 3$, $w_1 = 2$, $x_2 = -5$ and $w_2 = 0.5$. The purple numbers are all (immediate) results from the forward pass whereas the orange ones are the (immediate) gradients (of the output 1.225 with respect to the inputs of the currently considered node) determined in the backward pass according to the backpropagation algorithm. This computational graph could represent a neuron with two input activations, no bias parameter and (uncommon) quadratic activation function. The gradient with respect to the inputs of interest (the current weights $w_1 = 2$ and $w_2 = 0.5$) can be easily obtained ($\partial o / \partial w_1 = 21$ and $\partial o / \partial w_2 = -35$).

rule using Leibniz's notation: Given $y = (f \circ g)(x) = f(g(x))$ with the intermediate result $z = g(x)$ the chain rule states

$$\frac{\partial}{\partial x} f(g(x)) = \frac{\partial y}{\partial x} = \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x}$$

which can obviously be extended by combining arbitrarily many functions. The gradients obtained with respect to the ANN's parameters $\boldsymbol{\theta}$ are now used for gradient descent optimization to achieve the learning of the ANN. The directions of the forward and backward pass through an MLP are depicted in Fig. 2.4.

Expressing an ANN as a computational graph in order to perform the backpropagation algorithm for the gradient computation automatically (often called *autodiff* meaning *automatic differentiation* for short) is performed under the hood of most frameworks used for neural network implementations and learning such as *PyTorch* (Paszke et al., 2017), which was used to perform experiments for this thesis, or *Tensorflow* (Abadi et al., 2016). In most cases, the programmer has to define the network architecture, i.e., the forward pass, only and the differentiation is performed automatically when operations are used, which are defined within the framework (including the partial derivatives) or partial derivatives are explicitly provided.

Gradient estimation on batches:

As one can imagine, depending on the size of the training data set and the chosen learning rate, gradient descent might need many epochs, i.e., many parameter update computation steps, to reach a local minimum. Hence, it would be advantageous to determine the gradient on a subset of the entire training data set solely in order to perform a parameter update step based on this gradient. This would bring the advantage of updating the parameters much more frequently in the same amount of time and eventually reach convergence faster. A justification for using a subset only paired with a slightly modified gradient descent method, called *mini-batch gradient descent*, is shown in the following derivation (adapted from Schulz and Aziz (2018)).

Recall that the goal is to minimize the loss on the entire training data set and such loss function is defined as shown in Eq. 2.2 by taking each training data sample $(\boldsymbol{x}^{(d)}, y^{(d)}) \in \mathcal{D}$ into account. Hence, the gradient of interest can be reformulated as

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \sum_{d}^{D} \frac{1}{D} l\left(f(\boldsymbol{x}^{(d)}; \boldsymbol{\theta}), y^{(d)}; \boldsymbol{\theta}\right) = \sum_{d}^{D} \frac{1}{D} \nabla_{\boldsymbol{\theta}} l\left(f(\boldsymbol{x}^{(d)}; \boldsymbol{\theta}), y^{(d)}; \boldsymbol{\theta}\right).$$

Considering the sum followed by the factor $1/D$, it is apparent that this corresponds to the definition of an expectation of a discrete and uniformly distributed random variable X describing the index of a data sample in the training data set $\mathcal{D}$ and each index d, i.e., each data sample $(\boldsymbol{x}^{(d)}, y^{(d)})$, is taken equally likely with a probability of $P(X = d) = 1/D$. This insight allows for reformulating

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) = \mathop{\mathbb{E}}_{X \sim \mathcal{U}(1,D)} \left\{ \nabla_{\boldsymbol{\theta}} l\left(f(\boldsymbol{x}^{(X)}; \boldsymbol{\theta}), y^{(X)}; \boldsymbol{\theta}\right) \right\}$$

where $\mathcal{U}(1, D)$ denotes the discrete uniform distribution parameterized so that it is defined on $\{1, 2, \ldots, D\} \subset \mathbb{N}$. Yet, we can take M samples of X and compute the mean over the gradients of the loss for each sample and we would have a Monte Carlo estimate of the gradient

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) = \sum_{m=1}^{M} \nabla_{\boldsymbol{\theta}} l\left(f(\boldsymbol{x}^{(d_m)}; \boldsymbol{\theta}), y^{(d_m)}; \boldsymbol{\theta}\right) \quad \text{where} \quad d_m \sim \mathcal{U}(1, D) \quad \text{(w.o. replacement)}$$

or, more informally speaking, we replace the expectation by sampling via Monte Carlo simulation (Anderson, 1999). By choosing the mini-batch size $M \ll D$, the computational effort per gradient estimate, which requires more frequent parameter updates but introduces gradients with higher variance than the gradient of Eq. 2.2. The gradient estimates are therefore also referred to as stochastic gradients where the stochasticity (or variance) increases when the mini-batch size M is decreased. This leads to the special case of setting the $M = 1$ so that the gradient is estimated based on a single data sample only, which is called *stochastic gradient descent* but it has to be noted that the distinction between the terms of mini-batch gradient descent and stochastic gradient descent is not that strict and clear in the literature where the stochastic gradient descent terminology is sometimes used in combination with mini-batches. Generally, the hyperparameter M controls the trade-off between the accuracy of the gradient estimate and computation time. However, when M is set too low, this might require a small learning rate to maintain stability due to the high variance in the gradient estimate what, on the other hand, might lead to a disproportionately high convergence time (Goodfellow et al., 2016, ch. 8.1.3, p. 276).

2.3 Convolutional Neural Networks (CNNs)

The presented neural network architecture of simply stacking several layers of neurons upon each other (i.e., MLP) can be further extended to more sophisticated neural network architecture designs. The architecture of *Convolutional Neural Networks (CNNs)* (LeCun et al., 1998; Fukushima, 1980) has become indispensable in today's field of deep learning and is presented in this section.

The main inspiration for convolutional neural networks is visual information processing in the mammal brain, more precisely the primary visual cortex (V1). There, the signals are interpreted in stages with nerve cells firing at specifically oriented edges (the simple cells) or, based on this activations, at edges moving in a particular direction (the complex cells) (Hubel and Wiesel, 1963). The information signals

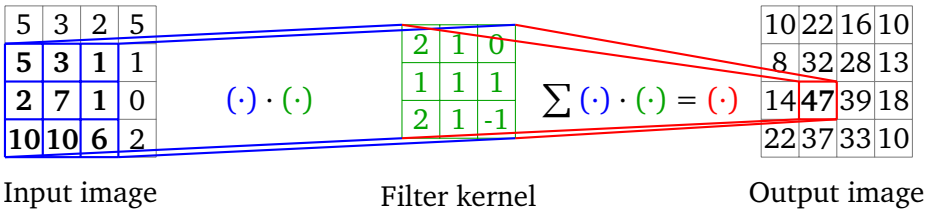


Figure 2.6.: Visualizes the convolution operation with a 3x3 kernel exemplarily where zero padding is applied. The currently considered input and output pixels are printed in boldface and the filter kernel's weights in green. The colored (·) notation refers to the elements from the specific image/kernel the color is associated with.

thus become more and more abstract in subsequent areas/layers of the brain (Brodmann, 1909), up to so-called grandmother cells, i.e., (hypothetical) nerve cells that fire when a specific person is recognized (Gross, 2002). Since convolutional neural networks, or more precisely, convolutional layers, follow the conceptional idea of recognizing edges and subsequently more abstract visual patterns like the nerve cells in the V1, they are successfully used in the field of image processing (e.g., image classification, as in the experiments in this thesis). Nevertheless, the use of convolutional neural networks could also be surprisingly profitable in other areas, e.g., in natural language processing (Kim, 2014).

Further motivation behind the use of convolutional neural networks concerns reasons for computing efficiency since visual input data is high-dimensional (i.e., dimensions = the number of pixels times the number of color channels). Here, the use of a standard MLP architecture is very computationally intensive since the number of neurons and, consequently, the number of weights would be extremely high due to the fully-connected layers. Be aware that the input layer alone would already contain as many neurons as input dimensions, whose output activations would then be sent to each neuron in the next layer. Thus each of these neurons must have a single weight parameter for each of these many activations, which results in a tremendously high number of weights. To make this point more clear, consider colored images of 32x32 pixels (with a color depth of 3), which is a $32 \cdot 32 \cdot 3 = 3\,072$-dimensional input. If an MLP architecture with the first hidden layer only consisting of 1 000 neurons is applied, the number of weights in this very first layer would be 3 072 000. As a rough comparison, the successful and state-of-the-art `ResNet` architecture (He et al., 2015) with 110 layers defines a convolutional neural network, which would only contain 1 730 714 parameters in total (for a 10-class classification problem).

Convolutional neural networks consist of two additional components that are not present in the MLP design: the convolutional layer and the pooling layer. Both types of layers are explained in detail in the following sections. A convolutional layer contains the neural network's parameters and can be considered as a replacement for the layers of neurons (fully-connected layers) as used in the MLP architecture. A pooling layer does not contain any parameters and is ordinarily placed after a convolutional layer. This is repeated several times at the beginning of the neural network to compress the high-dimensional input into informational features (i.e., of a lower dimension) that are then processed in standard (fully-connected) layers of neurons up to the output layer.

2.3.1 Convolutional layer

The basic operation behind a convolutional layer is the *convolution* (i.e., linear filtering) from the field of computer vision. Filtering, in general, refers to the manipulation of pixel values of an image. Therefore, a filter kernel is specified, which determines the calculation of a new value for each pixel by additionally including adjacent pixels. A linear filter kernel (i.e., convolution) has a specific (commonly square)

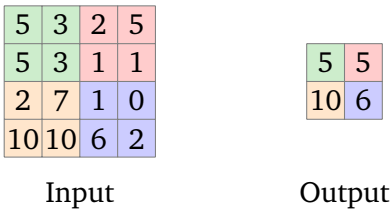


Figure 2.7.: Visualizes max pooling on 2x2 regions exemplarily. The colored areas highlight over which values the max function is applied to determine the output values.

shape, which tells how many adjacent pixels are considered. For example, a kernel with a shape of 3x3 includes the eight direct neighbor pixels (in vertical, horizontal and diagonal direction) to calculate the filtered value for the considered pixel (in the center of the kernel). The kernel and its filtering behavior are further specified by weights that are used to build a weighted sum of the considered pixel and its adjacent pixels as the new pixel value. Frequently, filtering is imagined so that the filter kernel slides over the image and calculates the output image pixel by pixel. An example of filtering is visualized in Fig. 2.6 where zero padding was applied, which means that missing adjacent pixels needed for the kernel calculation at the edges of the input image are filled with zeros as placeholder values (= black pixels).

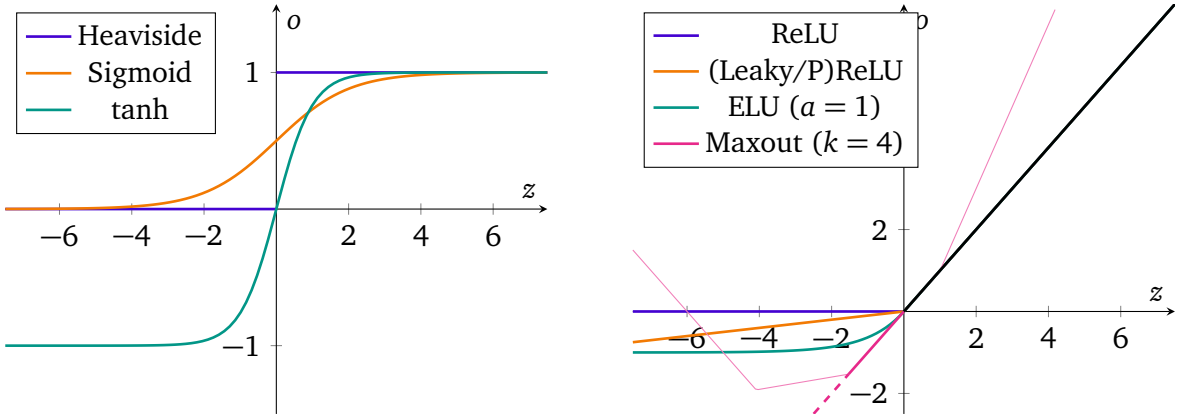
Depending on the kernel's weights, a convolution can express different filter behaviors. For example, blurring, sharpening and, most importantly, edge detection. Depending on the weights, differently oriented edges can be detected, which is where the connection to the functioning of the simple cells in the primary visual cortex should become clear.

As the term weight suggests, one could ponder to learn the weights of a filter kernel as any other parameters in a neural network. In order to extract more diverse properties from the input (e.g., differently aligned edges), several filters could be applied to the same input. This is exactly the case in a convolutional layer. It consists of several convolution kernels that are applied to the input activations for this layer where the weights are learned individually. Input activations can be either the pixel values of an input image that is applied in the input layer or activations from a previous convolutional layer. Note that the output of a convolutional layer can be considered as an image with the same number of color channels as convolutions are applied in the convolutional layer. Further, the convolution operation is applicable to inputs with arbitrary (color) depth where the kernel extends to a tensor instead of a matrix.

Just like in a standard MLP layer of neurons, a bias parameter can be added to each convolution output, and an activation function is applied element-wisely. It should become apparent that this definition can drastically reduce the number of parameters compared to standard MLP layers since the number of parameters does only depend on the number of convolutions and their kernel sizes (and of the depth of the input) but does not correlate one-to-one with the input dimensionality. This implies that the same weights interact with multiple input activations to generate the output, which is a kind of weight sharing. Further note that as the convolution operation is linear by definition (Goodfellow et al., 2016, ch. 9, p. 326), this can be implemented efficiently in matrix/tensor operations.

2.3.2 Pooling layer

A pooling layer is different from standard MLP layers of neurons since it does not contain anything that could be considered as neurons or parameters at all. It is primarily used to reduce the spatial size of the output of a convolutional layer for computational reasons, on the one hand, but also helps to make the



(a) Sigmoidal activation functions

(b) Rectifier activation function variants

Figure 2.8.: Plots of **(a)** sigmoidal and **(b)** (variants of) rectifier activation functions mapping the neuron's pre-activation z to the final output (activation) o. As all rectifier activation function variants have the same (linear) image for $z \geq 0$, this part is plotted uncolored in black. The orange rectifier activation function variant in **(b)** can be seen as the Leaky ReLU as well as the parametric ReLU function (PReLU) with the slope parameter set to 0.1. The actual output o of a maxout unit (pink in **(b)**) does not depend on a single pre-activation z unlike all other activation functions presented. Maxout maps the pre-activation value z directly to the output o (thick pink and partly black line) whenever no other pre-activation (thin lines) is higher.

representation, the neural network learns, invariant to small translations of the input (Goodfellow et al., 2016, ch. 9, p. 336). The reduction takes place by pooling the activations within subdivisions similar to a convolution kernel, which means that a single value is produced based on the input activations in each pooling region (i.e., subdivision). In contrast to filtering, this cannot be imagined as a kernel sliding pixel by pixel over the image but jumps so that the areas do not overlap, i.e., no pixel is involved in several output calculations. The most common choice for a pooling operation is the *max pooling* (Zhou and Chellappa, 1993) where the maximum of the values within the considered area makes the output. Max pooling is exemplarily visualized in Fig. 2.7.

2.4 Activation functions

The *activation function* is a single-variable and scalar-valued function, which is applied to transferring the pre-activation signal z in a neuron and this result is used as the output (activation) o (thus, also called transfer function). The activation function is commonly of (strictly) monotonically increasing, differentiable, and bounded nature. Originally, it is inspired by the *all-or-none* principle of nervous activity where a certain threshold must be exceeded before the neuron fires a signal. This can be modeled by the *Heaviside step function* (purple in Fig. 2.8a) but it is utterly unsuitable in neural networks when training with a gradient-based optimization method (e.g., gradient descent paired with backpropagation) is planned since the derivative is zero wherever the function is differentiable, which would result in no parameter adjustments for all parameters in the neural network. Apart from the neuroscientific inspiration, another compelling reason why a (non-linear) activity function is required in a neural network is that only then does the stacking of layers improve the expressive power of the neural network (or model) whereas the model would remain linearly separating by stacking layers with linear activation functions or even without any activation functions put in.

2.4.1 Sigmoidal activation functions

As close approximations of the Heaviside step function and satisfying the requirements of differentiability and the existence of non-zero values in the derivative, the family of sigmoidal functions can be taken into account.

Sigmoid function:

The *sigmoid function* $\sigma(\cdot)$ is a special case of the *logistic function* and the most prominent function within the family of sigmoidal functions and was heavily used in the early history of neural networks (see the orange curve in Fig. 2.8a). It can be construed as a smooth (i.e., continuous and differentiable) variant of the Heaviside step function, which means that the inputs are also squashed into the range of $[0,1]$ with $\lim_{z\to-\infty}\sigma(z)=0$ and $\lim_{z\to\infty}\sigma(z)=1$.

The sigmoid function and its derivative are defined as follows

$$\sigma(z)=\frac{1}{1+\exp(-z)}=\frac{\exp(z)}{\exp(z)+1}\quad\text{and}\quad\frac{\partial\sigma(z)}{\partial z}=\sigma(z)(1-\sigma(z)).$$

One drawback in using the sigmoid function as activation function is that the derivative is nearly zero for z values being of a high magnitude of about $|z|>4$ producing very small gradients, which are back-propagated (by multiplication according to the chain rule) through the network (or computational graph) to all parameters of upstream nodes. This phenomenon is called the *problem of vanishing gradients*. Whenever the pre-activation z is of a high magnitude so that the resulting gradient is quite small, the neuron is called saturated since it would take many training iterations (i.e., parameter update steps) to adjust the parameters so that the magnitude of z will decrease and the gradients become larger again.

Another disadvantage is that the mapping of the sigmoid function is not zero-centered whereby a bias in the gradient is induced, which is an undesired property assuming that the data has been normalized and zero-centered beforehand (which is common practice) (LeCun et al., 1998).

Hyperbolic tangent:

The *hyperbolic tangent* function $\tanh(\cdot)$ (see the green curve in Fig. 2.8a) is a shifted and scaled version of the sigmoid function (Goodfellow et al., 2016, ch. 6.3.2, p. 191) which does not suffers from the disadvantage of not being zero-centered. For that reason, the hyperbolic tangent function is preferred over the sigmoid function as a choice of the activation function. The function and its derivative are defined as follows

$$\tanh(z)=\frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}=2\sigma(2z)-1\quad\text{and}\quad\frac{\partial\tanh(z)}{\partial z}=1-\tanh^2(z).$$

Note that the problem of saturating neurons using the hyperbolic tangent activation function is still present. Moreover, this is not only a specific problem of the sigmoid and hyperbolic tangent function but rather a serious limitation the entire family of sigmoidal function suffers from.

2.4.2 Rectifier activation function variants

The widespread problem of saturating sigmoidal units can make gradient-based learning very difficult especially for very deep neural networks. For this reason, their use as hidden units in feedforward neural

networks is now discouraged (Goodfellow et al., 2016, ch. 6.3.2, p. 191). Note that using rectifier activation function variants results in the neural network approximating the function that is aimed to be learned with a piece-wise linear function.

Rectifier (linear unit (ReLU)):

The *rectifier linear unit*, *ReLU* for short, $\rho(\cdot)$ denotes a neuron using the rectifier function as activation function, which is plotted in purple Fig. 2.8b.

The ReLU function and its derivative are defined as follows

$$\rho(z) = \max(0, z) \quad \text{and} \quad \frac{\partial \rho(z)}{\partial z} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0 \end{cases}$$

where one can see that the rectifier is not differentiable in zero. As it is required that the activation function is differentiable (in all values of z) in order to perform gradient-based optimization, the value of the derivative at zero can be chosen to be 0 or 1 where the letter choice evokes the derivative to become the Heaviside step function. However, this choice is of little importance in practice because of the pre-activation z rarely approaches exactly zero.

The problem of saturation does not exist for positive inputs anymore. Furthermore, the gradients for positive inputs are of constant magnitude ($= 1$) and does not begin to vanish. For that reason, training very deep neural networks is much faster if the hidden layers are composed of ReLUs as shown by Glorot et al. (2011) as well as Krizhevsky et al. (2012), who observed a speed up in convergence time by a factor of six for the CIFAR-10 image data set. Considering the rectifier for negative inputs z the gradient is exactly zero regarding such pre-activations z. This is an extreme case of saturation since parameters and, consequently, the unit's pre-activation production behavior will not change anymore as no parameter updates apply why such neurons are commonly called dead. In order to avoid having dead neurons immediately after initialization, the bias of a neuron can be initialized with a slightly positive value.

Unfortunately, the output activation of a ReLU is not zero-centered.

Leaky ReLU:

The *leaky ReLU* $\rho^{\text{Leaky}}(\cdot)$ (orange in Fig. 2.8b) addresses the problem that the neurons will not die over the course of training. A negative pre-activation z in such neurons is not mapped to exactly zero anymore and produces small gradients. This is achieved by tilting the negative side of the rectifier slightly, leaving an eponymous leak, that allows negative pre-activations to pass down-scaled. The slope of the linear mapping on the negative side is set to a small positive value such as 0.1 or 0.01.

The Leaky ReLU function and its derivative are formally defined as follows

$$\rho^{\text{Leaky}}(z) = \max(0.1z, z) \quad \text{and} \quad \frac{\partial \rho^{\text{Leaky}}(z)}{\partial z} = \begin{cases} 1 & \text{if } z > 0 \\ 0.1 & \text{if } z < 0 \end{cases}.$$

All in all, this has no disadvantages compared to the standard ReLU.

Parametric ReLU:

The *parametric ReLU*, *PReLU* for short, $\rho^{\text{PReLU}}(\cdot)$ can be seen as a generalization of the (leaky) ReLU (see orange in Fig. 2.8b) as the slope of the negative side of the rectifier function is controlled by

an additional parameter for each neuron which is learned through training via backpropagation like a classical parameter in a neuron (or, alternatively, held fixed and set beforehand, which makes it a hyperparameter). Hence, the PReLU can be comprehended as a leaky ReLU where the degree of the leak (on the negative side) is controlled by the neuron itself.

The PReLU function is defined as follows

$$\rho^{\text{PReLU}}(z) = \begin{cases} z & \text{if } z > 0 \\ az & \text{otherwise} \end{cases}.$$

Exponential ReLU (ELU):

The *exponential ReLU*, *ELU* for short, $\rho^{\text{ELU}}(\cdot)$ (see the green curve in Fig. 2.8b) has all benefits of (leaky) ReLU but is closer to zero-mean output activations and speeds up the training further compared to (leaky) ReLUs (Clevert et al., 2015).

The ELU function is defined as follows

$$\rho^{\text{ELU}}(z) = \begin{cases} z & \text{if } z > 0 \\ a\,(\exp(z) - 1) & \text{otherwise} \end{cases}$$

where $a \geq 0$ is a hyperparameter, which can be set to 1 by default.

Note that this variant of the rectifier function is differentiable in all values of z.

Maxout units:

A *maxout unit* (Goodfellow et al., 2013) further generalizes the (leaky) ReLU making it possible to approximate arbitrary convex functions with a single maxout unit (with a proper number k of affine transformations) by computing the maximum over linear functions. The key difference to classical neurons with rectifier activation functions is that within a maxout unit not only a single pre-activation is produced but k many. Each pre-activation z_j with $j = 1, \ldots, k$ is produced by a linear model (or any other affine transformation is possible) as usual in an artificial neuron with distinct parameters for each of the k pre-activations but all of them receive the same input (vector) $\boldsymbol{x}$. As activation function, the max-operator is applied over the k pre-activations and aggregating them into a single output for this unit/neuron. To get a better understanding, see Fig. 2.8b for an exemplary visualization of a maxout unit activation for $k = 4$ (in pink).

Apart from the increased expressive power of a single (maxout) unit paired with the benefits of (P)ReLU, a major drawback of maxout units is that the number of parameters per unit increases by the factor of k. However, it turns out that maxout networks are able to outperform networks with rectifier activations using fewer neurons and even less parameters (see the empirical results in Goodfellow et al. (2013)).

2.5 Loss functions

The choice of the loss function is a crucial factor for successful learning and is task-dependent in most cases.

2.5.1 Mean squared error

Given a regression problem, the mean squared error (MSE) is an appropriate choice and measures the mean squared difference between the model's output $o^{(d)} = f(\boldsymbol{x}^{(d)}; \boldsymbol{\theta})$ and the data target $y^{(d)}$ for all data samples in the training set $(\boldsymbol{x}^{(d)}, y^{(d)}) \in \mathcal{D}$. This can also be comprehended as the squared Euclidean distance (L^2-*norm*) of the difference between the (vector of the) model's outputs $\boldsymbol{o} = \boldsymbol{f}(\boldsymbol{X}; \boldsymbol{\theta})$ regarding the (block matrix of the) training input samples $\boldsymbol{X} = (\boldsymbol{x}^{(1)} \quad \boldsymbol{x}^{(2)} \quad \dots \quad \boldsymbol{x}^{(D)})^\top$ and the corresponding (vector of) data targets $\boldsymbol{y} = (y^{(1)}, y^{(2)}, \dots, y^{(D)})^\top$ and eventually being reduced by dividing with the number of training samples D

$$L(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) = \text{MSE}(\boldsymbol{o}, \boldsymbol{y}) = \sum_{d}^{D} \frac{1}{D} (f(\boldsymbol{x}^{(d)}; \boldsymbol{\theta}) - y^{(d)})^2 = \frac{1}{D} \|\boldsymbol{f}(\boldsymbol{X}; \boldsymbol{\theta}) - \boldsymbol{y}\|_2^2$$

where $\|\cdot\|_2$ denotes the L^2-norm.

This represents the most intuitive mathematical formulation of an error in the form of distance-based deviation from the data targets. Furthermore, it is differentiable what is not so for the absolute difference between output and data target or further the *Manhatten distance* (or L^1-*norm*) between the vector difference.

2.5.2 Cross entropy loss

MSE is improper for training on multi-class classification problems (i.e., where the number of classes is $n > 2$) in combination with the model output trained on *one-hot encoded* targets since we do not care about exact values but rather are interested in the class prediction represented by the highest value in the output vector as well as the match between this class prediction and the data target. The class prediction is represented by the *argmax* operation (abbreviation for arguments of the maxima) of the model's output vector $\boldsymbol{o} = (o_1, o_2, \dots, o_n)^\top \in \mathbb{R}^n$, which yields the index of the class (i.e., component of the output vector) the highest output value assigned

$$\hat{y} = \underset{i=1,\dots,n}{\text{argmax}}(o_i).$$

Alternatively, this can be seen as a ranking, which means that the classes are sorted by the output values (also called scores in the context of rankings) in descending order, and we choose the top-ranked class (the class at the first position in the ranking) as the model's prediction.

Hereafter, we assume that the model's output vector o is in the form of a discrete probability distribution over the classes. Formally, the necessary conditions

$$\sum_{i=1}^{n} o_i = \mathbb{1}^\top \boldsymbol{o} = 1 \quad \text{and} \quad o_i \geq 0 \quad \forall i = 1, 2, \dots, n \tag{2.4}$$

do apply. Applying the so-called *softmax* function on the outputs squashes the values into a probability distribution to fulfill these conditions while preserving the ranking of the classes as described in Sec. 2.6. Note that the one-hot encoding vector of a true (data) target also meets the conditions and is, therefore, a discrete probability distribution which assigns probability of one (= 100%) to the correct class and probability of zero to all the other classes.

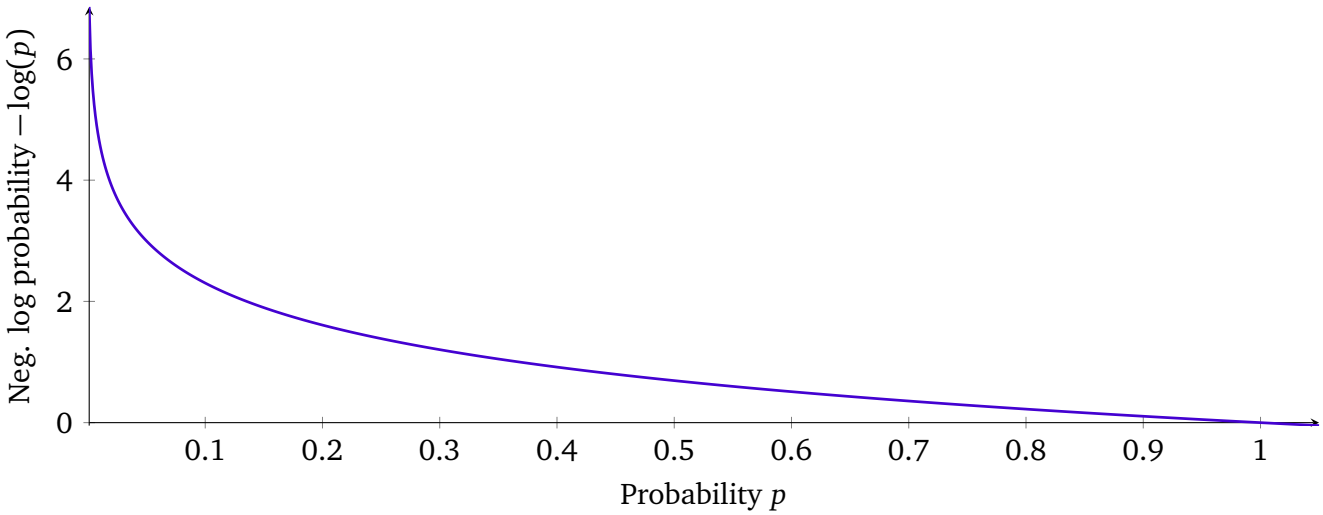


Figure 2.9.: This plot of $p \mapsto -\log(p)$ visualizes the penalization effect within the cross entropy loss, especially the high penalization when the probabilities predicted for the target class are low.

When the *cross entropy loss* is put to use in a classification learning problem (coupled with one-hot encoded true targets), the loss aims to penalize model errors with regard to the true targets by considering the assigned probability (output) o_j for the true class c_j of the target $\boldsymbol{y}$. The loss is high whenever the assigned probability o_j for the correct class c_j is low and, consequently, low whenever the assigned probability o_j is high. This is accomplished by using the negative logarithm of the probability o_j as loss

$$l(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) = -\log(o_j) \quad \text{where} \quad y_j = 1, \tag{2.5}$$

i.e., c_j denotes the target class. This loss is also often referred to as the negative log likelihood loss. The strength of the penalization is visualized in Fig. 2.9. If the model makes a grave error on the sample, i.e., assigns a low probability to the class which is the target class but requires a high probability, the cross entropy loss highly penalizes the model for this mistake. In fact, the loss function grows faster as the probability moves further towards zero and reveals the limiting property $\lim_{p \to 0} -\log(p) = \infty$. On the other hand, the loss approaches zero when the probability output of the true class is close to one since this is the desired output and in the case of being exactly one equal to the one-hot encoded target.

The further generalization of this loss by additionally considering the errors between the output and target probabilities for the other classes (o_i and y_i with $i = 1, \ldots, n$ where $i \neq j$ when c_j denotes the true class), leads to the full formulation of the loss as the cross entropy $H(\cdot, \cdot)$ between a sample $(\boldsymbol{x}, \boldsymbol{y})^\top$ and the (probability) output of the model $\boldsymbol{o} = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$ as follows

$$l(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) = H(\boldsymbol{y}, \boldsymbol{o}) = -\sum_{i=1}^{n} y_i \log(o_i) \tag{2.6}$$

where all terms within the sum evaluate to zero when *hard targets* are used since $y_i = 0$ with $i = 1, \ldots, n$ where $i \neq j$ when c_j denotes the target class and $y_j = 1$ what is, therefore, equivalent to Eq. 2.5.

Hard (target) in this context is equivalent to the designation of a one-hot encoded target so that all components y_i are zero except for the target class c_j where y_j is one. On the contrary, *soft targets* assign a non-zero probability to each class and aiming to assign the true class c_j the highest probability ($y_j > y_i$ for $\forall i = 1, \ldots, n$ where $j \neq i$) in order to capture the information of which class is most likely the true

class. Let $\boldsymbol{y}$ be soft targets in the cross entropy loss; then it obliges the probabilities to be high with an importance of the target probabilities. This is enforced through the penalization with the negative log probabilities weighted by the corresponding probabilities carried in the soft target. Soft targets can be particularly rich in information about the corresponding input $\boldsymbol{x}$ since they can encode the uncertainty of the true class (the class with the highest probability assigned to) depending on the magnitude of the probability of this class and, on the other hand, further information such as similarities between the classes can be expressed with soft targets. Those ideas and conceptions behind soft targets are of major importance for the method introduced in this thesis.

In fact, the output probabilities of the model $\boldsymbol{o}$ are nearly always soft targets due to the characteristics of the softmax function (Sec. 2.6).

From an information theoretical perspective, where the name cross entropy also derives from, one can derive the connection of the cross entropy to two other information theoretical metrics, the *entropy* $H(\cdot)$ and the *Kullback-Leibler (KL) divergence* (or called *relative entropy*) $D_{\mathrm{KL}}(\cdot \parallel \cdot)$, as follows

$$\begin{aligned} H(\boldsymbol{y}, \boldsymbol{o}) &= -\sum_{i=1}^{n} y_i \log(o_i) \\ &= -\sum_{i=1}^{n} y_i \left(\log(o_i) - \log(y_i) + \log(y_i)\right) \\ &= -\sum_{i=1}^{n} \left(y_i \log\left(\frac{o_i}{y_i}\right) + y_i \log(y_i) \right) = -\sum_{i=1}^{n} y_i \log\left(\frac{o_i}{y_i}\right) - \sum_{i=1}^{n} y_i \log(y_i) \\ &= D_{\mathrm{KL}}(\boldsymbol{y} \parallel \boldsymbol{o}) + H(\boldsymbol{y}). \end{aligned} \tag{2.7}$$

In Information Theory, the *entropy* $H(\cdot)$ of a distribution is a measure of the unpredictability of the possible values being adopted by a random variable obeying this distribution. This means that a distribution with a single (or only a few) narrow spikes and elsewhere low or near-zero values in the probability mass (or density) function has a lower entropy than a distribution with high or non-zero values everywhere in the probability mass (or density) function since the unpredictability of the outcome when sampling from the latter distribution is higher compared to the former one where it is most likely that such sample lies in the region of the spike in the probability mass (or density) function. For example, consider a discrete probability distribution over two classes representing whether the weather will be "sunny" and "rainy" tomorrow. If we are in a desert-like region, the probabilities assigned to the "sunny" class are meant to be relatively high and, thus, the unpredictability of tomorrow's weather is low which means that the entropy of this probability distribution is low. On the contrary, a probability distribution which assigns each weather class the same probability of occurrence, as it might be the case for being located in Germany in the month of April, would have a high entropy which represents the unpredictability of the next day's weather, i.e., a sampling outcome. The *KL divergence* $D_{\mathrm{KL}}(p \parallel q)$ defines an asymmetrical distance metric measuring the distance of a probability distribution q to another one p. The more the distribution q is similar to p, the lower is the KL divergence $D_{\mathrm{KL}}(p \parallel q)$, and vice versa. Regarding the weather example, assume that the cross entropy does now capture the true weather distribution as target and you or the weather forecast makes a prediction about the weather for tomorrow. If the prediction about the weather is congruent with the true distribution, the cross entropy will just be equal to the entropy of the weather distribution (i.e., its intrinsic unpredictability). But if your predictions are wrong, the cross-entropy will be higher than the entropy by the amount of the KL divergence (Géron, 2017, ch. 4).

This information theoretical understanding and additional definition of the cross entropy in Eq. 2.7 supports the theoretical work for this thesis. Furthermore, it encourages further investigation of the difference between applying either the cross entropy or the KL divergence. Since the data targets being used for training are not influenced by the model or the model's parameters θ, the entropy term $H(\boldsymbol{y})$ of the cross entropy loss (in Eq. 2.7) does not affect the computation of the gradient and

$$\nabla_{\boldsymbol{\theta}} H(\boldsymbol{y}, \boldsymbol{o}) \overset{(2.7)}{=} \nabla_{\boldsymbol{\theta}} D_{\mathrm{KL}}(\boldsymbol{y} \parallel \boldsymbol{o}) + \nabla_{\boldsymbol{\theta}} H(\boldsymbol{y}) = \nabla_{\boldsymbol{\theta}} D_{\mathrm{KL}}(\boldsymbol{y} \parallel \boldsymbol{o}) - \sum_{i=1}^{n} \underbrace{\nabla_{\boldsymbol{\theta}} y_i \log(y_i)}_{=0} = \nabla_{\boldsymbol{\theta}} D_{\mathrm{KL}}(\boldsymbol{y} \parallel \boldsymbol{o})$$

shows that the gradient with respect to the parameters $\boldsymbol{\theta}$ of the cross entropy loss $H(\boldsymbol{y}, \boldsymbol{o})$ is equal to the gradient of the KL divergence $H(\boldsymbol{y} \parallel \boldsymbol{o})$ for the model's output probabilities $\boldsymbol{o}$ and the data target $\boldsymbol{y}$. This holds for both hard as well as soft targets where the only difference lies in the quantities of the two metrics since for (one-hot encoded) hard targets the entropy is zero while it is greater than zero for soft target and shown in the following. Recap the definition of the possible values that hard target vectors can consist of $\boldsymbol{y} \in \{0,1\}^n$ for which the entropy $H(\boldsymbol{y}) = -\sum_{i=1}^{n} y_i \log(y_i)$ is always zeros since either $y_i = 0$ yields $y_i \log(y_i) = 0$ or $y_i = 1$ yields $y_i \log(y_i) = 0$ due to $\log(y_i) = \log(1) = 0$. Note that the argumentation is also valid in the opposite direction (that hard target probability distributions zero the entropy) since only either $y_i = 0$ or $y_i = 1$ zero the entropy. Thus, when soft targets are used, i.e., $\boldsymbol{y} \in (0,1)^n \neq \{0,1\}^n$, the entropy is $H(\boldsymbol{y}) \neq 0$ and, in fact, $H(\boldsymbol{y}) > 0$ since the entropy is non-negative per definition.

Nevertheless, the gradients are equal according to the above reasoning. Therefore, the cross entropy is preferable over the KL divergence since the evaluation is less computationally intensive (when computing the original definition in Eq. 2.6), more numerically stable and better compatible with the softmax squashing function applied to yield a probability distribution in the model's output $\boldsymbol{o}$.

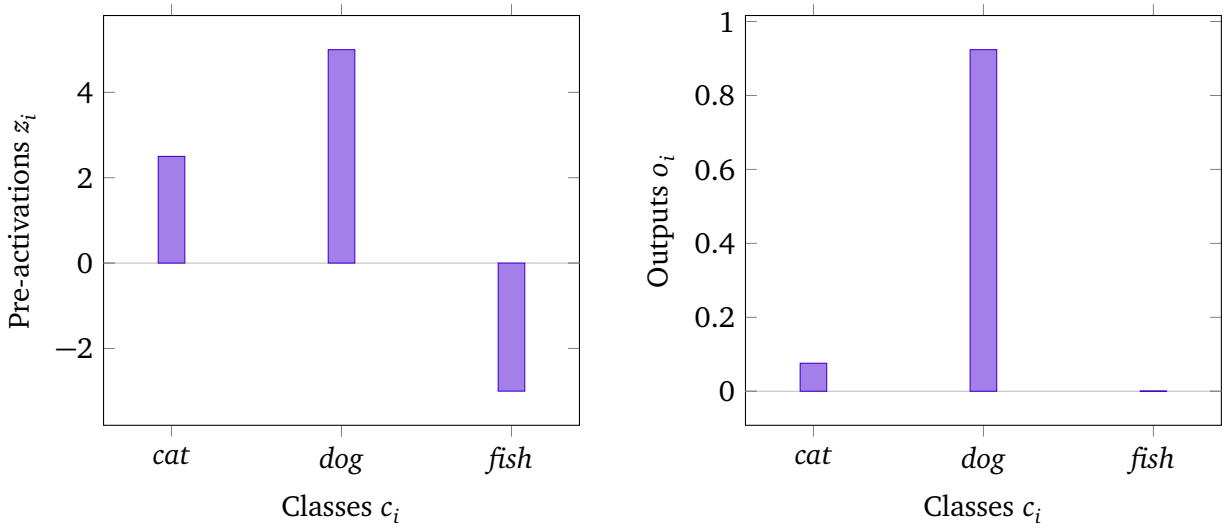
The exact partial derivatives (for the gradient) of the cross entropy with respect to the model's output $\boldsymbol{o}$ are

$$\begin{aligned}
\frac{\partial H(\boldsymbol{y}, \boldsymbol{o})}{\partial o_i} &= \frac{\partial}{\partial o_i} \left(-\sum_{k=1}^{n} y_k \log(o_k) \right) \\
&= -\sum_{k=1}^{n} \frac{\partial}{\partial o_i} y_k \log(o_k) = -\frac{\partial}{\partial o_i} y_i \log(o_i) - \sum_{\substack{k=1 \\ k \neq i}}^{n} \underbrace{\frac{\partial}{\partial o_i} y_k \log(o_k)}_{=0} \\
&= -\frac{y_i}{o_i}.
\end{aligned} \tag{2.8}$$

2.6 Softmax

The *softmax* function $\boldsymbol{\sigma}(\cdot)$, also known as the *normalized exponential function*, is a squashing function that ensures the output vector components forming proper probabilities/masses of a discrete probability distribution, i.e.,meeting the necessary conditions defined in Eq. 2.4. A probability is assigned to each softmax input component z_j by preserving the ranking of the input components such that

$$\forall \boldsymbol{z} \in \mathbb{R}^n : \quad \forall z_i, z_j : \quad z_i \geq z_j \Leftrightarrow \sigma_i(\boldsymbol{z}) \geq \sigma_j(\boldsymbol{z}) \tag{2.9}$$



(a) Pre-activations (no softmax)

(b) Outputs (softmax applied on pre-activations $o_i = \sigma_i(z_i)$)

Figure 2.10.: Illustrates the properties of the softmax function for **(a)** a pre-activation distribution over the three classes $\mathcal{C} = \{cat, dog, fish\}$ mapped to **(b)** the softmax probability distribution for some classification problem.

holds. Formally, the softmax function is defined as follows

$$\boldsymbol{\sigma} : \mathbb{R}^n \to (0,1)^n; \quad \boldsymbol{\sigma}(\boldsymbol{z}) = \begin{pmatrix} \sigma_1(\boldsymbol{z}) \\ \sigma_2(\boldsymbol{z}) \\ \vdots \\ \sigma_n(\boldsymbol{z}) \end{pmatrix} \quad \text{where} \quad \sigma_i(\boldsymbol{z}) = \frac{\exp(z_i)}{\sum_{k=1}^{n} \exp(z_k)}. \tag{2.10}$$

For the application of the cross entropy loss in solving classification problems (see Sec. 2.5.2) the output (vector) of the neural network $\boldsymbol{o}$ is meant to form a probability distribution, i.e., fulfill the necessary conditions in Eq. 2.4. For that reason, the softmax function is commonly used as activation function in the output layer of the neural network $\boldsymbol{o} = \boldsymbol{\varphi}_L(\boldsymbol{z}) = \boldsymbol{\sigma}(\boldsymbol{z})$ where $\boldsymbol{z}$ denotes the pre-activation (vector) in the output layer (the Lth layer). Especially the softmax as activation function is not only an element-wise application of a scalar-valued activation function on the layer as opposed to the other activation functions presented so far (see Sec. 2.4) but rather uses the pre-activations of all neurons within this layer for normalization purposes. Furthermore, the softmax function can be regarded as multi-class generalization or also called multinomial logistic regression ($n > 2$) of the sigmoid function (defined in Sec. 2.4.1) (Bishop, 2006, p. 198), which is utilized as activation function in the output neuron in a binary classification ($n = 2$) task (and also known as logistic regression).

Though, the question arises why the softmax function, in particular, is preferred over other techniques that manage to transform the pre-activation so that a discrete probability distribution over the classes is represented. To answer this question, the transformation of the softmax function should be further investigated. Fig. 2.10 exemplarily shows the mapping properties of the softmax function in two bar charts for an arbitrary pre-activation distribution over three classes $\mathcal{C} = \{cat, dog, fish\}$ for some classification problem. First of all, it is apparent that the ranking is preserved (as stated in Eq. 2.9) as well as the mapped distribution is a discrete probability distribution (meets the necessary conditions in Eq.

2.4). Moreover, this reveals the main feature of the softmax function, to which the function owes its name, that the maximum value in the pre-activations (i.e., class *dog* in Fig. 2.10a) is emphatically lifted in the output distribution in relative comparison with the other classes' probabilities (Fig. 2.10b). The probabilities for the other classes are strongly pushed towards zero (i.e., *cat* and *fish* in Fig. 2.10b). It can be seen as a form of "winner-take-all" (Goodfellow et al., 2016, ch. 6.2.2.3, p. 183).

Hence, the softmax produces a vector which is nearly similar to the one-hot encoded index of the maximum pre-activation $\operatorname{argmax}_{i=1,\dots,n} z_i$. It is called softmax since it delivers a softened one-hot encoding of the maximum input component. At the same time, it must be noted that this name is trifle misleading since it is a softened argmax and not a softened maximum which is why the term "softargmax" would be more appropriate (Goodfellow et al., 2016, ch. 6.2.2.3, p. 184) but rather unusual. In fact, as the components cannot exactly adopt the values of zero and one, the output can be construed as a soft target.

However, since the distribution is highly similar to the one-hot encoding representing the argmax, this comes in handy when learning from hard targets as it is the case for the targets in most classification problems (e.g., in combination with cross entropy loss) with the benefit of differentiability compared to the plain argmax function, which is a valuable property in order to bring gradient-based optimization (such as gradient descent) coupled with backpropagation to bear.

It is worthwhile mentioning that a popular misunderstanding of the probability distributions produced by a neural network with a softmax activation in the output layer is present where one interpret the probability values assigned to each class as quantification of certainty. For example falsely saying: "The neural network is 99% sure that this picture shows a cat". There is no evidence for making this interpretation since the neural network is trained on hard targets, on the one hand, where no further information of certainty is encoded and, on the other hand, the softmax function highly squashes the pre-activations so that the class with the highest pre-activation gets a probability value assigned of nearly 100% (nearer to 100% the higher this pre-activation is) without considering the (relative) ratio to the pre-activations of the other classes. For example, consider the pre-activations for the classes $\mathcal{C} = \{cat, dog, fish\}$ from Fig. 2.10a $\boldsymbol{z} = (2.5, 5, -3)^\top$ where the ratio between the pre-activations of *dog* and *cat* is one half and the softmax produces the probabilities of about $\boldsymbol{o} = (0.0758, 0.9239, 0.0003)^\top$. If these pre-activations were scaled by a factor of ten, $\boldsymbol{z} = (25, 50, -30)^\top$, maintaining the ratio of one half, *dog* would get assigned a probability of 100% (obviously depending on the floating point precision) and all other classes a probability of 0% showing evidently that the softmax does not keep the ratio between the pre-activations and is scale-variant. However, the probabilities of the other classes (besides the top class) can be indeed informative as they can carry similarities between the classes, for example. That information reveals the dark knowledge of the neural network and is further investigated for the method introduced in this thesis (see Sec. 2.8).

Apart from that, the softmax is indeed shift-invariant which means that pre-activation vectors with the same element-wise difference (i.e., shifted by a scalar) yielding the same softmax probability distribution. This can also be comprehended as that the (absolute) differences between the pre-activations are considered in the softmax mapping instead of the (relative) ratio. The shift-invariance is shown in the following

$$\sigma_i(\boldsymbol{z}-c) = \frac{\exp(z_i - c)}{\sum_{k=1}^{n} \exp(z_k - c)} = \frac{\exp(z_i)\exp(-c)}{\sum_{k=1}^{n} \exp(z_k)\exp(-c)} = \frac{\exp(z_i)\exp(-c)}{\exp(-c)\sum_{k=1}^{n} \exp(z_k)} = \frac{\exp(z_i)}{\sum_{k=1}^{n} \exp(z_k)} = \sigma_i(\boldsymbol{z}) \quad \forall c \in \mathbb{R}.$$

This property comes in handy to increase the numerical stability by subtracting the maximum pre-activation before applying the softmax whereby the values of the exponential terms in the softmax

definition (see Eq. 2.10) becoming smaller. This helps to prevent numerical issues like over- or underflows in the floating point arithmetic.

(Partial) Derivative:

Since the softmax function is further investigated for the topic of this thesis and gradient-based optimization is used for learning neural network, the partial derivatives of the softmax function are of high importance. They are given by

$$\frac{\partial \sigma_i(\boldsymbol{z})}{\partial z_j} = \sigma_i(\boldsymbol{z})\big(I_{ij} - \sigma_j(\boldsymbol{z})\big) \tag{2.11}$$

where I_{kj} are the elements of the $n \times n$ identity matrix $\boldsymbol{I}$ (Bishop, 2006, p. 209) and their derivation is shown in the following.

A case-by-case analysis [1] regarding j is done:

If $j \neq i$:

$$\begin{aligned}
\frac{\partial \sigma_i(\boldsymbol{z})}{\partial z_j} &= \frac{\partial}{\partial z_j}\left(\frac{\exp(z_i)}{\sum_{k=1}^{n}\exp(z_k)}\right) = \frac{\overbrace{\left(\frac{\partial}{\partial z_j}\exp(z_i)\right)}^{=0}\left(\sum_{k=1}^{n}\exp(z_k)\right) - \exp(z_i)\left(\frac{\partial}{\partial z_j}\sum_{k=1}^{n}\exp(z_k)\right)}{\left(\sum_{k=1}^{n}\exp(z_k)\right)^2} \qquad (2.12)\\
&= -\frac{\exp(z_i)\exp(z_j)}{\left(\sum_{k=1}^{n}\exp(z_k)\right)^2} = -\frac{\exp(z_i)}{\sum_{k=1}^{n}\exp(z_k)} \cdot \frac{\exp(z_j)}{\sum_{k=1}^{n}\exp(z_k)}\\
&= -\sigma_i(\boldsymbol{z}) \cdot \sigma_j(\boldsymbol{z}) \qquad (2.13)
\end{aligned}$$

If $j = i$:

$$\begin{aligned}
\frac{\partial \sigma_i(\boldsymbol{z})}{\partial z_j} &= \frac{\partial}{\partial z_j}\left(\frac{\exp(z_i)}{\sum_{k=1}^{n}\exp(z_k)}\right) = \frac{\left(\frac{\partial}{\partial z_j}\exp(z_i)\right)\left(\sum_{k=1}^{n}\exp(z_k)\right) - \exp(z_i)\left(\frac{\partial}{\partial z_j}\sum_{k=1}^{n}\exp(z_k)\right)}{\left(\sum_{k=1}^{n}\exp(z_k)\right)^2} \qquad (2.14)\\
&= \frac{\exp(z_i)\left(\sum_{k=1}^{n}\exp(z_k)\right) - \exp(z_i)\exp(z_j)}{\left(\sum_{k=1}^{n}\exp(z_k)\right)^2} = \frac{\exp(z_i)\left(\sum_{k=1}^{n}\exp(z_k)\right)}{\left(\sum_{k=1}^{n}\exp(z_k)\right)^2} - \frac{\exp(z_i)\exp(z_j)}{\left(\sum_{k=1}^{n}\exp(z_k)\right)^2}\\
&= \frac{\exp(z_i)}{\sum_{k=1}^{n}\exp(z_k)} - \frac{\exp(z_i)}{\sum_{k=1}^{n}\exp(z_k)} \cdot \frac{\exp(z_j)}{\sum_{k=1}^{n}\exp(z_k)}\\
&= \sigma_i(\boldsymbol{z}) - \sigma_i(\boldsymbol{z}) \cdot \sigma_j(\boldsymbol{z}) \qquad (2.15)
\end{aligned}$$

[1] Similar to Eli Bendersky - The Softmax function and its derivative: https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/ (as of April 11, 2019)

where the quotient rule applies for Eq. 2.12 and Eq. 2.14. In conclusion for arbitrary j, Eq. 2.13 and Eq. 2.15 compose to the final solution

$$\begin{aligned}\frac{\partial \sigma_i(\boldsymbol{z})}{\partial z_j} &= \begin{cases}\sigma_i(\boldsymbol{z}) - \sigma_i(\boldsymbol{z}) \cdot \sigma_j(\boldsymbol{z}), & \text{if } j = i\\ -\sigma_i(\boldsymbol{z}) \cdot \sigma_j(\boldsymbol{z}), & \text{otherwise}\end{cases}\\ &= \begin{cases}\sigma_i(\boldsymbol{z})\big(1 - \sigma_j(\boldsymbol{z})\big), & \text{if } j = i\\ \sigma_i(\boldsymbol{z})\big(0 - \sigma_j(\boldsymbol{z})\big), & \text{otherwise}\end{cases}\\ &= \sigma_i(\boldsymbol{z})\big(I_{ij} - \sigma_j(\boldsymbol{z})\big).\end{aligned}$$

2.6.1 Temperature Softmax

The *temperature (of the) softmax function* is a mechanism allowing for controlling the softness of the resulting distribution (or soft targets) $\boldsymbol{o}$ as exemplary visualized in Fig. 2.11 for the pre-activation distribution of Fig. 2.10a over three classes $\mathcal{C} = \{cat, dog, fish\}$ for some classification problem. It is mainly used in reinforcement learning (Sutton and Barto, 1998, ch. 2.3) to implicitly control the amount of exploration the agent performs during training by lifting the probability of taking actions that differ from the agent's greedy behavior. Applied on the output layer of an ANN, a higher temperature ($\tau > 1$) let the softmax produce softer outputs (Fig. 2.11c) and a lower temperature ($\tau < 1$) sharpens the distribution (Fig. 2.11a) by increasing the probability of the class which got the highest pre-activation value assigned to (and, consequently, decreasing the probabilities of the other classes). Intuitively speaking, the distribution is melted by a high temperature like candle wax and sharpened by a low temperature like freezing water expands and forms jags.

The temperature mechanism leads to a further generalized formulation of the softmax function $\boldsymbol{\sigma}(\cdot/\tau)$ and incorporates the hyperparameter of temperature $\tau \in (0, \infty) = \mathbb{R}^+$:

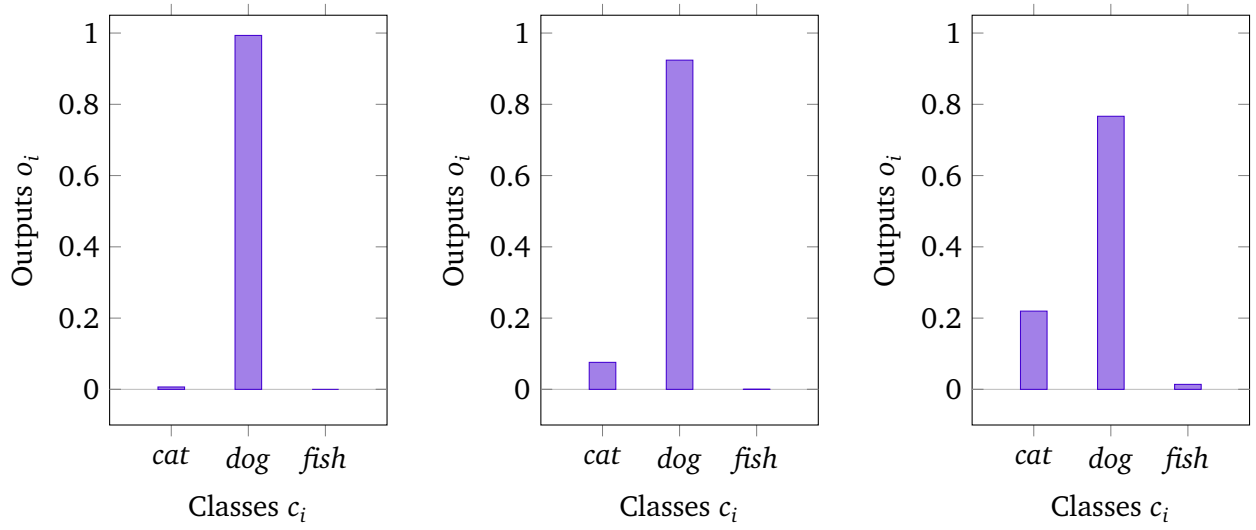
$$\sigma_i(\boldsymbol{z}/\tau) = \frac{\exp(z_i/\tau)}{\sum_{k=1}^{n} \exp(z_k/\tau)},$$

which, in other words, means that the pre-activation vector $\boldsymbol{z}$ is scaled by the inverse temperature $\tau^{-1} \cdot \boldsymbol{z} = \boldsymbol{z'}$ before it will be passed to the softmax function. Note, that $\tau = 1$ is the special case of this generalization leading to the standard softmax formulation as stated in Eq. 2.10.

(Partial) Derivative:

The partial derivatives of the temperature softmax function can be determined trivially based on the partial derivatives of the standard softmax function (see Eq. 2.11) by making use of the chain rule

$$\begin{aligned}\frac{\partial \sigma_i(\boldsymbol{z}/\tau)}{\partial z_j} &= \frac{\partial \sigma_i(\tau^{-1} \cdot \boldsymbol{z})}{\partial z_j}\\ &= \frac{\partial \sigma_i(\boldsymbol{z'})}{\partial z'_j}\frac{\partial z'_j}{\partial z_j} = \sigma_i(\boldsymbol{z'})\big(I_{ij} - \sigma_j(\boldsymbol{z'})\big)\,\tau^{-1}\\ &= \tau^{-1}\sigma_i(\tau^{-1}\boldsymbol{z})\big(I_{ij} - \sigma_j(\tau^{-1}\boldsymbol{z})\big) = \tau^{-1}\sigma_i(\boldsymbol{z}/\tau)\big(I_{ij} - \sigma_j(\boldsymbol{z}/\tau)\big).\end{aligned}$$



(a) Outputs $o_i = \sigma_i(\boldsymbol{z}/\tau)$ with $\tau = 0.5$

(b) Outputs $o_i = \sigma_i(\boldsymbol{z}/\tau)$ with $\tau = 1.0$; Equal to standard softmax $o_i = \sigma_i(\boldsymbol{z})$

(c) Outputs $o_i = \sigma_i(\boldsymbol{z})/\tau$ with $\tau = 2.0$

Figure 2.11.: Illustrates the characteristics of the temperature softmax function for the pre-activation distribution of Fig. 2.10a over the classes $\mathcal{C} = \{cat, dog, fish\}$. As a reference, the center chart **(b)** is the same as the standard softmax output (i.e., $\tau = 1.0$). For the left chart **(a)**, a low temperature of $\tau = 0.5$ is applied and sharpens the distribution whereas a high temperature of $\tau = 2.0$ applies for the right chart **(c)** softening the distribution.

Clearly, the obtained partial derivatives are equal to the ones of the standard softmax in Eq. 2.11 when scaling the pre-activations and the resulting partial derivatives by the inverse temperature τ^{-1}.

As further characteristics of the temperature hyperparameter τ, we examine its limits. If the temperature is set really low, $\tau \to 0$, the resulting distribution would have exactly the form of a one-hot encoding since the class with the highest pre-activation gets 100 % assigned. On the other hand, if the temperature is set extremely high, $\tau \to \infty$, a uniform distribution over the n classes with probability $1/n$ would be returned by the softmax function. Note that the latter case completely loses the information about the top-class and the ranking whereas the first case preserves this information but loses all information about the other classes.

The temperature softmax function is going to be of great importance regarding the exposure and amplification of *dark knowledge* as proposed in the method of *Knowledge Distillation (KD)* (Hinton et al., 2015) (see Sec. 2.8) as well as the method introduced in this thesis (see Sec. 3).

2.6.2 Combination of softmax and cross entropy loss

As frequently mentioned before, the softmax distribution and the cross entropy loss suit each other fruitfully and are almost always applied in combination in practice when it comes to classification problems. Since the method introduced in this thesis is implemented and applied in the loss function regarding these two functions, some properties are discussed here as they are being referred to later.

The partial derivatives (forming the gradient) with respect to the pre-activations (of the output layer in the neural network) $\boldsymbol{z}$ are

$$\frac{\partial H(\boldsymbol{y}, \boldsymbol{\sigma}(\boldsymbol{z}))}{\partial z_j} = \sigma_j(\boldsymbol{z}) - y_j = o_j - y_j \tag{2.16}$$

(Goodfellow et al., 2016, ch. 6.5.9, p. 218) and can be derived as follows

$$\begin{aligned}
\frac{\partial H(\boldsymbol{y}, \boldsymbol{\sigma}(\boldsymbol{z}))}{\partial z_j} &= \sum_{k=1}^{n} \frac{\partial H(y_k, \sigma_k(\boldsymbol{z}))}{\partial \sigma_k(\boldsymbol{z})} \frac{\partial \sigma_k(\boldsymbol{z})}{\partial z_j} \\
&= -\sum_{k=1}^{n} \frac{y_k}{\sigma_k(\boldsymbol{z})} \sigma_k(\boldsymbol{z})(I_{kj} - \sigma_j(\boldsymbol{z})) = -y_j(1-\sigma_j(\boldsymbol{z})) - \sum_{\substack{k=1 \\ k\neq j}}^{n} y_k(0-\sigma_j(\boldsymbol{z})) \\
&= -y_j + y_j\sigma_j(\boldsymbol{z}) + \sigma_j(\boldsymbol{z}) \sum_{\substack{k=1 \\ k\neq j}}^{n} y_k = -y_j + y_j\sigma_j(\boldsymbol{z}) + \sigma_j(\boldsymbol{z})(1-y_j) \qquad (2.17) \\
&= -y_j + y_j\sigma_j(\boldsymbol{z}) + \sigma_j(\boldsymbol{z}) - \sigma_j(\boldsymbol{z})y_j \\
&= \sigma_j(\boldsymbol{z}) - y_j = o_j - y_j.
\end{aligned}$$

where Eq. 2.17 is valid since $\sum_{k=1}^{n} y_k = 1$ due to y is representing a probability distribution (according to the first necessary condition in Eq. 2.4) and, therefore, $\sum_{k=1, k\neq j}^{n} y_k = \sum_{k=1}^{n} y_k - y_j = 1 - y_j$. The gradient term is intuitive as well as straightforward to interpret as the differences between the true and predicted probabilities per class pointing towards the direction where this difference increases.

2.7 Regularization

As described in Sec. 2.1.4, one major challenge in ML is to achieve the desired generalization of a model to "unseen" new data while learning on a limited and finite training data set. In other words, addressing the problem of overfitting (on the training data).

Methods and approaches that do so are referred to as *regularization*. More specifically, regularization denotes any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error (Goodfellow et al., 2016, ch. 5.2.2, p. 224). This goal can be accomplished through various approaches and techniques like constraining the parameters or adjusting the neural network's architecture.

Some well-established and prevalent regularization methods are presented in the following. These can be divided into two categories of regularization techniques: the parameter norm penalty regularization methods (e.g., L^1 and L^2 regularization) as well as the maxnorm regularization can be seen as techniques that constrain the weights of the neural network. Hence, the capacity of the model is limited to prevent overfitting since much specific information has to be encoded within the model to memorize particular training data samples or peculiarities. Thus, a general concept has to be learned in order to perform well on the training data. This is frequently justified with the statement that a simple model/description should be preferred over a complex one according to *Occam's razor*, which, however, is seen as a misinterpretation (Domingos, 1999) as the concept of simplicity has not directly to correlate with the syntactic size as it is expressed through the (number of parameters in a) model. Further, this is why weight constraining regularization methods are partly criticized and, e.g., replaced by more sophisticated regularization methods such as dropout. Dropout, for example, is a regularization method (explained in Sec. 2.7.3) from the second category of regularization techniques that are considered in this thesis, which can be phrased as encompassing modification techniques of the neural network's architecture while learning.

As the original intention of the method, Self-Imitation Regularization (SIR), introduced in this thesis (see Sec. 3) is the regularization of neural networks, it is investigated in comparison to such standard

regularization methods. Furthermore, combinations of SIR and standard regularization methods are further studied to understand the complementarity of the methods. Note that SIR does not directly fit into one of the two categories stated above.

Moreover, other techniques exist that are not directly involved in the learning (algorithm) but applied on the training data set and can therefore be referred to as regularization methods. Data augmentation serves an example for this type of regularization where the training data samples are randomly modified such as tilting or cropping samples of an image data set to prevent overfitting (Goodfellow et al., 2016, ch. 7.4; Mikołajczyk and Grochowski, 2018).

2.7.1 Parameter norm penalty regularization

This type of regularization adds a norm penalty $\Omega(\boldsymbol{\theta})$ to the loss function $l(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta})$ leading to a soft constraint regarding the parameter values, i.e., the magnitude of the parameters according to the chosen norm. The parameter norm penalty regularized loss is generally formulated as

$$\widehat{l}(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) = l(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) + \lambda\Omega(\boldsymbol{\theta})$$

where the non-negative hyperparameter $\lambda \in [0, \infty) \subset \mathbb{R}$ controls the strength of regularization where higher values of λ correspond to higher regularization and the special case of setting $\lambda = 0$ leading to no regularization (i.e., $\widehat{l}(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) = l(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta})$) (Goodfellow et al., 2016). Usually, the norm penalization does not apply for all parameters of the neural network but only the weights of the neurons (i.e., means that the biases remain unconstrained). Hence, $\boldsymbol{w}(\boldsymbol{\theta})$ denotes the vector containing all weights but no biases from the parameters (vector) $\boldsymbol{\theta}$ of the neural network.

With the choice of the norm, we can encode preferences for the weights having specific characteristics since, in general, different instances of weight vectors can achieve the same low (or even optimal) loss on the training data but different results regarding a proper generalization to unseen data. Two common norm choices and their characteristics are presented in the following.

L^1 regularization:

Choosing the L^1-*norm* as parameter norm penalty is called L^1(-norm) regularization or lasso regularization since the use of the L^1-norm as norm penalty is the core principle in lasso regression (Tibshirani, 1996). Formally, as follows

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}(\boldsymbol{\theta})\|_1 = \sum_k |w_k(\boldsymbol{\theta})|$$

results in the weight vector $\boldsymbol{w}(\boldsymbol{\theta})$ becoming sparse throughout the training since a sparse weight vector minimizes the penalty $\lambda\Omega(\boldsymbol{\theta})$ (Tibshirani, 1996), on the one hand, but still being able to solve the task reasonably, i.e., minimize the original loss function $l(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta})$. Sparsity means that most elements of a vector are zero (or almost zero due to numerical inaccuracies). The L^1-norm can be seen as measuring the sparsity by counting all violations in a vector where values differ from zero (weighted by the magnitude of the deviation).

The advantage of sparse weight vectors is that the neurons ignore most of the input (activations) and solely focus on the import ones whereby they will become nearly invariant to "noisy" inputs (Karpathy and Li, 2015, ch. 1: Neural Networks Part 2) which contributes to generalization. This can also be seen as a form of *feature selection* (i.e., using a small subset of features only) which commonly facilitates ML algorithms (Goodfellow et al., 2016, ch. 7 p. 232; Tibshirani, 1996).

L^2 regularization:

Regularizing with the L^2-*norm* penalty is defined as the (half of the) squared length of the weight vector $\boldsymbol{w}(\boldsymbol{\theta})$ in Euclidean space or, in other words, penalizing by the sum of quadratic weights. Formally, as follows

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{w}(\boldsymbol{\theta})\|_2^2 = \frac{1}{2} \left(\sqrt{\sum_k w_k(\boldsymbol{\theta})^2} \right)^2 = \frac{1}{2} \sum_k w_k(\boldsymbol{\theta})^2.$$

L^2(-norm) regularization is also known as ridge regularization as the L^2-norm penalty has its origin in ridge regression (Hoerl and Kennard, 1970; Tikhonov, 1943). Generally, the L^2-norm penalty is minimized by small values in the weight vector $\boldsymbol{w}(\boldsymbol{\theta})$ (not necessarily being sparse but rather diffuse) (Bishop, 2006, ch. 3.1.4, pp. 144-145). By penalizing peaky weight vectors and preferring diffuse ones, a neuron benefits from using all of its inputs a little rather than relying on some of its inputs a lot. Nevertheless, the noise sensitivity (against the inputs) decreases due to the small weight magnitudes with which such noise is only slightly amplified. This counteracts overfitting and consequently supports generalization.

L^2 regularization is also called *weight decay* since, in every update step, each weight is linearly decayed towards zero (Karpathy and Li, 2015, ch. 1: Neural Networks Part 2).

Note that squaring and halving the L^2-norm are applied due to computational reasons since this cancels the square root in the L^2-norm definition without changing the minimum due to the monotonicity of the square root function, and simplifying the gradient computation, respectively.

Usually, the regularization effects of the L^2-norm penalty are superior over the ones caused by the L^1-norm for what reason L^2 regularization is commonly preferred as weight penalty.

Elastic net regularization:

Nevertheless, the different aims of the L^1-norm and the L^2-norm penalty cannot necessarily be considered as entirely counteractive or, moreover, it might be meaningful to pursue both objectives to a certain extent and achieving a minimizing equilibrium between both penalties.

Therefore, both norm penalties can be used for regularization contemporaneously by adding both to the loss function $l(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta})$

$$l^{\text{ENR}}(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) = l(\boldsymbol{o}, \boldsymbol{y}; \boldsymbol{\theta}) + \lambda_1 \|\boldsymbol{w}(\boldsymbol{\theta})\|_1 + \lambda_2 \frac{1}{2} \|\boldsymbol{w}(\boldsymbol{\theta})\|_2^2$$

where the strength of each norm penalty is controlled individually by the two distinct hyperparameters λ_1 and λ_2, which have to be set carefully.

This approach is called *elastic net regularization* (Karpathy and Li, 2015, ch. 1: Neural Networks Part 2).

2.7.2 Maxnorm constraint

The *maxnorm* regularization method introduces a hard constraint in the form of an upper bound on the length of the weight vector of **each** neuron $\boldsymbol{w}^{(i)}(\boldsymbol{\theta})$ (where i indexes the ith neuron) by down-scaling the whole vector in order to fulfill the constraint $\left\|\boldsymbol{w}^{(i)}(\boldsymbol{\theta})\right\|_2 \leq c$ after each parameter update. It is called max-norm regularization since the maximum value of the norm is bounded to the value of the hyperparameter c. Furthermore, c can be seen as the radius of a ball in the weight space of a neuron wherein all weight vectors lie that fulfill the constraint. Whenever a weight vector $\boldsymbol{w}^{(i)}(\boldsymbol{\theta})$ went out of

this ball through an update step, it is projected onto the surface of this ball (Srivastava et al., 2014) by down-scaling the vector in order to fulfill the constraint.

2.7.3 Dropout

Concept and implementation:

Srivastava et al. (2014) introduced the method of *dropout* as a simple way to prevent neural networks from overfitting. Dropout is applied on a higher level than only (soft or hard) constraining the parameters of the neural network as it is the case for the other regularization methods presented above but randomly modifying the architecture of the neural network over the course of training and, consequently, altering the flow of information through the neural network. Dropout implements a simple form of randomly changing the architecture by temporarily "dropping" a random subset of neurons for a single training iteration (including forward- as well as backward pass). By dropping a neuron, it is referred to temporarily removing the neuron from the neural network along with all its incoming and outgoing connections (Srivastava et al., 2014). From another point of view, this can be seen as sampling a (sub-)network or binary masked network from the 2^N possible neural network instances (when N denotes the number of neurons within the neural network) or setting the output activation of a dropped neuron to zero no matter what input activations are perceived. The hyperparameter $p \in [0,1] \subset \mathbb{R}$ specifies the probability of a neuron being dropped. Be aware that the parameters of a dropped neuron are being retained and are not affected by the parameter update in this training iteration and that no dropout is applied at prediction/test time after training is done.

At test time, the output activations of all neurons o_i have to be scaled by $(1-p)$ in order to maintain the expected output activation of a neuron from training with dropout (and hence the desired and learned prediction behavior of the neural network) (Karpathy and Li, 2015, ch. 1: Neural Networks Part 2) since this expected output activation with dropout applied is given by

$$\mathop{\mathbb{E}}_{m_i \sim \text{Bernoulli}(1-p)} \{m_i o_i\} = (1-p) \cdot 1 \cdot o_i + p \cdot 0 \cdot o_i = (1-p) o_i$$

where the random variable m_i denotes the dropout mask for the ith neuron, which is Bernoulli-distributed and takes the value zero at a probability of p causes the ith neuron being dropped in this iteration and takes the value one at a probability of $1-p$ prevent the ith neuron from dropping. This is an efficient approximation of the predictions of all 2^N possible neural networks, which had been trained with dropout and should be therefore queried as an ensemble at test time. However, Srivastava et al. (2014) showed that averaging the predictions of $k \geq 1$ (Monte Carlo model averaging) neural networks using dropout each test case can indeed slightly reduce the prediction error for high values of k but is not worth the computational overhead. Thus, the scaling approach is a fairly good approximation of the average of the true ensemble (i.e., $k \to \infty$).

Moreover, since changing the behavior of the neural network at prediction/test time is commonly undesired, *inverted dropout* is applied in practice by remaining the output activations of the neurons untouched at prediction/test time but scaling these by the inverse of $1-p$ (i.e., dividing by $1-p$) at training time (Karpathy and Li, 2015, ch. 1: Neural Networks Part 2). This is another way to ensure

that the expected output activation of a neuron is the same during test and training time (viz. o_i in both scenarios) as shown in the following

$$\mathop{\mathbb{E}}_{m_i \sim \text{Bernoulli}(1-p)} \left\{ \frac{m_i o_i}{1-p} \right\} = \frac{(1-p) \cdot 1 \cdot o_i}{1-p} + \frac{p \cdot 0 \cdot o_i}{1-p} = o_i.$$

It is important to mention that the definition of the hyperparameter p is not consistent throughout the literature since it often denotes the probability of turning a neuron on instead of dropping it.

Interpretation and features:

The most widespread interpretation of the regularizing effect of dropout is that this implicitly trains an ensemble of models. Learning an ensemble of several models is a common technique in ML to depress the generalization error with the idea that an incorrect prediction of a single model can be compensated or outvoted by other models. In fact, each of the 2^N dropout binary masks represents one model in the ensemble, but these models share their parameters as opposed to a classical ensemble of models with independently optimized parameters (Hinton et al., 2015).

On the other hand, dropout can be interpreted that the neural network is being forced to have a redundant representation of knowledge acquired through learning since certain knowledge about specific classes or inputs is not necessarily available at some iteration if the neurons, which are dedicated to encoding this knowledge, are currently dropped (Canny, 2016, Lecture 8, p. 47). Moreover, this makes it quite difficult and unlikely that the neural network is being able to overfit on specific training samples as training particular neurons to trigger on such samples is not really achievable when these neurons are likely being dropped in various iterations. Even if they are turned on, the input activations would probably be too different due to the (mostly) different dropout situations in the upstream neurons in the neural network over various iteration.

Srivastava et al. (2014) showed that dropout provokes that the hidden features (i.e., the output activations of the hidden layers) becoming sparse. Regarding proper generalization, this might be preferable for the same reasons that apply to L^1 regularization.

Apart from that, Srivastava et al. (2014) pointed out that the breaking of co-adaptions by dropout might be the main factor for a low generalization error. The partial derivative for each parameter in the neural network determines how the parameter should be changed so that the loss on the mini-batch is reduced but depending on the current behavior of all other neurons or parameters. Consequently, the parameters of a neuron might be changed by the gradient to fix mistakes induced by other neurons what Srivastava et al. (2014) refer to as complex co-adaptions leading to overfitting since these co-adapted parameters do not generalize to unseen data. Therefore, dropout might inhibit co-adaption by preventing a malfunctioning neuron from relying on the presence of other specific units correcting its mistakes.

2.8 Knowledge Distillation and Dark Knowledge

The work of *Knowledge Distillation (KD)* (Hinton et al., 2015) is of a central role in this thesis and introduces the concept of *dark knowledge* (Hinton, 2014). While SIR, the method introduced in this thesis, uses the KD mechanism as a basis for regularizing neural networks, the original motivation behind KD comes from a different purpose:

A pivotal impediment to the deployment of large deep neural networks (or even ensembles of such) in the field such as industrial real-time applications on potentially low-power computing devices (e.g., mobile devices) is that these suffer from requiring higher computing power to meet such demands.

Additionally, decreasing the computing requirements when deploying deep neural networks via cloud services in data centers, is of a keen economic and ecological interest. By speaking of deployment, it is implied that the neural networks were trained beforehand and are then put to use as they are, i.e., without further learning, for which reason only forward passes within the neural network are performed. Learning a smaller neural network from the very start is most often barely an adequate option since this will most likely be outperformed by a larger neural network (ensemble) in terms of the capability in recognizing and extracting patterns, structures, and hidden properties from the data to achieve superb learning success. This obviously comes with the drawback of great computation power and time for making predictions, i.e., performing forward passes through the neural network. Changing the structure/architecture of the neural network while perceiving the learned knowledge is hard or rather impossible since this knowledge is encoded in the parameter values and, thus, closely intertwined with the structure of the neural network. Hence, rather a technique of transferring or projecting the learned knowledge from a large neural network to a smaller one is needed. Hinton et al. (2015) continued and further developed the work of Bucila et al. (2006); Ba and Caruana (2014) and realized this concept with the method of KD by distilling the knowledge in large neural networks (and even ensembles of neural networks) into smaller (single) neural networks. In fact, the method that was proposed in Bucila et al. (2006) turns out to be only a special case of KD as shown by Hinton et al. (2015).

However, Urban et al. (2017) recently showed in empirical studies that CNNs require a certain depth in order to learn well even if distillation is used to support besides learning directly on the data.

Dark knowledge:

To understand the knowledge distillation technique, it is crucial to understand the concept behind dark knowledge first. As stated in the KD introduction, small and shallow neural networks tend to be less able to learn from the (hard) data targets and to gain information compared to large and deep neural networks. The hard data targets (one-hot encodings) simply tell the correct class that should be predicted. Therefore, a large deep neural network must have acquired more knowledge than a small shallow one that supports predicting the correct class and is not obviously represented in the hard data targets, as well as not easily detectable in the input data. This supporting knowledge is referred to as *dark knowledge* (Hinton, 2014) and buried in the neural network.

Hinton (2014), as well as Hinton et al. (2015), propose that the dark knowledge is revealed in the soft prediction of a neural network since this can be interpreted as similarities between the classes the neural network has learned (given an input sample). The similarities are expressed through the probabilities assigned to the classes that are not predicted (i.e., all classes except the one with the highest probability) so that the classes with higher probabilities are relatively more similar to the predicted class than the other ones with lower probabilities. From another perspective, one can say that the neural network is more sure about the given input should have been rather predicted with some class holding a higher probability than another one with a lower probability. In another way, the probabilities can be used as scores in order to form a ranking of the classes, analogous to preference learning (see Sec. 2.1.1), where the relation $c_i \succ c_j$ ($\Leftrightarrow y_i > y_j$) allows the interpretation that the neural network considers it more likely that the correct class for the given input is c_i instead of c_j. Or for $c_i \succ c_j \succ c_k$ that class c_j is more similar to c_i than c_k is, for the given input.

As commonly all other probabilities (except the highest probability) of a softmax output are rather low, Hinton et al. (2015) used the temperature softmax (with temperature $\tau > 1$) to amplify the probabilities of the classes that are most similar to the predicted class and consequently uncovering the dark knowledge more strongly. As an example, refer to Fig. 2.11 where one can see that the outputs of the neural network with a standard softmax activation (Fig. 2.11b) show the expected similarities that *cat* is (more)

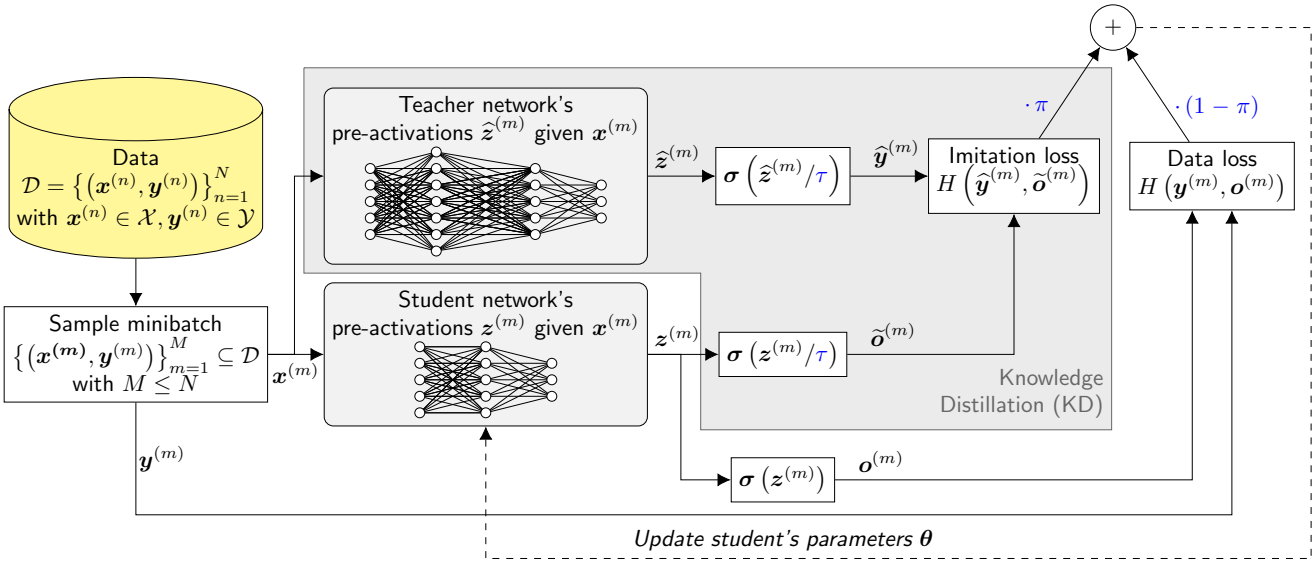


Figure 2.12.: Knowledge Distillation (KD) framework diagram. A full (student) neural network training setting is framed where the additional KD technique is highlighted in gray and KD hyperparameters are printed in blue.

similar to *dog* than *fish* is (more) similar to *dog*. Alternatively, the neural network would rather predict the class *cat* for this input than the class *fish*. The higher temperature ($\tau = 2.0$ in Fig.2.11c) emphasizes this relation by noticeable increasing the probability of the class *cat* while keeping the probability of the class *fish* comparatively low.

In this thesis, further techniques to reveal and emphasize the dark knowledge are proposed (in Sec. 3).

Distillation technique:

As a large deep neural network seems to occupy superior dark knowledge than a small shallow one the goal of KD is now to distill this dark knowledge in the large deep neural network, called the *teacher (network)* $\widehat{\boldsymbol{y}} = \widehat{\boldsymbol{f}}(\boldsymbol{x}; \widehat{\boldsymbol{\theta}})$ for this purpose, to the smaller neural network, accordingly called the *student (network)* $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$. See Fig. 2.12 that visualizes the entire KD framework as an abstract diagram.

To reveal the dark knowledge in the teacher network, all training input data samples are processed by the teacher network to obtain corresponding soft targets containing the dark knowledge. As these soft targets are produced with a (temperature) softmax activation, the necessary conditions for a discrete probability distribution (see Eq. 2.4) are fulfilled, and these soft targets can be used as targets in the cross entropy loss $H(\cdot, \cdot)$. Hence, the student network can be trained on the training data but using the teacher's soft targets instead of the data targets which are much more informative (carry the dark knowledge) in the cross entropy loss. Hinton et al. (2015) claimed that this can be improved by additionally

incorporating the data targets, i.e., the loss to the data targets, and using a weighted average of both loss functions as the final loss

$$
\begin{aligned}
L^{\mathrm{KD}}(\boldsymbol{Z}, \boldsymbol{Y}, \widehat{\boldsymbol{Y}}; \boldsymbol{\theta}) &= \frac{1}{M} \sum_{m=1}^{M} l^{\mathrm{KD}}(\boldsymbol{z}^{(m)}, \boldsymbol{y}^{(m)}, \widehat{\boldsymbol{y}}^{(m)}; \boldsymbol{\theta}) \\
&= \frac{1}{M} \sum_{m=1}^{M} \left(\pi \cdot \underbrace{H\left(\widehat{\boldsymbol{y}}^{(m)}, \widetilde{\boldsymbol{\sigma}}(\boldsymbol{z}^{(m)})\right)}_{(2.18)} \quad + \quad (1-\pi) \cdot \underbrace{H\left(\boldsymbol{y}^{(m)}, \boldsymbol{\sigma}(\boldsymbol{z}^{(m)})\right)}_{(2.19)} \right)
\end{aligned}
\tag{2.20}
$$

where the hyperparameter $\pi \in [0,1] \subset \mathbb{R}$ denotes the (relative) weight for the loss of Eq. 2.18 contribution to the final KD loss in relation to the data loss (Eq. 2.19), $(\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})$ is the mth sample of the data (mini-)batch, M denotes the (mini-)batch size, $\boldsymbol{z}^{(m)}$ the pre-activations of the student network given the mth data sample input $\boldsymbol{x}^{(m)}$, $\widehat{\boldsymbol{y}}^{(m)} = \widehat{\boldsymbol{f}}(\boldsymbol{x}^{(m)}; \widehat{\boldsymbol{\theta}})$ the (mth) soft target predicted by the teacher network given the mth data sample input $\boldsymbol{x}^{(m)}$, $\boldsymbol{Z}$ the (matrix of) pre-activations, $\widehat{\boldsymbol{Y}}$ the (matrix of) soft targets, and $\boldsymbol{Y}$ the (matrix of) one-hot encoded data targets.

It should be mentioned that another interpretation of distilling (or transferring) the knowledge to the student network exist. Namely, that the student network mimics or imitates the teacher network (Ba and Caruana, 2014; Urban et al., 2017). That is analogous to saying that a neural network fits or approximates the training data originated by reducing the loss to this data. The imitation rate π represents the (relative) degree of imitating the teacher in contrast to fitting the data. Note, that the terminology of imitation is eponymous for the introduced method, Self-Imitation Regularization (SIR), in this thesis.

Note that both the teacher's prediction and the student's softmax output used in the imitation loss (Eq. 2.18) should have used the same temperature τ for the softmax activation. The use of a high temperature leads to softer distributions having a higher entropy and providing more information per training case (since the higher magnitudes of the probabilities of the non-predicted classes affect the gradient more intensely) and can improve the learning speed by reducing the variance of the gradients between update steps (Hinton et al., 2015).

3 Self-Imitation Regularization (SIR)

Self-Imitation Regularization (SIR) is a regularization method implementing a form of self-imitation in a neural network and, therefore, leverages its interim (dark) knowledge by explicitly incorporating this knowledge in the loss formulation. It can be understood as being based on KD and is inspired by the work of Hu et al. (2016) where a method was proposed to transfer prior knowledge, such as domain knowledge or human intention, to a neural network through first-order logic rules. Here, SIR, as well as Hu et al. (2016), follow the idea of not using an additional teacher network beside the training of the student network but only has the (student) neural network whose predictions are used as a basis to form (teacher) targets in a KD loss (i.e., as $\widehat{\boldsymbol{y}}^{(m)}$ in Eq. 2.20). In strong contrast to Hu et al. (2016), no further knowledge, like logic rules encoding, is used to obtain the (teacher) targets from reshaping the (student) neural network's predictions. Therefore, SIR can be easily applied in any neural network classification learning setting since no more than the data is needed and the imitation (loss) targets are based on the (dark) knowledge that the neural network has previously acquired up to the current training iteration. On the other hand, towards KD, the non-negligible overhead – in terms of computation and (hyperparameter) tuning time, as well as computational resources – of training an exhaustive teacher network drops since only one network, the student, is being trained. Therefore, the computational overhead induced by SIR is very small since only information is used that is already required for the computation of the data loss (i.e., the per-activations of the neural network). Incorporating the neural network's own (soft) output in the imitation loss (Eq. 2.18) again and further training on this should make the naming *self-imitation* clear. That it is not necessary to use a teacher network of a higher capacity or expressive power in a KD-like setting in order to contribute generalization but a neural network of the same architecture as the student network is shown by Furlanello et al. (2018) and Yim et al. (2017). The components of SIR are explained in detail below, but it is worth already referring to the framework diagram in Fig. 3.1.

However, another technique to reshape the predictions is needed since using the exact neural network's softmax output $\boldsymbol{\sigma}(\boldsymbol{z})$ as imitation target (i.e., $\widehat{\boldsymbol{y}} = \boldsymbol{\sigma}(\boldsymbol{z})$; note that the mini-batch index m is omitted for the sake of clarity) in the cross entropy term again (Eq. 2.18) can indeed lead to a non-zero (cross entropy) imitation loss, but the gradient of this imitation loss with respect to $\boldsymbol{z}$ is always zero and, therefore, will not affect the learning. The former can be seen in the fact that the cross entropy simplifies to the entropy, which is non-zero for soft targets (i.e., if probabilities besides 0% or 100% exist). This simplification is derived in the following

$$H(\boldsymbol{\sigma}(\boldsymbol{z}), \boldsymbol{\sigma}(\boldsymbol{z})) = -\sum_i \sigma_i(\boldsymbol{z}) \log(\sigma_i(\boldsymbol{z})) = H(\boldsymbol{\sigma}(\boldsymbol{z}))$$

or can also be justified by the fact that the cross entropy is composed of the sum of the entropy and the KL divergence (Eq. 2.7) and the KL divergence, forming a distance metric for the deviation of two probability distributions, is obviously zero in that case. The fact that the gradient with respect to the pre-activations $\boldsymbol{z}$ (over which the gradients flow back to the parameters $\boldsymbol{\theta}$ in the neural network) is zero

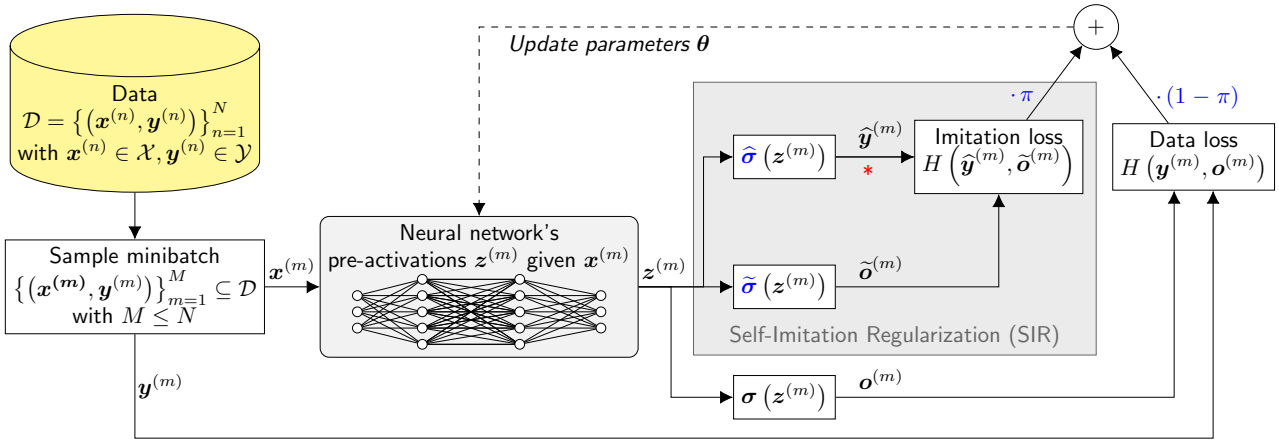


Figure 3.1.: Self-Imitation Regularization (SIR) framework diagram. A full neural network training setting is framed where the additional method of SIR is highlighted in gray and SIR hyperparameters are printed in blue. The red asterisk * refers to the path where the backward flow through the imitation target can be considered.

can be seen by putting $\widehat{\boldsymbol{y}} = \boldsymbol{\sigma}(\boldsymbol{z})$ in the partial derivatives of the softmax and cross entropy combination (defined in Eq. 2.16)

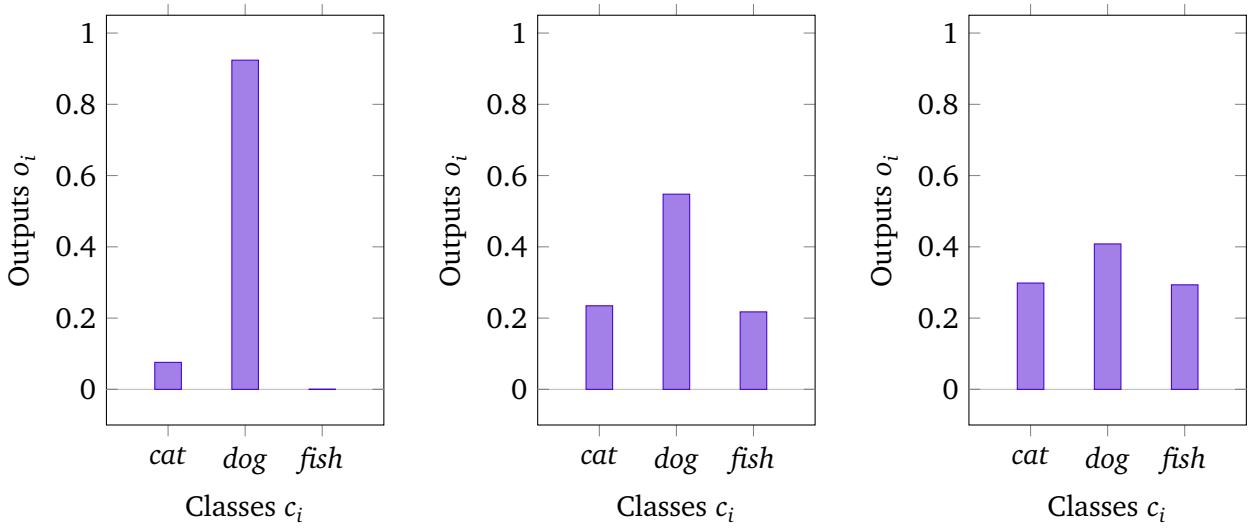
$$\frac{\partial H(\boldsymbol{\sigma}(\boldsymbol{z}), \boldsymbol{\sigma}(\boldsymbol{z}))}{\partial z_j} = \sigma_j(\boldsymbol{z}) - \sigma_j(\boldsymbol{z}) = 0.$$

Asymmetric softmax imitation:

Hinton et al. (2015) suggested using the temperature softmax function to reveal the dark knowledge and improve the distillation where such softening can be seen as a modification of a standard softmax output, and can be used to form the (teacher) soft target from the neural network's prediction (i.e., the pre-activations $\boldsymbol{z}$). Obviously, if the same temperature τ is used for both the neural network's output $\widetilde{\boldsymbol{o}} = \widetilde{\boldsymbol{\sigma}}(\boldsymbol{z})$ and the soft target $\widehat{\boldsymbol{y}}$ in the imitation loss as it is the case in the KD setting (Eq. 2.18), the problem of zero gradients, mentioned before, would come up again. For that reason, different temperatures have to be used either for the neural network's output $\widetilde{\boldsymbol{o}} = \widetilde{\boldsymbol{\sigma}}(\boldsymbol{z})$ and the soft target $\widehat{\boldsymbol{y}} = \widehat{\boldsymbol{\sigma}}(\boldsymbol{z})$, which are denoted as $\widetilde{\tau}$ and $\widehat{\tau}$ respectively, with $\widetilde{\tau} \neq \widehat{\tau}$. Normally, we make sure that the teacher target is softer than the student one so that the student is able to profit from soft(er) targets (i.e., $\widetilde{\tau} < \widehat{\tau}$.). We call this *asymmetric softmax imitation*. In order to verify this, we ran preliminary experiments in the standard KD setting with asymmetric softmax imitation, and it turned out that the imitation/distillation of/from a pre-trained teacher network is still possible. The achieved performance is comparable to those achieved in the KD experiments where the same temperature was used (i.e., symmetric softmax imitation with $\tau = \widetilde{\tau} = \widehat{\tau}$). For more details on these experiments, refer to Sec. 3.3.2.

Chained softmax function:

Another approach to soften the softmax distribution, which, to the best of our knowledge, has not been studied or used before, is to chain the softmax function a few times. We refer to this as the *chained*



(a) Standard softmax outputs $o_i = \sigma_i(\boldsymbol{z})$

(b) Outputs $o_i = \sigma_i(\sigma(\boldsymbol{z}))$

(c) Outputs $o_i = \sigma_i(\sigma(\sigma(\boldsymbol{z})))$

Figure 3.2.: Illustrates the characteristics of the ζ-chain softmax function for the pre-activation distribution of Fig. 2.10a over the classes $\mathcal{C} = \{cat, dog, fish\}$. As a reference, the left bar chart **(a)** is the same as the standard softmax output (i.e., $\zeta = 1$). The distribution is softened **(b)** with $\zeta = 2$ and **(c)** with $\zeta = 3$ for the chained softmax function $\boldsymbol{\sigma}^{(\zeta)}(\boldsymbol{z})$.

softmax function or, more precisely, as the ζ*-chain softmax function* where ζ denotes the number of the chained softmax functions what is formally defined as

$$\boldsymbol{\sigma}^{(\zeta)}(\boldsymbol{z}) = \underbrace{(\boldsymbol{\sigma} \circ \ldots \circ \boldsymbol{\sigma})}_{\zeta \text{times}}(\boldsymbol{z}) \quad \text{with} \quad \zeta \in \mathbb{N}^+.$$

The softening characteristics of the chained softmax are similar to the temperature softmax since for both the distribution gets softer when the ζ respectively τ is increased. Note that the probabilities of the non-predicted classes are strongly increased, even if the probability after applying the softmax function once is nearly zero. In Fig. 3.2 this can be exemplarily seen for the 2-chain (Fig. 3.2b) and 3-chain (Fig. 3.2c) softmax function compared to the standard softmax (Fig. 3.2a). The pre-activations that serve as inputs for the softmax functions are the same as shown in Fig. 2.10a for the three classes *cat*, *dog*, and *fish*. Of course the same applies to the chained softmax function that the concept behind asymmetric softmax imitation must be obeyed in order to obtain non-zero gradients. So the distinction regarding the hyperparameter ζ needs to be made that $\widetilde{\zeta}$ specifies the number of times the softmax is applied on the neural network's pre-activations $\boldsymbol{z}$ in order to form the imitation prediction/output $\widetilde{\boldsymbol{o}}$ and $\widehat{\zeta}$ the number of times the softmax is applied on $\boldsymbol{z}$ to obtain the soft imitation targets $\widehat{\boldsymbol{y}}$ with $\widetilde{\zeta} \neq \widehat{\zeta}$.

Again, it is demonstrated in Sec. 3.3.1 that the chained softmax function can be seen as a replacement for the temperature softmax function in KD to achieve comparatively accurate performance according to preliminary experiments with KD.

Loss formulation:

For the sake of clarity, the (multi-objective) loss function incorporating the method of SIR is stated here

$$
\begin{aligned}
L^{\mathrm{SIR}}(\boldsymbol{Z}, \boldsymbol{Y}; \boldsymbol{\theta}) &= \frac{1}{M} \sum_{m=1}^{M} l^{\mathrm{SIR}}(\boldsymbol{z}^{(m)}, \boldsymbol{y}^{(m)}; \boldsymbol{\theta}) \\
&= \frac{1}{M} \sum_{m=1}^{M} \left(\pi \cdot \underbrace{H\left(\widehat{\boldsymbol{\sigma}}(\boldsymbol{z}^{(m)}), \widetilde{\boldsymbol{\sigma}}(\boldsymbol{z}^{(m)})\right)}_{(3.1)} \quad + \quad (1-\pi) \cdot \underbrace{H\left(\boldsymbol{y}^{(m)}, \boldsymbol{\sigma}(\boldsymbol{z}^{(m)})\right)}_{(3.2)} \right)
\end{aligned} \tag{3.3}
$$

where the definitions remain the same as in the KD loss formulation (Eq. 2.20) except the distinct softmax function variants $\widehat{\boldsymbol{\sigma}}(\cdot)$ and $\widetilde{\boldsymbol{\sigma}}(\cdot)$ in Eq. 3.1 producing the imitation targets $\widehat{\boldsymbol{y}}$ and the imitation predictions $\widetilde{\boldsymbol{o}}$, respectively. Hence, these two can be either set individually as any chained or temperature softmax function so that the concept of asymmetric softmax imitation can be represented with this notation.

Gradient flow through the imitation targets:

Since the imitation targets are produced by the neural network itself, which is trained simultaneously on this loss, the target values depend on the neural network's parameters $\boldsymbol{\theta}$. This is in contrast to the loss computation based on the training data (Eq. 3.2) where the data targets obviously do not depend on $\boldsymbol{\theta}$ or in the KD setting where the teacher network that was trained beforehand is kept fixed during distillation. Now, it can be considered to incorporate the gradients flowing back through the imitation targets $\widehat{\boldsymbol{y}}$ with respect to the parameters $\boldsymbol{\theta}$ instead of only considering the gradients induced by the neural network's prediction $\widetilde{\boldsymbol{o}}$ within the imitation loss. To make this more clear, consider the schematic diagram of the SIR framework in Fig. 3.1 as an abstract computational graph, where the path marked with the red asterisk (*) indicates the path over which the gradients can be propagated back in SIR. Intuitively, this gradient explains how the "self-teaching" behavior of the neural network (represented by $\widehat{\boldsymbol{y}}$ in the imitation loss) should be changed in order to reduce the loss.

Furthermore, by incorporating the target gradients, this also enables to use the symmetric softmax imitation for SIR since a non-zero gradient is obtained even though the imitation target $\widehat{\boldsymbol{y}}$ and the prediction $\widetilde{\boldsymbol{o}}$ are being the same. As mentioned above, symmetric SIR can be construed as an entropy penalty, which means that this penalty is minimized by harder prediction distributions. Further, the gradient of the imitation loss in symmetric SIR is exactly the gradient of the entropy function. Note that in the rest of this thesis, symmetric SIR does implicitly refer to the symmetric SIR variant where the target gradients are incorporated because otherwise no gradients exist, which would make any experimental examination obsolete. On the other hand, asymmetric SIR produces non-zero gradients for both variants – with or without target gradients – which is why the chosen variant is explicitly mentioned.

3.1 Hypotheses

Before presenting the experimental set-up (Sec. 4) and results (Sec. 5), all (initial) hypotheses about SIR or, more specifically, about the regularization behavior of SIR are outlined here first since they have given a strong impetus to the investigations and the design of the experiments. Note that the conclusion (Sec. 6) summarizes whether the hypotheses can be rejected or accepted.

Hypothesis I (Regularization)**.** Producing soft targets instead of using the raw hard data targets only can be seen as a qualitative enrichment of targets regarding their information content. We make the hypothesis that even soft targets produced by the network itself – for which reason these might be of poor quality at times – can already contribute generalization (i.e., regularize) even if the quality is lower than for targets obtained by a sophisticated teacher network that was trained beforehand in KD. That soft targets produced by a proper teacher network do have regularizing properties per se emerges from various experiments in Ba and Caruana (2014); Hinton et al. (2015); Urban et al. (2017) where they ascertained that the (student) neural networks were able to learn better in terms of the accuracy on unseen (test) data when soft targets were provided compared to learning on the training data only.

Hypothesis II (Compensate for erroneous data labels)**.** Generally, data labels are assumed to be correct, but in practice, this is not always the case. For example, if the data (including the labels) was collected via sensor measurements, noise signals might have occurred; or if the data set was labeled manually, careless mistakes or, even worse, frequent mistakes due to an advanced difficulty in labeling the data set might have been made. Therefore, compensating such erroneous data labels while training can be a highly desirable property for a regularization method such as SIR.

(a) By directly incorporating the neural network's predictions on the input samples into the loss computation, we hypothesize that the possibility of declining a data target arises whenever the neural network's prediction indicates another class and, therefore, assesses the data label as wrong. This could be made possible by the SIR loss (Eq. 3.3) constituting a multi-objective loss. A minimum specifies a trade-off between satisfying the two objectives of staying with the neural network's opinion (possibly represented by the self-imitation loss, Eq. 3.1) and learning something new from the data (represented by the data loss, Eq. 3.2). The relative importance of the two objectives can be set by the hyperparameter π. More intuitively speaking, with SIR, the neural network might be able to question the training data instead of unthinkingly learning from the data targets even if they evidently seem to be wrong. This is exactly the case in a standard neural network learning setting with no regularization applied where the neural network is subject to the training data and will entirely overfit on incorrect data targets (if the capacity is high enough as well as the expressive power being sufficient). Ba and Caruana (2014) concluded that the teacher targets in a KD arrangement eliminate label errors and, thus, facilitates the learning of the student network what supports this hypothesis.

(b) Note that label correction as delineated in Hyp. II **(a)** could already work with hard imitation targets and does not require the dark knowledge in the soft targets since a hard target prediction can already replace the wrong (hard) data target. Nevertheless, we hypothesize that soft targets could improve the label correction ability further. As stated in Sec. 2.6, the class probabilities resulting from a softmax activation cannot be interpreted directly as a quantification about the confidence that the corresponding class might be the correct one but relatively compared to other input samples the probability can be seen as the confidence in (the correctness of) the predicted class. Furlanello et al. (2018) picked up this idea and described the KD gradient induced by the imitation loss (Eq. 2.18) as learning from the training targets using importance weights for each sample based on the teacher's confidence in the predicted class. This form of quantification by the neural network's soft prediction has the possibility of being more fine-grained than only using the (hard) prediction in order to correct a possibly erroneous data target. Thus, the training could be much more stable in early iterations when the neural network is not properly fit (underfitted) on the training data and might produce faulty and diffuse predictions since this low confidence does not affect the data labels much. In contrast, for a pre-trained teacher in a KD setting, the difference in the correction ability of erroneous data labels between the use of soft and hard

targets might not be that decisive than for SIR as the teacher produces reliable predictions from the early beginning.

Hypothesis III (Learn about a class through information about other classes)**.** The main idea behind dark knowledge and further KD is that a soft target can reveal rich information about other classes apart from the one the hard target indicates for a given input (data) sample such as similarities. As similarities can be coded, we assume that learning by using soft targets is supported to the extent that the neural network can also learn about the other classes (apart from the class the hard data target refers to) while a data example is shown.

As an intuition, this might be similar to association when one shows a child something new like a picture of an animal the child has not seen before. It is a usual behavior to associate certain aspects with already existing knowledge and, moreover, to draw further conclusions on this basis. For example, if you show a photo to a child that shows a zebra which the child does not know, the child will soon notice that there are great similarities to a horse, e.g., in terms of physique and anatomy. Thus it can remember a zebra better in the future (e.g., if it is to paint one from memory) because it has seen horses more often and can therefore also transfer aspects of a horse to the zebra, which may not have been visible on the original zebra photo. Furthermore, it will make further assumptions based solely on this association. For example, the child would now paint carrots for food because it assumes that a zebra could also be a herbivore due to the high (visual) similarity to a horse.

Back to neural network learning such an assistance that points out the similarities could be made explicitly available in the imitation loss and can be assumed to improve the learning. In contrast, in classical neural network learning this does not happen as any predicted probability higher than zero for a class, which is not the true one according to the data target, is penalized and pushed lower by the gradient. The partial derivative with respect to such corresponding pre-activation z_j, if a final softmax activation $\boldsymbol{\sigma}(\cdot)$ and cross entropy loss $H(\cdot,\cdot)$ are being used, is simply the probability of the softmax distribution for that pre-activation $\sigma_j(\boldsymbol{z})$ since the desired probability is $(y_j = 0)$ according to the hard data target $\boldsymbol{y}$ (see Eq. 2.16). This means that the pre-activation should be lower after the parameter update by performing a gradient descent step. Moreover, the gradient pushes pre-activations that are higher compared to other ones even stronger (except for the one that corresponds to the data target) since the steepest update step is taken, and so a harder distribution is predicted next time the (input) data sample is shown. On the contrary involving soft targets, the imitation loss would appreciated a predicted probability higher than zero if the soft target carries a non-zero probability for the particular class. Therefore, the gradient of the imitation loss would affect, that the pre-activations are adjusted to fit the soft target probability whereby the neural network's parameter have to adjust such that the class can also be considered as reasonable for the given input sample (whose target originally refers to another class).

We are aware of the fact that the initially stated arguments consider that the soft targets are of a certain quality (e.g., if they come from a pre-trained teacher network). However, it might be possible that similarities between classes are being discovered early in training – simply due to the fact that the neural network makes a wrong classification and thus a wrong soft target in the imitation loss in combination with the correct data target might teach a notion of similarity between the two classes.

Hypothesis IV (Complementary to standard regularization methods)**.** The standard regularization methods, which are considered in this thesis, namely, L^1-norm and L^2-norm regularization, maxnorm and dropout, address the problem of overfitting by explicitly constraining the weights or changing the neural network architecture temporally by dropping neurons, which implicitly intends to affect the predictions of the neural network so that these are more general. SIR, on the other hand, works more abstractly by

regularizing on the basis of existing knowledge and can be interpreted as explicitly addressing the actual prediction distribution of the neural network and thus implicitly adapting the parameters.

Obviously, there is a fundamental difference in the overall approach of SIR and we hypothesize that thus the mechanism of action may be different from the standard regularization methods. Moreover, that this regularization behavior of SIR is complementary to the standard regularization methods. In this case, SIR could be combined with other standard regularization methods so that a better result would be obtained than if each of the two methods had been applied on its own.

However, in the context of KD, it has often been reported that regularizing the student network by dropout or weight decay leads to worsening the generalization performance (Ba and Caruana, 2014; Urban et al., 2017) and could also be observed in preliminary experiments in KD when dropout was activated in the student network, which speaks against this hypothesis.

Hypothesis V (Data efficiency)**.** On the one hand, soft targets could generally be considered as an enrichment over hard targets and Hinton et al. (2015) showed, on the other hand, that it was possible to train a student network in a KD setting on an extremely small subset of the original training data (the full set was used to train the teacher beforehand) without further loss in generalization compared to the baseline where the student network (architecture) was trained on all training data available. Based on these results, Hinton et al. (2015) concluded that the further information that is encoded in the soft targets must be crucial in learning and generalizing.

Therefore, we hypothesize that by using soft targets (and thus if SIR is applied), the neural network can still achieve good results in terms of generalization with less data, making the training more data efficient. This feature is very much in demand in practical applications, either because labeling data is associated with very high costs or simply because there is not much (input) data available for a problem, e.g., because it is not easy to obtain.

Again, one should be aware of the fact that the stated arguments consider that the soft targets are of a certain quality (e.g., if they come from a pre-trained teacher network) but as argued in Hyp. II (Compensate for erroneous data labels), the diffuse predictions that are produced in the early beginning of training when the model is underfitted should not bias the training much (the data loss prevails in the gradient) and basic similarities should have been discovered quickly (as argued in Hyp. III; Learn about a class through information about other classes).

3.2 Further variants of Self-Imitation Regularization (SIR)

In this section, further variants besides the ones that are already described above in the introduction of the method of SIR (e.g., target gradients or asymmetric vs. symmetric SIR), are proposed. Potentially, these can bring advantages, which, however, did not become considerably apparent in preliminary studies, which is why these are not regarded as direct components of SIR and are not extensively investigated in the context of this thesis.

3.2.1 Penalty formulation

The loss formulation of SIR, as stated in Eq. 3.3, can be construed as a multi-objective loss constructed by the weighted average between the imitation loss (Eq. 3.1) and the data loss (Eq. 3.2) where the imitation rate π controls the weighting.

On the other hand, one could consider reformulating this loss so that the imitation loss penalizes the data loss. In the following, this concept is referred to as *penalizing SIR* and formally results in the alternative loss formulation

$$\begin{aligned} L^{\text{pSIR}}(\boldsymbol{Z}, \boldsymbol{Y}; \boldsymbol{\theta}) &= \frac{1}{M} \sum_{m=1}^{M} l^{\text{pSIR}}(\boldsymbol{z}^{(m)}, \boldsymbol{y}^{(m)}; \boldsymbol{\theta}) \\ &= \frac{1}{M} \sum_{m=1}^{M} \Big(H(\boldsymbol{y}^{(m)}, \boldsymbol{\sigma}(\boldsymbol{z}^{(m)})) \quad + \quad \pi_p \cdot H(\widehat{\boldsymbol{\sigma}}(\boldsymbol{z}^{(m)}), \widetilde{\boldsymbol{\sigma}}(\boldsymbol{z}^{(m)})) \Big) \end{aligned} \tag{3.4}$$

where $\pi_p \geq 0$ controls the imitation strength.

This choice can be motivated in three ways: first, many regularization methods incorporated in the loss function are applied as a penalty (e.g., the norm-penalty regularization methods shown in Sec. 2.7.1). Second, this is more neatly related to solving a constrained optimization problem via the Lagrangian with the objective to minimize the data loss subject to the constraint that the imitation loss should be low (i.e., equal to zero). The obtained Lagrangian multiplier is the imitation strength π_p hyperparameter, defining the hardness of the constraint (Boyd and Vandenberghe, 2015, ch. 5.1.4, p. 218). Third, this could be preferable for implementation as it only requires the addition of the imitation loss to the data loss and does not require the data loss to be modified (i.e., scaled).

In the following, the equivalence up to a scaling factor, i.e., $1/(1-\pi)$, between the penalizing SIR loss (Eq. 3.4) and the standard (multi-objective) SIR loss (Eq. 3.3) can be easily shown

$$\begin{aligned} l^{\text{SIR}} &= (1-\pi) \cdot l^{\text{data}} + \pi \cdot l^{\text{imit}} \\ \underbrace{\frac{1}{1-\pi} \cdot l^{\text{SIR}}}_{= l^{\text{pSIR}}} &= l^{\text{data}} + \underbrace{\frac{\pi}{1-\pi}}_{= \pi_p} \cdot l^{\text{imit}} \end{aligned} \tag{3.5}$$

where imitation loss and data loss are abbreviated with l^{imit} and l^{data}, respectively, for the sake of clarity.

Furthermore, preliminary experiments comparing the application of SIR in the standard (i.e., multi-objective) loss formulation to the penalizing SIR loss formulation have shown that there is no apparent difference so both formulations can be used interchangeably. Therefore, in the following, the formula for conversion between the imitation strength π_p and the imitation rate π is presented that keeps the ratio between the influences of the imitation loss and the data loss. This guarantees that the intention and aim in setting the hyperparameter for one loss variant are maintained when switching between the loss formulations. As already derived in Eq. 3.5, given π, the equivalent π_p is defined by $\pi_p = \pi/(1-\pi)$ whereas $\pi = \pi_p/(1+\pi_p)$ if π_p is given since

$$\begin{aligned} \pi_p &= \frac{\pi}{(1-\pi)} \\ \pi_p - \pi \cdot \pi_p &= \pi \\ \pi_p &= \pi + \pi \cdot \pi_p \\ \pi_p &= \pi(1+\pi_p) \\ \frac{\pi_p}{1+\pi_p} &= \pi \end{aligned}$$

applies.

It is left to be said that due to the simple addition of the imitation loss penalty, the penalizing SIR formulation increases the magnitudes of the gradients over the ones of the gradients that are induced by the data loss alone (i.e., without the appliance of SIR at all) or even over the ones of the multi-objective SIR loss formulation. The increase factor of the gradients of the loss is exactly $1/(1-\pi)$ if the same ratio between the imitation and data loss is kept as shown in Eq. 3.5. The standard (multi-objective) SIR loss formulation eliminates or at least keeps this difference small since the two loss functions are averaged, i.e., both losses are scaled down and so the gradients by π and $(1-\pi)$, respectively. This limitation of the gradients' magnitudes can be beneficial in practice by, e.g., simplifying the tuning of the learning rate α for the gradient descent optimization (see Sec. 2.2.4). This is because α has to be chosen smaller if the gradients can adopt more differently large magnitudes between while training in order to prevent too large parameter update steps that could otherwise destabilize training. This is the reason why the multi-objective loss formulation is applied in the conducted experiments and unexceptionally considered in the rest of the thesis.

3.2.2 Target network

In general, the training samples that are used for learning through the loss computation based on the data targets are assumed to be i.i.d. (as mentioned in Sec. 2.1.4). However, from the perspective of the imitation loss in SIR (Eq. 3.1), the imitation targets $\widehat{\boldsymbol{y}}$ are obviously not i.i.d at all. The main reason for this is that the imitation targets change over time (while learning) as the neural network's parameters $\boldsymbol{\theta}$ change since the imitation targets depend on the neural network (i.e., on its parameters $\boldsymbol{\theta}$). The violation of the i.i.d. assumption could entail serious consequences with regard to training instability, such as the parameters $\boldsymbol{\theta}$ in the neural network diverging, and the neural network will not learn anything.

The problem that targets, which are involved in a loss computation, depending on the neural network's parameters $\boldsymbol{\theta}$, is also commonly present in the field of RL, e.g., in the widely applied approach of Q-learning, which is explained in Sec. 2.1.1. The core idea behind the *Deep Q-Networks (DQN)* algorithm (Mnih et al., 2013) is to put a (deep convolutional) neural network into use to approximate the Q-function, which is referred to as the Q-network, via Q-learning. Consequently, the loss computation involves the temporal difference error, where the targets depend on the neural network's parameters $\boldsymbol{\theta}$. Since this is deemed unstable, an improvement of the DQN algorithm was subsequently proposed in Mnih et al. (2015): the concept of a *target network*. Here, the targets for the Q-network training loss are no longer obtained using the current parameters $\boldsymbol{\theta}$. Instead, the parameters, denoted as $\boldsymbol{\theta}^-$, are only updated with the Q-network parameters every C iterations and held fixed in between resulting in the so-called target network (i.e., containing the parameters $\boldsymbol{\theta}^-$). This stabilizes the training since the targets, the Q-function estimate is fitted on, stay fixed for some time.

This idea could be taken up for SIR by periodically copying (e.g., after C epochs) the current parameters $\boldsymbol{\theta}$ and held them fixed to form a target network (with the parameters $\boldsymbol{\theta}^-$) that produces the imitation targets $\widehat{\boldsymbol{y}} = \widehat{\boldsymbol{\sigma}}(\boldsymbol{z}^-)$ where $\boldsymbol{z}^-$ is the pre-activation vector predicted by the target network whereas everything remains the same and, thus, using the neural network with the current parameters $\boldsymbol{\theta}$ for the predictions $\widetilde{\boldsymbol{\sigma}}(\boldsymbol{z})$ in the imitation loss. By doing so, the i.i.d. assumption holds over the period the target network parameters $\boldsymbol{\theta}^-$ are held fixed since the target network produces constant targets per input sample during this time. This would engage a more stable training as the neural network can approach the imitation targets without them moving away. Furthermore, it is worthwhile mentioning that this naturally resolves the problem of symmetric SIR without target gradients where, in contrast to KD, only zero gradients are obtained because the targets and predictions in the imitation loss differ as they are

now produced from differently parameterized neural networks; except in the few iterations where the copy $\boldsymbol{\theta}^{-}$ just has been renewed with $\boldsymbol{\theta}$.

According to preliminary studies, the appliance of the target network does not bring any considerable or obvious advantages in terms of improved stability or generalization. Moreover, according to the experimental results, standard SIR (without a target network) already turns out to be very stable, and beyond that, it is even able to stabilize the training of a neural network that is not able to learn anything from the training data without regularization or even with some standard regularization methods applied (for more details on this, refer to Sec. 5.5). On the other hand, without incorporating the target network, the hyperparameter C denoting the target network parameter update period is not involved, which simplifies the hyperparameter tuning. Overall, for these reasons, this SIR variant is not further examined in this thesis and is left for future work.

3.3 Preliminary studies in Knowledge Distillation (KD)

The results of preliminary studies on the method of KD are briefly presented in this section. These were conducted to validate the major modifications to KD that are used in order to make SIR functioning and to get a sense in advance whether these might be meaningful in SIR or already lead to the basic KD principle collapse. If these modifications do not hold for KD where high-quality teacher/imitation targets are available, then one can imagine that the chance of success in SIR is even lower. The two approaches that are investigated here are the chained softmax function as a replacement for the temperature softmax function and the asymmetric (softmax) imitation compared to the symmetric imitation as per the KD method.

For the KD experiments, a pre-trained teacher neural network with a `ResNet` architecture consisting of 18 layers (see Sec. 4.3 for details on this architecture) achieving a test accuracy of 92.10% was used. The student neural network has a standard convolutional neural network architecture with three convolutional layers (with a 3x3 convolution kernel), each connected with max pooling, followed by two fully-connected layers. The ReLU activation function is applied after each layer except for the output layer. If the student neural network is learned without KD, it achieves a test accuracy of 86.17%.

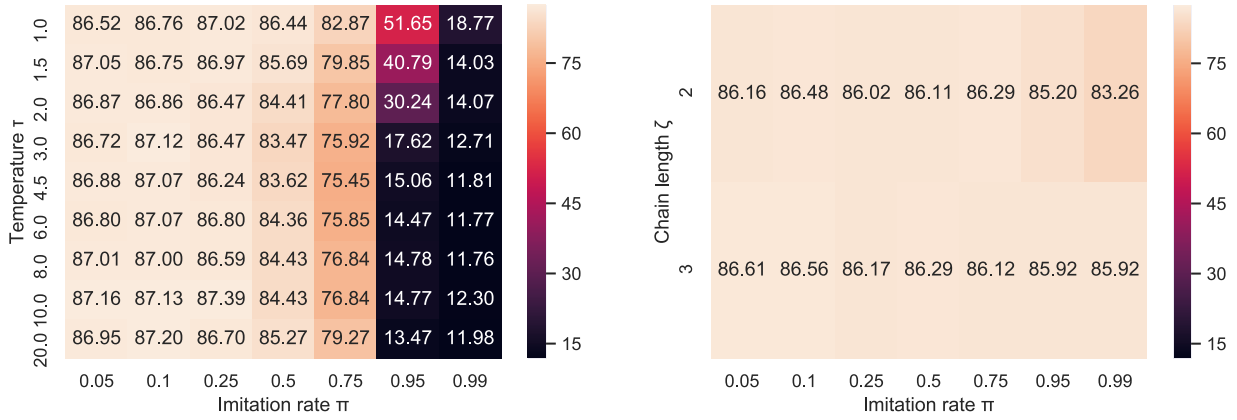
Note that the implementation of Li (2018), which is provided on GitHub[1], was used for the execution of the preliminary experiments the KD. For this reason, the results obtained in the preliminary experiments are not comparable to the results of the main experiments in this thesis, which are explained in Sec. 4 and discussed in Sec. 5. Furthermore, in this implementation, the data (i.e., the CIFAR-10 data set (Krizhevsky, 2009)) is heavily pre-processed via normalization and data augmentation (Goodfellow et al., 2016, ch. 7.4; Mikołajczyk and Grochowski, 2018) by randomly cropping and horizontal flipping the image data.

3.3.1 Comparison: temperature and the chained softmax function

First of all, the experimental results have shown that KD with the chained softmax function as replacement for the temperature softmax function works and reaches similar performance (in terms of test accuracy).

To examine the effects of the different hyperparameter settings in more detail, Fig. 3.3 visualizes the achieved test accuracy scores in heat maps for both the temperature softmax function (Fig. 3.3a) and the chained softmax function (Fig. 3.3b). Through this visualization, it becomes strikingly apparent that the

[1] `https://github.com/peterliht/knowledge-distillation-pytorch` (April 15th, 2019); Many thanks for sharing.



(a) Temperature softmax function

(b) Chained softmax function

Figure 3.3.: Heatmaps showing the test accuracy scores achieved with different hyperparameter settings regarding the knowledge distillation method (i.e., imitation rate π as well as τ and ζ for the temperature and chained softmax function, respectively). The results are presented for **(a)** the temperature softmax function and **(b)** the chained softmax function.

chained softmax function is less sensitive to the hyperparameter choice than the temperature softmax function is. This applies for both scenarios: varying the softmax function hyperparameter τ respectively ζ while leaving the imitation rate π fixed as well as varying the imitation rate π while leaving the softmax function hyperparameter τ respectively ζ fixed. Furthermore, the variance induced by the choice of hyperparameters is much higher in the temperature softmax function case, and no heuristics exist according to which the τ can be set. Moreover, the wrong choice of hyperparameters in the temperature softmax function has drastic consequences and destabilizes the training almost completely.

For this reason, in nearly all experiments conducted in the context of this thesis in order to investigate the method of SIR, the 2-chain softmax function is used as a replacement for the temperature softmax function (whenever a softened softmax distribution is desired). Although the maximum achieved test accuracy scores in the preliminary KD experiments are (slightly) higher with the temperature softmax function (but be aware that comparably more temperature softmax experiments were conducted). Nevertheless, using the 2-chain softmax function eliminates the need to tune the temperature hyperparameter τ at all, which simplifies the investigation and hyperparameter tuning much. This makes it possible to focus on the imitation rate π without encountering the problem that the result correlates much with the softmax function hyperparameter choice.

3.3.2 Comparison: Symmetric and asymmetric imitation

The results regarding preliminary experiments comparing the symmetric imitation and two different asymmetric imitation settings (where once the student's prediction is softened and once the teacher's prediction) in KD are aggregated in Table 3.1. These show that asymmetric imitation does not interfere with KD. Much rather the chained softmax function considerably improved over the symmetric imitation when asymmetric imitation is applied so that the teacher's prediction (i.e., the imitation targets) are softened. Here the gap to the maximum achieved test accuracy with the temperature softmax function is almost nullified. Overall, the result that asymmetric imitation where the student's prediction is only softened and the imitation target is not is ineffective was expected since the softening of the imitation

	Symmetric		Asymmetric, soft student		Asymmetric, soft teacher	
	Max.	**Min.**	**Max.**	**Min.**	**Max.**	**Min.**
Temperature softmax	87.39	11.76	87.17	13.02	87.22	11.72
Chained softmax	86.61	83.26	86.58	62.29	87.21	56.46

Table 3.1.: The results from KD experiments using either symmetric, asymmetric with softened student distributions or asymmetric with softened teacher distributions. Further, either a temperature or a chained softmax function was used to produce the softened distributions. Both, the minimum and maximum test accuracy that were achieved over several experiments within an experimental configuration are presented.

targets should contribute to the dark knowledge becoming revealed. Hence, asymmetric imitation is used for SIR only in a sense that the imitation targets are softened more than the imitation predictions.

Note that these experimental results demonstrate again that the chained softmax function is more stable than the temperature softmax function comparing the minimum achieved test accuracy scores.

4 Experiments

4.1 Experimental set-ups

In order to thoroughly investigate the method of SIR and to point out specific properties (especially regarding the formulated hypotheses in Sec. 3.1), different types of experiments were conducted, and their set-ups will be outlined in the following. The results are presented, discussed and analyzed in Sec. 5. Generally, the experiments do not aim to achieve the highest performance possible, e.g., not trying to achieve a new state-of-the-art performance, and therefore, the hyperparameters have not been tuned extensively. Rather, they were kept the same between the experiments so that the results are comparable and observed effects can be attributed to isolated modifications, such as the use of SIR. In detail: all neural networks were trained over 100 epochs, with mini-batch size $M = 64$ and using the Adam optimizer with the hyperparameters as suggested by Kingma and Ba (2014), namely with learning rate $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. Furthermore, the random seeds were set fixed for all experiments to ensure that the (random) neural network initialization and the ordering in which the training samples are drawn are the same over all experiments. The neural networks are initialized according to the *Xavier initialization* (Glorot and Bengio, 2011).

4.1.1 Noisy data labels

Directly related to Hyp. II (Compensate for erroneous data labels), the ability to correct erroneous data labels and the robustness against overfitting on such labels are being examined in this type of experiments.

For this purpose, noisy data labels are used for learning, which are obtained by randomly selecting and assigning another class to a random portion of the training data labels. Formally, this is expressed as *label noise* on a certain portion of data. For instance, applying label noise on 20% of the training data yields that the label of every fifth training sample is incorrect (on average). Obviously, the test data labels remain untouched to measure the final (generalization) performance and, thus, to infer the robustness against overfitting on the noisy labels.

Note that this artificial data label noise cannot reliably simulate all types of erroneous data labels that might occur in practice since this noise is (uniformly) distributed over *all* classes. Systematical or biased mistakes as they might occur in manual labeling if some classes are more difficult to distinguish than others, for example, cannot be captured by this approach. On the other hand, a lot of label noise is used (up to 50% in some experiments to make learning particularly difficult), and it is considered rather unlikely to occur in practice that such a high false rate is present in the data labels.

4.1.2 Learning on a small training data subset

For some experiments, the number of training data samples is reduced (by preserving the class distribution of the original training data set). These experiments are intended to serve as evidence regarding

Hyp. V (Data efficiency). Note that other types of experiments are frequently combined with this configuration since learning on a very small amount of training data increases the susceptibility to overfitting.

4.1.3 Reduced training samples of a class

One of the main motivations about making the dark knowledge explicitly available in the loss formulation, and probably the primary idea in KD, is that information about the other classes in terms of similarities and correlations are revealed.

In order to study how well SIR enables to learn about a class through the similarities and correlations to other classes as hypothesized in Hyp. III (Learn about a class through information about other classes), the training samples of a particular class are reduced in some experiments. Therefore, it is made more challenging for the neural network to acquire a general concept for this class and, simultaneously, the risk of overfitting on these few training samples for this class increases. This experimental setting is based on Hinton et al. (2015) where they completely removed all samples of a particular class when training the student neural network with KD while the teacher neural network was trained on the full data set beforehand. They were able to achieve a remarkable high test performance with the student neural network for the class that was not available in the training data which shows the immense support of the soft teacher targets. We aim to reproduce the learning through the samples from other classes when applying SIR.

4.1.4 Standard regularization methods

Apart from applying SIR for regularization purposes in the experiments only, additional experiments were conducted applying standard regularization techniques (as presented in Sec. 2.7) to the neural networks. These results enable us to tell whether effects observed with SIR are unique to this method or can also be achieved by other (standard) regularizers.

Furthermore, experiments incorporating combinations of SIR and standard regularization methods were carried out to scrutinize the complementarity of SIR and standard regularization techniques (Hyp. IV; Complementary to standard regularization methods). Since the number of possible combinations of various regularizers grows (exponentially) with the number of regularizers involved in a single combination, we have limited the experiments to complement SIR with a single additional standard regularizer. Otherwise, the possibility of performing an ablation study (e.g., as done by Hessel et al. (2017)) would have existed where all regularization techniques (including SIR) are used together and the result is compared to the results obtained from the combinations where a single regularization technique is dropped. This reveals which regularizer had the strongest impact on the entire combination. However, this exacerbates the problem of finding proper hyperparameters for the regularizers and would require a lot of computing time and effort. This is also the reason why the elastic net regularization was not used since the tuning of the two hyperparameters controlling the L^1-norm and L^2-norm penalty is particularly sensitive and often ends up with only one of the two penalties coming into effect.

4.2 Data sets

Since deep learning revolutionized computer vision in recent years, (deep convolutional) neural networks are used over various practical applications within this field today and continue to be heavily researched (e.g., for autonomous driving). Moreover, many visual data sets have been established as

Data set	Classes	Train samples	Balanced	Test samples	Res.	No. channels
MNIST	10	60000	– (almost)	10000	28x28	1
CIFAR-10	10	50000	✓	10000	32x32	3
CIFAR-100	100	50000	✓	10000	32x32	3
EMNIST letters	26	124800	✓	20800	28x28	1
EMNIST digits	10	240000	✓	40000	28x28	1
EMNIST combined	47	112800	✓	18800	28x28	1
Fashion-MNIST	10	60000	✓	10000	28x28	1
KMNIST	10	60000	✓	10000	28x28	1
SVHN	10	73257	–	26032	32x32	3

Table 4.1.: Information about the data sets where the resolution (**Res.**) is given in pixels, **Balanced** indicates whether the training samples are equally distributed over the classes, and **No. channels** refers to the number of values describing one pixel: 1 → grayscale or 3 → colored (RGB).

benchmark data sets to evaluate new developments around neural networks. Therefore, the experiments in this thesis were focused on classification problems on visual data.

Facts about the data sets used, such as the number of classes, the amount of data samples and properties of the images, are summarized in Table 4.1. Note that the data sets are used as they are in the experiments without further preprocessing such as PCA whitening or ZCA whitening (Krizhevsky, 2009), but are padded to a resolution of 32x32 pixels if necessary.

MNIST:

The MNIST data set (LeCun, 1998), probably the most famous (visual) data set in ML, is a collection of 28x28 pixels grayscale images showing handwritten digits. Consequently, this data set contains $n = 10$ classes representing the 10 Arabic numerals. Some train samples are shown in Fig. 4.1d.

CIFAR-10:

The CIFAR-10 data set (Krizhevsky, 2009) is a labeled subset of the 80 million tiny images data set (Torralba et al., 2008) with $n = 10$ classes $\mathcal{C} = \{plane, car, bird, cat, deer, dog, frog, horse, ship, truck\}$. The images are down-scaled (32x32 pixels) colored photos showing real-world objects and animals, e.g., as presented in Fig. 4.1a.

CIFAR-100:

The CIFAR-100 data set (Krizhevsky, 2009) is equal to the CIFAR-10 data set in terms of resolution and colored real-world photos. The images are also taken from the 80 million tiny images data set (Torralba et al., 2008) with difference that $n = 100$ classes are present. The class labels are $\mathcal{C} = \{$*apple*, *aquarium fish*, *baby*, *bear*, *beaver*, *bed*, *bee*, *beetle*, *bicycle*, *bottle*, *bowl*, *boy*, *bridge*, *bus*, *butterfly*, *camel*, *can*, *castle*, *caterpillar*, *cattle*, *chair*, *chimpanzee*, *clock*, *cloud*, *cockroach*, *couch*, *crab*, *crocodile*, *cup*, *dinosaur*, *dolphin*, *elephant*, *flatfish*, *forest*, *fox*, *girl*, *hamster*, *house*, *kangaroo*, *keyboard*, *lamp*, *lawn mower*, *leopard*, *lion*, *lizard*, *lobster*, *man*, *maple tree*, *motorcycle*, *mountain*, *mouse*, *mushroom*, *oak tree*, *orange*, *orchid*, *otter*, *palm tree*, *pear*, *pickup truck*, *pine tree*, *plain*, *plate*, *poppy*, *porcupine*, *possum*, *rabbit*, *raccoon*, *ray*, *road*, *rocket*, *rose*, *sea*, *seal*, *shark*, *shrew*, *skunk*, *skyscraper*, *snail*, *snake*, *spider*, *squirrel*, *streetcar*, *sunflower*, *sweet pepper*, *table*, *tank*, *telephone*, *television*, *tiger*, *tractor*, *train*, *trout*, *tulip*, *turtle*, *wardrobe*, *whale*, *willow tree*, *wolf*, *woman*, *worm*$\}$ and a (training data set) example image for each class is depicted in Fig. 4.1b.

The classes are further categorized into the 20 super classes *aquatic mammals*, *fish*, *flowers*, *food containers*, *fruit and vegetables*, *household electrical devices*, *household furniture*, *insects*, *large carnivores*, *large man-made outdoor things*, *large natural outdoor scenes*, *large omnivores and herbivores*, *medium mammals*, *non-insect invertebrates*, *people*, *reptiles*, *small mammals*, *trees*, and two types of vehicles (*vehicles 1* and *vehicles 2*). For instance, this is useful to get more information about relationships (e.g., similarities) between the classes. However, the 100 (sub-) classes were used for the neural network training in the experiments carried out in the context of this thesis.

EMNIST:

EMNIST is an extension of MNIST to handwritten letters (Cohen et al., 2017) and the images are therefore of the same resolution (and also grayscale). Some examples of handwritten letters are printed in Fig. 4.1g and it becomes apparent that handwritten letters in both lower case and upper case variants are present. EMNIST does not refer to a single data set but rather to a collection of MNIST-like (with handwritten letters in addition) data sets. The EMNIST data sets that were used in the experiments are the following:

- **EMNIST letters:** Contains $n = 26$ classes for the handwritten letters (case insensitive).
- **EMNIST digits:** Only handwritten digits as in MNIST but more training and test samples (by a factor of two and four, respectively).
- **EMNIST combined:** Contains both handwritten digits and letters. In principle, it is case sensitive, but letters that have the same appearance regardless of the case are merged into one class. This is the case for the letters C, I, J, K, L, M, O, P, S, U, V, W, X, Y and Z, and results in a total number of $n = 47$ classes.

Fashion-MNIST:

The Fashion-MNIST dataset (Xiao et al., 2017) contains 28x28 grayscale images of fashion products from $n = 10$ different categories/classes as shown in Fig. 4.1e. The data set was intentionally designed to serve as a replacement for the classical MNIST data set to test implementations of machine learning algorithms that already support the MNIST data set (i.e., for any ML frameworks that are compatible with MNIST) but it is considered a more challenging task due to the much more diverse images of fashion products. The classes are $\mathcal{C} = \{$*t-shirt/top*, *trouser*, *pullover*, *dress*, *coat*, *sandal*, *shirt*, *sneaker*, *bag*, *ankle boot*$\}$.

KMNIST

The KMNIST (abbreviation of Kuzushiji-MNIST) data set (Clanuwat et al., 2018) is a subset of the Kuzushiji data set. This data set is a visual data set with ancient Japanese characters (written in cursive Kuzushiji style; examples are depicted in Fig. 4.1f) that are used in classical Japanese literature (for over 1,000 years and printed in books until 150 years ago). However, most Japanese natives today cannot read it. The Kuzushiji character images are labeled with the modern Hiragana counterpart. The KMNIST data set serves as a replacement for MNIST and, therefore, comprises 28x28 grayscale images of $n = 10$ different classes (i.e., character types). In contrast, the full Kuzushiji data set contains 3,999 character types.

Architecture	No. parameters	Depth	Conv. kernel	Max pooling region	Activation
LeNet5	62 006	$2+3=5$	5x5	2x2	ReLU
ConvNet	10 071 562	$3+3=6$	5x5	3x3	ReLU
ResNet-8	78 042	$7+1=8$	3x3	No	ReLU
ResNet-20	272 474	$19+1=20$	3x3	No	ReLU
ResNet-32	466 906	$31+1=32$	3x3	No	ReLU
ResNet-56	855 770	$55+1=56$	3x3	No	ReLU
ResNet-110	1 730 714	$119+1=110$	3x3	No	ReLU

Table 4.2.: Information about the neural network architectures where **Depth** shows the number of convolutional layers, the number of fully-connected layers and the total without counting the input layer (No. conv. + No. fully-connected = No. total). Note that the actual number of parameters varies depending on the number of classes ($n = 10$ classes are assumed here). Note that the ResNet architectures also incorporate convolutional layers with a 1x1 kernel.

SVHN:

The SVHN (Street View House Numbers) data set (Goodfellow et al., 2014) provides a digit classification problem ($n = 10$ classes), but in comparison to the optical character recognition data sets from above, the images are real-world photographs. Namely, house numbers from Google's Street View imagery as exemplarily printed in Fig. 4.1c. Two more facts, making the SVHN data set different from the other character recognition data sets presented before is that the images are colored (as images in CIFAR-10/100) and that the digits are not narrowly cropped so that it happens that further digits can (partly) appear at the sides of the image, if it is a multi-digit house number, from which the photo was taken.

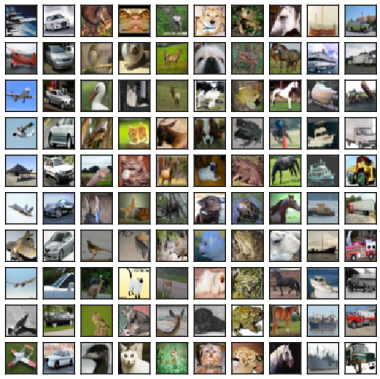
4.3 Neural network architectures

As the focus is on visual data and image classification in the experiments within the scope of this thesis, convolutional neural networks are put to use. All facts and information about the neural network architectures are aggregated in Table 4.2.

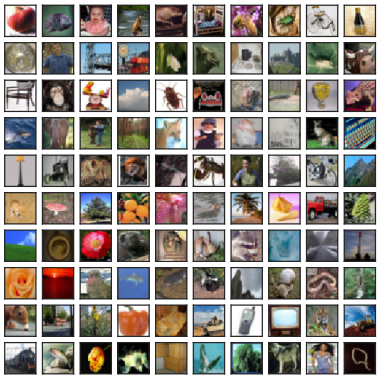
The architectures utilized the most are referred to as LeNet5 and ConvNet in the course of the thesis. The LeNet5 architecture follows exactly the definition in LeCun et al. (1998) and is a convolutional neural network of low capacity (i.e., a low number of parameters with 62 006). When dropout is used, only the activations of the last convolutional layer as well as the first fully-connected layer are affected, and the dropping rate is commonly set to $p = 0.5$ (but it is specified in the discussion of the experimental results).

In sharp contrast to this, the ConvNet architecture is constructed by Srivastava et al. (2014) with the intention of obtaining a high capacity model that is more susceptible to overfitting and therefore relies on effective regularization methods. When dropout is applied, the neuron dropping rate p is set individually per layer. Specifically, rates of 0.1, 0.25, 0.25, 0.5, 0.5 and 0.5 are used going from the input to convolutional to fully-connected layers except the output layer/activations (Srivastava et al., 2014).

The ResNet architectures (He et al., 2015) are especially capable of learning while being very deep. This is achieved by introducing so-called skip connections, whereby the activations of a layer can skip some layers unchanged and then be made available for further processing. This allows the neural network to more flexibly determine how much depth is really needed as layers that are not necessary can be simply skipped. Further peculiarities of this architecture are that a modification of the Xavier initial-

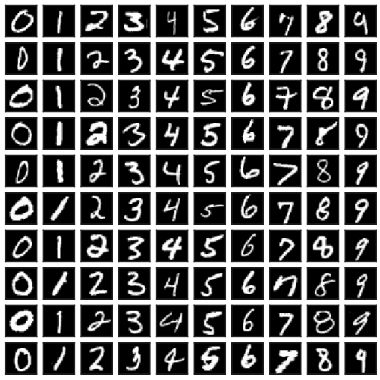


(a) CIFAR-10



(b) CIFAR-100

(c) SVHN



(d) MNIST

(e) Fashion-MNIST

(f) KMNIST

(g) EMNIST letters

Figure 4.1.: Samples taken from the (training) data sets **(a)** CIFAR-10, **(b)** CIFAR-100, **(c)** SVHN, **(d)** MNIST, **(e)** Fashion-MNIST, **(f)** KMNIST and **(g)** EMNIST letters. Each column shows ten examples of a single class except for the CIFAR-100 data set where only one sample is presented for each of the 100 classes.

ization, batch normalization (Ioffe and Szegedy, 2015) and average pooling before the (fully-connected) output layer are applied. Furthermore, the *basic block* variant is used – please refer to the original paper (He et al., 2015) for more detailed information on this. The `ResNet` architecture is used in the depths of 8, 20, 32, 56 and 110 layers to enable investigations of the effects in the neural network with respect to the depth (e.g., the results are presented in Sec. 5.6). Note that no dropout is used in the `ResNet` architecture in the experiments since He et al. (2015) did not use it nor suggested to what extent dropout could be applied in such a deep and thin neural network architecture. Furthermore, it is generally discouraged to combine batch normalization with dropout (Li et al., 2018).

Apart from the presented architectures that are intensively used in the experiments and are thoroughly investigated, two further architectures were put to use in preliminary experiments and showed similar results. Namely, the `AllConvNetC` exactly as defined in Springenberg et al. (2014), which is an architecture where only convolutional layers but neither pooling nor fully-connected layers are used, as well as the `MaxoutConvNet` (Goodfellow et al., 2013) where the neural network contains maxout units (explained in Sec. 2.4.2) instead of rectified linear units (ReLUs).

4.4 Evaluation methodologies

4.4.1 Metrics

Since the performance of a model can relate to a variety of aspects, there are also several metrics that can be used to measure its performance in terms of different aspects. The metrics used to evaluate the experimental results are presented in this section.

Note that for each conducted experiment, the performance scores, models and outputs were stored for both the best performing model according to the test accuracy throughout training and the final model at the end of training (after 100 epochs). The accuracy of the (best) interim model is briefly referred to as the *best accuracy* in the discussion of the experimental results.

Accuracy:

The *accuracy* is a very simple but powerful metric to measure a model's performance since it expresses the relative number of correctly predicted samples of a data set. This metric ranges between zero and one where higher values indicate a more accurate model, and can also be understood as a percentage. A model that does not make any mistakes has an accuracy score of 100% while a randomly acting one achieves an accuracy score of $1/n$ on average in the case of evaluating on a data set with n balanced classes. Formally, the accuracy of a model $f(\boldsymbol{x};\boldsymbol{\theta})$ on a given data set $\mathcal{D} = \{(\boldsymbol{x}^{(d)}, y^{(d)})\}_{d=1}^{D} \subseteq (\mathcal{X} \times \mathcal{Y})$ where $\mathcal{Y} \subset \mathbb{N}$ refers to the indices of all n classes $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$ in a classification problem, is defined as follows

$$\frac{1}{D}\sum_{d=1}^{D} I_{y^{(d)}, f(\boldsymbol{x}^{(d)};\boldsymbol{\theta})}$$

where $I_{i,j}$ are the elements of the $n \times n$ identity matrix $\boldsymbol{I}$ or, alternatively, this can be seen as evaluating the indicator function indicating whether i and j are equal (i.e., evaluates to 1) or not (i.e., evaluates to 0).

Precision:

The *precision* measures the performance on a particular class. It is defined by the percentage share of the instances that were correctly classified with this class. Therefore, it ranges from zero to one and favors precise models when predicting the considered class, but in return, they might rather predict this class less often.

The precision is formally defined as

$$\frac{tp}{tp + fp}$$

where tp counts the true positives and fp the false positives (regarding the investigated class).

Recall:

The *precision* measures the performance on a particular class and can be seen as a counterpart to the precision. It is defined by the percentage share of correctly classified instances from this class. It ranges from zero to one, and favors models that predict the considered class for as many instances that are from this class as possible, but does not devalue models predicting this class for instances that are not from this class.

The recall is formally defined as

$$\frac{tp}{tp + fn}$$

where tp counts the true positives and fn the false negatives (regarding the investigated class).

F_1 score:

The F_1 *score* is the harmonic mean of precision and recall (Chinchor, 1992; Sasaki, 2007) and is therefore intended for use if both precision and recall should be maximized (equally weighted). This is often appreciated since heavily optimizing a model only according to one of these evaluation criteria might result in undesired and absurd models. For example, a model classifying only one instance correctly in the investigated class results in an optimal precision but a very low recall whereas always predicting the investigated class entails an optimal recall but a low precision.

The F_1 score is formally defined as

$$2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Position error:

The *position error* (Hüllermeier and Fürnkranz, 2005) is a metric that is used in multi-label classification (outlined in Sec. 2.1.1) but can be adapted to measure the quality of a ranking of classes generated by the softmax prediction probabilities or the pre-activation values of a neural network. The position error measures the deviation of the predicted target label's position from the top-rank. The higher its position (i.e., the lower is the deviation from the top-rank) the better (i.e., the lower the position error). In this case, the model considers this label as more important (i.e., more likely that this would be the correct prediction) than if its ranking position is low. The position error is closely related to the conventional (classification) error (i.e., $1 -$ accuracy) but for a misclassification, the position of the target label in the predicted ranking is considered. This provides a finer criterion for distinguishing the prediction quality of two models, even if they achieve the same accuracy on a data set.

Given that the target class c' got assigned the ranking position $\text{rank}(c') \in \{1, 2, \ldots, n\}$ where n is the number of classes/labels and the top-rank is n where the target is desired to be ranked at. Formally, the *normalized position error* (Hüllermeier and Fürnkranz, 2005) is defined as

$$\frac{(n - \text{rank}(c'))}{n - 1}$$

where the error values range between zero and one.

Note that a model that predicts random rankings achieves an average (normalized) ranking error of 50% instead of 0%. Since in order to achieve a ranking error of zero, the model must always rank the target class intentionally on the lowest position.

Confusion matrix:

The *confusion matrix* is a detailed representation of the prediction performance from which several metrics, such as the accuracy, precision or recall, can be derived and calculated. It is a matrix that shows all cases of classification as well as misclassification of a model evaluated on a given data set. The y-axis refers to the (data) target classes while the x-axis refers to the predicted classes. Now, the i, jth element (ith row and jth column) of this matrix carries the number of times the model predicted the class c_j (the class with the class index j) when the target class of these data instances was actually c_i (the class with the class index i). Consequently, the number of correct classifications can be found along the diagonal of the confusion matrix since $i = j$ applies for the diagonal elements and all misclassifications in the rest of the elements in the confusion matrix ($i \neq j$). Therefore, the higher the numbers on the diagonal and the lower the remaining numbers in the confusion matrix, the better (i.e., more accurate) the model predicts. This representation is richer than the single accuracy score since it provides insights in specific issues like indicating particular classes the model frequently mix up.

As an example (visualization), please refer to Fig. 5.10. Here, a poorly functioning neural network that always predicts a single class can be indicated by its confusion matrix in Fig. 5.10a, which can be concluded since all elements are zero except the ones of a single column. The confusion matrix of a far better behaving model is shown in Fig. 5.10b as the numbers along the diagonal clearly stands out from the others.

Entropy:

As Hinton et al. (2015) suggest, soft targets that have a high entropy can be seen as more informative than hard targets. Furthermore, this is the original information theoretic interpretation of the entropy describing how much information is carried within the given distribution. Therefore, the entropy measurement is used to determine the informational content of the softmax predictions, as well as the soft targets used for the self-imitation, and make it comparable between neural networks that were learned differently. However, one has to be cautious because the maximum entropy value implies a uniform distribution, and it is controversial to speak of high informational content with such a soft target.

As the entropy depends on the number of classes n, the *normalized entropy* of a predicted probability distribution $\boldsymbol{p}$ (i.e., expressed as a vector $\boldsymbol{p}$ of probabilities) is defined so that it ranges from zero to one as follows

$$\frac{H(\boldsymbol{p})}{H(\underbrace{(1/n, 1/n, \ldots, 1/n)^\top}_{n \text{ times}})} = \frac{1}{\log(1/n)} \sum_{p_i} p_i \log(p_i)$$

where the highest entropy that is achieved with a uniform distribution is mapped/normalized to one.

Receiver operating characteristic (ROC) analysis:

The *ROC space* is the space that is spanned over the true positive rate on the y-axis and false positive rate on the x-axis. This enables comparing different classifiers within the ROC space according to different cost models, which specify some desired trade-off between the true positive and false positive rate. The upper left corner, i.e., point $(0, 1)$, represents a perfect classifier since the true positive rate is 100% while the false positive rate is 0% (Fawcett, 2004). Per definition, this analysis can only be applied to binary classification. However, one can consider a multi-class classification problem as a binary one when analyzing the performance regarding a particular class by referring to all samples in this class as positive and to all the other samples as negative instances (Provost and Domingos, 2003). Two possible application scenarios for considering the *ROC curve* are presented in the following.

The performance of a ranker (i.e., an ML model that produces a score for each instance) can be taken as a classifier by defining a threshold score above which all instances are classified as positive. This ends up in an individual classifier for each possible threshold, and the corresponding ROC curve of these classifiers can be used to determine an optimal threshold according to a cost model or further the quality of the instance ranking can be compared to other rankers. For instance, in areas where the ROC curve of ranker A is above the one of ranker B, the ranking quality of A regarding the cost model corresponding to this area is superior over the quality of B. Consequently, ROC analysis can be used to determine the ability in separating instances of a particular class form the others, e.g., by the associated pre-activation value in a neural network (alternatively, the probability of the output distribution can be used as well). Here, the considered class represents the positive and all other classes the negative as ROC analysis does only apply for binary classification. This analysis has the advantage of being independent of the other classes in a multi-class classification problem if they dominate the pre-activation of the investigated class too strongly so that it is only rarely predicted, which can occur, e.g., by learning on imbalanced training data (Hinton et al., 2015). This is because the model captures the prior associated with the imbalanced class distribution (Buda et al., 2018).

As presented as a special case of preference learning in Sec. 2.1.1, rankings of labels can be used for multi-label classification. Hence, the ROC curve can be used to examine the quality of the ranking of the labels produced for a given input instance. Therefore, the correct labels (according to the labels that are assigned to the input instance in the data set) are treated as positives and the remaining ones as negatives where a good ranking ranks all positives precede negatives. The ROC analysis can be used to find an appropriate threshold above which the labels should be predicted.

ROC Area under curve (AUC):

Considering the ROC analysis regarding the class separation quality of a ranker as described above, the *ROC Area under curve (AUC) score* reduces the ROC performance to a single score, which can be used to compare different rankers. Since the ROC AUC is a portion of the area of the unit square, its value ranges between zero and one and has further the statistical property that the ROC AUC is the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance (Fawcett, 2004).

Especially when a (convolutional) neural network is learned on imbalanced training data, Buda et al. (2018) suggested using the ROC AUC to evaluate the discriminating power for particular classes independently of the magnitudes of the pre-activation scores associated with the other classes.

For the case of solving multi-label classification with predicting label rankings, the ROC AUC can be used to evaluate the quality of the ranking and the suitability to split the set of labels in two subsets

either of the labels that should be predicted or these that should not (which is represented by a ROC AUC score of 1). Note that maximizing the ROC AUC is equivalent to minimizing the frequently used rank loss (Kotlowski et al., 2011; Rubin et al., 2011).

4.4.2 Statistical tests

If one wants to draw conclusions about the performance of an ML algorithm A compared to another algorithm B that are as general as possible (e.g., specifically here that SIR improves the test accuracy over a baseline), it is indispensable to test the performance of the algorithms on multiple data sets in order not to show the superiority on only one task a single data set represents. Webb (2000) warns that it is debatable whether comparing the mean performance scores across data sets is meaningful because the performance scores in different data sets cannot simply be regarded as directly comparable. For example, it is hard to tell whether an improvement of 95% to 96% accuracy on some data set is worth more (or not) than an improvement from 35% to 40% on another data set. Pair-wise comparisons of the achieved performance scores per data set would provide a tendency as to which of the two algorithms is superior. However, the question arises whether this tendency is sufficient enough to draw a reliable conclusion, or whether it is just a product of chance. In other words, it is demanded to keep the probability p that the performance differences between the algorithms are just a product of chance minimal when concluding. This is exactly where statistical testing gets involved.

Formally, consider n (test) data sets where the two algorithms are evaluated on to obtain a sequence of n independent paired performance scores (or also referred to as observations). The *null hypothesis* H_0 formulates the assumption that the paired observation sequences follow the same underlying distribution. By appliance of the statistical test, the probability p (also called significance value or p-value) is determined and quantifies how likely it is that H_0 is true or, conversely, that H_0 is false, and the observations follow distinct distributions (two-tailed testing), with a probability of $1-p$. If p is low, H_0 is rejected since H_0 seems very unlikely being true and the *alternative hypothesis* H_1 (stating that the observations have been drawn from distinct distributions in the two-tailed testing) is accepted. For that purpose, a significance level α is defined that serves as a threshold to reject H_0 if $p \leq \alpha$. Typical choices are $\alpha = 0.05$ or $\alpha = 0.01$ where the test outcome, i.e., that the distributions are different as formulated through H_1, is commonly concluded to be significant or highly significant, respectively. Most statistical tests produce an intermediate result, called the *test statistic*, which can be compared to the *critical values* (under a certain level of significance α) to show significance (under α) for what $p \leq \alpha$ is guaranteed without the need of explicitly calculating the p-value. Note that the critical value is also dependent on the number of observations. Obviously, it is desired to show that the proposed method, SIR, improves the neural networks being learned on significantly many data sets, but that does not imply that one would not care nor that it would not be of interest when SIR worsens the result on significantly many data sets. Therefore, two-tailed significance tests will be considered (instead of one-tailed), which are able to expose both worsening and improvements under statistical significance.

Loza Mencía (2013) pointed out that it quickly happens to draw wrong conclusions from statistical tests such as it is only possible to detect the presence but not the absence of statistical significance in statistical testing. This is because the determined p-value only considers the probability of making a *type 1 error* (i.e., the rejection of a true H_0). If no significance (under a specified significance level α) is obtained, the *type 2 error* occurs and refers to the probability that a false H_0 is not rejected (i.e., accepted). Generally, most tests have a relatively high type 2 error (depending on the *power/sensitivity* of the test) because the tests mainly focuses on the p-value (i.e., type 1 error) by design.

In the following, two statistical tests are presented which suit our needs in terms of assumptions that these entail, especially as these are non-parametric. Parametric statistical tests assume a particular type of distribution the observations do obey. Since we cannot (or do not want to) tell that the observations are distributed in a certain way, e.g., normally distributed observations would be implicitly assumed if the *paired t-test* is used, Demšar (2006) suggests the *sign test* and the *Wilcoxon signed-rank test*, both being non-parametric, for pair-wise comparisons of two ML algorithms over multiple data sets.

The only fact that marginally undermines the independence assumption of the observations on multiple data sets is that each pair of observations was evaluated on the same test set instead of using cross-validation or similar. However, due to the large size of these test sets any bias/dependence induced through evaluating with the same test data samples is considered marginal, and independence can still be assumed. See Table 4.1 for more details on the sizes of the (test) data sets that have been used.

Sign test:

The *sign test* is simple and popular to test for significance by quantifying over the number of data sets on which algorithm A performs better/worse than algorithm B. Simply, the sign test counts the signs of the pair-wise differences between the observations which can intuitively be seen as counting the number of successes (positive sign), failures (negative sign) and ties (difference is zero) across the data sets. Note that ties are ignored but reduce the sample size, i.e., number of observations. From a statistical perspective, the observed differences can be seen as realizations of a Bernoulli distributed random variable X (i.e., the possible outcomes are either zero (no success) or one (success)). The null hypothesis H_0 now formulates that $P(X \geq 0) = 0.5$, which yields the intuition that successes and failures do occur equally often. Consequently, the alternative hypothesis H_1 states the opposite so that $P(X \geq 0) \neq 0.5$. The sequence of observations forms a Bernoulli process whose probability of occurrence is described by a binomial distribution parametrized by the number of observations and a success probability of 0.5 (assuming H_0). The significance level α indicates the sum of the (equally large) areas under the mass function of the binomial distribution on both ends (in the two-tailed test). The sequence of observations must lie within one of these areas so that H_0 could be rejected under this significance level.

The notation of (successes / failures / ties) will be used in the evaluation of the experiments and the number of successes serves as the test statistic. It is obvious that this test is non-parametric and, overall, that there are no further assumptions about the actual values of the observed differences are being made. On the other hand, this leads to the fact that the test is very conservative, and significance can only be proven with very clear observations.

Wilcoxon signed-rank test:

The *Wilcoxon signed-rank test* (Wilcoxon, 1950) is like the sign test a non-parametric test but is more powerful which is reflected in the fact that a type 2 error is less likely and, thereby, H_0 can be rejected with a smaller number of successes (depending on the difference values) and with a smaller number of observations, which is a handy feature since the generation of observations that require the training of a neural network is time-consuming and computationally intensive.

The test is attained by incorporating the values of the pair-wise differences: a ranking is formed according to the absolute difference values in ascending order, two sums, one for the positive differences and one for the negative differences, are formed from the corresponding ranking position numbers, and the minimum of the two sums serves as test statistic where H_0 can be rejected if this value is lower than the critical value under the significance level α. Further, the p-value can be computed exactly (the formula is omitted for the sake of brevity) or approximated with a normal distribution, which is very inaccurate and not recommended, if the sample size is small (smaller than 20). Note that differences

of zero (ties) are being excluded from the ranking and differences of the same magnitude are usually assigned the average rank.

However, the above-motivated advantages over the sign test are accompanied by a disadvantage: the performance score differences are assumed to be commensurable to a certain extent in order to form the ranking, but this assumption is still weaker than assuming normal distributions like in the paired t-test, for example. Nevertheless, the sign test is preferred over the Wilcoxon signed-rank test and, therefore, performed first. When the sign test does already indicate statistical significance, the Wilcoxon signed-rank test becomes superfluous since the certainty of rejecting H_0 with the sign test is as great as a rejection by the Wilcoxon signed-rank test because the type 1 error is guaranteed to be smaller than the required significance level α for both (Mittenecker, 1979, ch. 11b, p. 168), but no assumptions have to be made regarding the comparability of the differences between the observed performance scores.

5 Experimental results

5.1 Regularization capability

In order to be able to make statements about the regularization capability of SIR as well-founded as possible, the results were compared across several (visual) data sets. This variety of data sets makes it reasonable to draw overall conclusions and statements about properties of SIR that would be transferable to any other (visual) data sets.

In the following, it is verified that the method of SIR is indeed a regularization method (directly supporting Hyp. I; Regularization) with partially statistical significance according to the sign tests in different experimental configurations over multiple data sets (or relying on the Wilcoxon signed-rank test result whenever the null hypothesis cannot be rejected with the sign test). The statistical tests are applied on the accuracy scores the learned neural networks achieve on the test set.

Regularizing behavior can manifest itself in several ways, depending on the reason for which the unregularized model encounters problems to generalize properly. As mentioned in Sec. 2.1.4, the generalization (of the model) depends on (a) the model, (b) the learning algorithm and (c) the training data. Since (b), the learning algorithm, is the feature that is examined by comparing a neural network learned with SIR to an unregularized neural network, the other two determinants, namely the properties of (a) the model and (c) the training data, can be varied to control the difficulty of learning and, thus, the need for an adequate regularization method. Regarding (a) the model, the neural network architectures `LeNet5` and `ConvNet` are put to use where the `LeNet5` is a relatively small model and, therefore, should not be able to drastically overfit on the idiosyncrasies of the training data by design but should be able to (only) roughly capture the general concept of the data (i.e., the task that is represented by the data). The `ConvNet`, on the contrary, is of a high capacity, which might entail strong overfitting. Regarding (c) the training data, instead of only using the full portion of available training data, additional experiments on a small subset (i.e., 10%) of the training data samples were conducted. Since the (visual) data sets are very large, overfitting is often not too much of an impediment to learning if the capacity of the model is appropriate. In summary, several experimental configurations are present, some of which are more prone to overfitting than others.

While presenting the outcomes of the statistical tests (see Table 5.1), the neural networks using either the `LeNet5` or `ConvNet` architecture are referred to as small and large models respectively, as well as a full training data set is termed as large data set whereas the subset (i.e., a portion of 10%) of the training data set is termed as small data set. The outcomes of the statistical tests are presented in Table 5.1 while the raw performance scores (in terms of the test accuracy) and the pair-wise differences, which were used for statistical testing, can be found in Table A.1 in the appendix.

It has to be underlined that the fact that in most configurations the positive results or even statistical significance have been achieved without tuning the hyperparameters per configuration or much more per data set is very astonishing and remarkable.

Configuration		Sign test		Wilcoxon signed-rank test	
Data	**Arch**	***p*-value**	**Test statistic**	***p*-value**	**Test statistic**
10%	LeNet5	**0.03906** *	**(8/1/0)**	0.07422	7
10%	ConvNet	0.07031	(7/1/1)	**0.02086** *	**1**
100%	LeNet5	1.00000	(5/4/0)	0.16406	10
100%	ConvNet	1.00000	(4/4/1)	0.44121	12

Table 5.1.: Results from the statistical tests for different experimental configurations over multiple data sets. Self-imitation regularized neural networks are compared to unregularized neural networks. * denotes statistical significance according to the respective statistical test under a significance level of $\alpha = 5\%$.

Small data sets:

On the small data set variants, self-imitation regularized neural networks improved on significantly (under a significance level of $\alpha = 5\%$) many data sets compared to the unregularized neural networks according to the sign test for the `LeNet5` and to the Wilcoxon signed-rank test for the `ConvNet` (see the first two rows in Table 5.1).

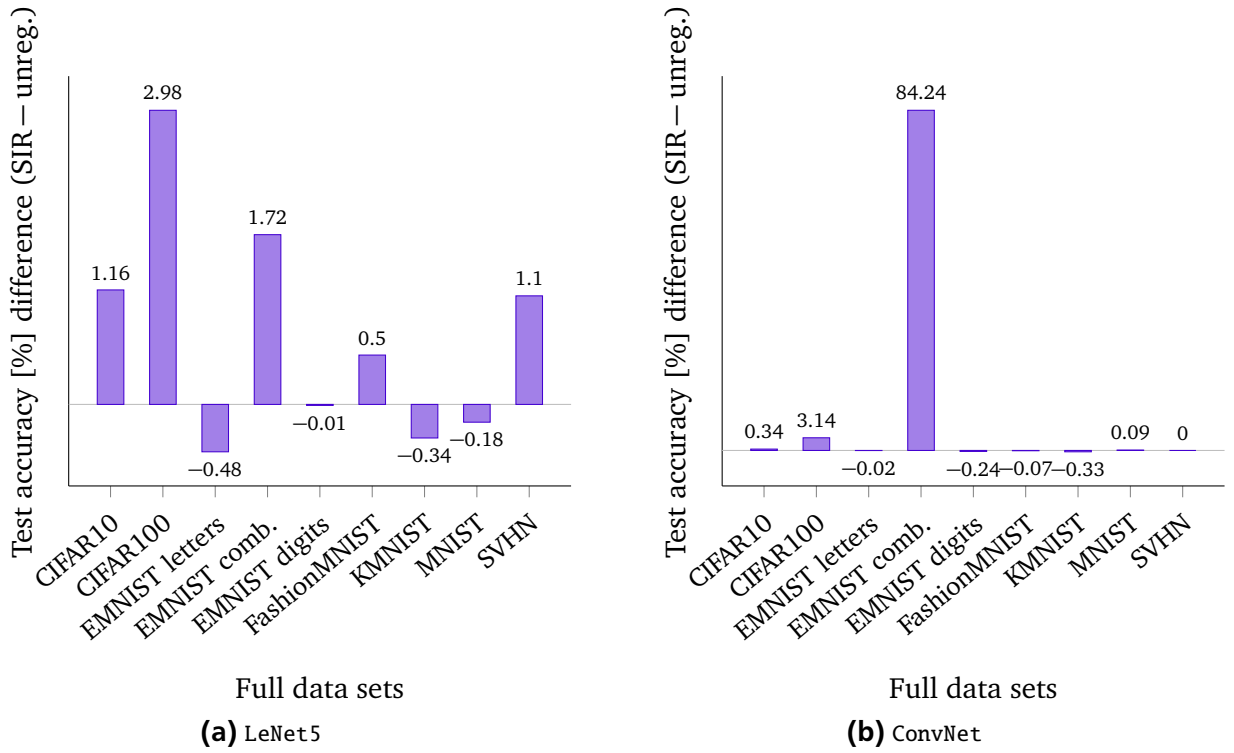
Therefore, it can be concluded that applying SIR will improve the learning of an (unregularized) neural network that is trained on a small set of data with similar characteristics to those of the used visual data sets. Moreover, without the need to tune the hyperparameters that are newly introduced by the appliance of SIR since the very same SIR hyperparameter setting was used in all the experiments considered for the statistical testing.

With regard to the architecture, i.e., the high difference in capacities between the `LeNet5` and `ConvNet` architecture, one can conclude that SIR achieves an improvement independently of the neural network's capacity on small data sets.

Large data sets:

On large data sets, we cannot say that the results achieved with self-regularized neural networks (with the very same hyperparameter setting as used in the case of small data sets) and the unregularized ones (significantly) differ according to the results of the sign test, as well as the Wilcoxon signed-rank test (see the last two rows in Table 5.1). This outcome seems to be without a doubt since SIR has only improved the performance over the unregularized neural network after the learning was finished on about half of the data sets. As the size of the data sets is the only difference to the experiment evaluations from above, one can infer that SIR has good guarantees in regularizing a neural network (independently of its capacity) when the amount of training data is low but does not affect the learning much when there is a lot of training data is available. So, this overall supports the hypothesis that SIR increases the data efficiency (Hyp. V; Data efficiency) since less information is available to learn from if the amount of data is low. The regularizing behavior of SIR on small data sets could be further attributed to the qualitative enrichment of the data targets by incorporating the soft targets (even if these targets are self-predictions) as it was assumed in Hyp. I (Regularization).

On the other hand, it cannot be claimed that SIR is not able to regularize neural networks on large data sets because if the actual differences per data set in terms of test accuracy between the neural network with SIR applied and without are considered, it is strikingly apparent that the declines against the unregularized baseline are comparatively negligible whereas the improvements are much more pronounced. This can be seen in the test accuracy differences plotted in Fig. 5.1 for the `LeNet5` (Fig. 5.1a) and `ConvNet` (Fig. 5.1b) architecture. Especially for the `LeNet5` architecture, the improvements are higher



(a) `LeNet5` **(b)** `ConvNet`

Figure 5.1.: Differences in test accuracy on multiple data sets between self-imitation regularized neural networks compared to unregularized ones for the **(a)** `LeNet5` and **(b)** `ConvNet` architecture.

than any decline in terms of magnitudes whereas, in the case of the `ConvNet` on the EMNIST (combined) data set, SIR enables the neural network to learn anything at all. More about the findings that SIR can contribute to stabilization of the training is presented in Sec. 5.5.

Overall, this analysis makes it clear why the applied statistical tests on this test accuracy differences do not prove any significance and why (in the case of the Wilcoxon signed-rank test) only minor tendencies could be shown. In the case of the sign test, only the sign of the differences is considered, distinguishing into improvement or deterioration. Although this property was initially desired and motivated the use of the (non-parametric) sign test since no assumptions about the comparability of accuracy values across multiple data sets are being made, these very one-sided results (i.e., greater improvement than worsening effects) would argue in favor of SIR. However, these tendencies were not clear enough to make the Wilcoxon signed-rank test showing significance on this (small) number of observations.

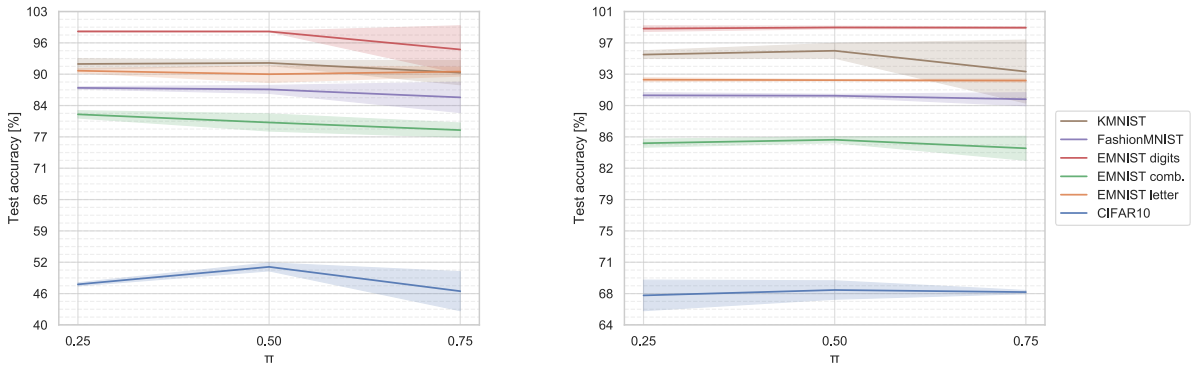
Further SIR hyperparameter settings:

The results shown above on a single hyperparameter setting are not intended to give the impression that SIR only works with these hyperparameters. Therefore, the results of neural networks learned on another hyperparameter setting of SIR, namely that the target gradients are now suppressed and a lower imitation rate $\pi = 0.25$ is used, over the same experimental configurations as above are shown in Table 5.2 while the raw performance scores (in terms of the test accuracy) and the pair-wise differences, which were used for statistical testing, can be found in Table A.2 in the appendix. The results are even more positive and significant than the ones obtained with the previous SIR hyperparameter setting.

However, one should note that since two SIR hyperparameter settings are used for statistical testing and are both compared to the same (unregularized) baseline, i.e., the same sequence of observations, the problem of multiple comparisons is entailed, though to a very small extent as it involves only two com-

Configuration		Sign test		Wilcoxon signed-rank test	
Data	**Arch**	***p*-value**	**Test statistic**	***p*-value**	**Test statistic**
10%	LeNet5	**0.03906** *	**(8/1/0)**	0.07422	7
10%	ConvNet	**0.00781** **	**(8/0/1)**	0.01427 *	0
100%	LeNet5	**0.03906** *	**(8/1/0)**	0.01953 *	3
100%	ConvNet	1.00000	(4/3/2)	0.79985	12

Table 5.2.: Results from further statistical tests with a different SIR hyperparameter setting for different experimental configurations over multiple data sets. Self-imitation regularized neural networks are compared to unregularized neural networks. * and ** denote statistical significance according to the respective statistical test under a significance level of $\alpha = 5\%$ and $\alpha = 1\%$, respectively.



(a) 10% train data **(b)** 100% train data

Figure 5.2.: Mean test accuracy (with standard error bands) achieved with the `ConvNet` architecture on the data sets KMNIST, EMNIST (letters, combined, digits), Fashion-MNIST and CIFAR-10 for either **(a)** 10% or **(b)** 100% of the training data. Averaged over different SIR hyperparameter settings. Note that the error bands for some data sets at $\pi = 0.75$ are actually wider and were down-scaled for the sake of clarity since the SIR configuration using symmetric standard softmax imitation does not managed to learn in these cases.

parisons with the baseline. Nevertheless, one could consider to use the Bonferroni correction (Dunnett, 1964), which is recommended for comparing multiple machine learning algorithms (Salzberg, 1997; Benavoli et al., 2016), and is a straightforward but conservative method to counteract the multiple comparison problem by multiplying the p-values by the number of multiple comparisons (i.e., two) or conversely, dividing the significance level α by this. According to the additionally presented results of the Wilcoxon signed-rank tests, the significance under the Bonferroni correction holds in most cases where the null hypothesis H_0 could not be longer rejected with the sign test.

However, the intention of these further statistical tests is to show that other hyperparameter settings of SIR work and by examining the test accuracy differences between the hyperparameter settings (compare the differences between Table A.1 and Table A.2) one can see that these are more pronounced in the hyperparameter setting evaluated before (target gradients and $\pi = 0.5$). This fact can, among other things, be attributed to the higher imitation rate π, which is investigated in the following in more detail.

Imitation rate π:

In investigating the effects in relation to the imitation rate π further, Fig. 5.2 shows that SIR with $\pi = 0.25$ works similarly well as $\pi = 0.5$ on average, but has a much smaller variance across most data sets. Furthermore, $\pi = 0.75$ is not recommended because the average performance does not increase compared to $\pi \in \{0.25, 0.5\}$, and the variance is considerably increased on average since the SIR configuration using symmetric standard softmax imitation (i.e., $\widehat{\boldsymbol{\sigma}}(\cdot) = \widetilde{\boldsymbol{\sigma}}(\cdot) = \boldsymbol{\sigma}(\cdot)$; with target gradients) does not manage to learn on the (small variants of the) data sets EMNIST letter and digits (in Fig. 5.2a), and the (full) KMNIST data set (in Fig. 5.2b). On the other hand, this could also only argue against the stability of symmetric SIR because such drastic problems were not observed when asymmetric SIR is applied in combination with $\pi = 0.75$. In conclusion, one can say that the effects of SIR are more pronounced if a higher imitation rate π is used but at a higher risk, at the same time.

5.1.1 Regularization ability with other regularizers being present

With regard to Hyp. IV (Complementary to standard regularization methods), the complementary of SIR and other standard regularization methods is investigated over multiple visual data sets (i.e., all the ones used above and defined in Table 4.1) and on the four experimental configurations, again, concerning the capacity of the neural network as well as the data set size. Analogous to the isolated analysis of SIR above, statistical tests, the sign test and the Wilcoxon signed-rank test, are carried out on this results comparing the performance scores (in terms of test accuracy) of the neural network regularized with a standard regularization method in combination with SIR to the neural network regularized with the standard regularizer alone.

SIR is used in the same hyperparameter settings as above, namely with the asymmetric softmax imitation $\widehat{\zeta} = 2$ and $\widetilde{\zeta} = 1$, and either with target gradients at an imitation rate of $\pi = 0.5$ or without at $\pi = 0.25$. The outcomes of the statistical tests are presented in Table 5.3. In more detail, the outcomes for the SIR with target gradients and $\pi = 0.5$ are aggregated in Table 5.3a whereas the raw performance scores can be found in Table A.3 (L^1-norm reg.), Table A.4 (L^2-norm reg.), Table A.5 (maxnorm) and Table A.6 (dropout) in the appendix. The outcomes for SIR without target gradients and $\pi = 0.25$ are collected in Table 5.3b whereas the raw performance scores can be found in Table A.7 (L^1-norm reg.), Table A.8 (L^2-norm reg.), Table A.9 (maxnorm), and Table A.10 (dropout) in the appendix.

In contrast to the isolated analysis of SIR, where a single hyperparameter setting allowed very clear conclusions to be made across different configurations over multiple data sets, the results of the statistical tests here are not that clear and the test statistics (successes and failures in the test statistic of the sign test) show less obvious tendencies. Possibly the combination of SIR and standard regularizers also requires a finer tuning of the hyperparameters per experimental configuration or even per data set. Thus the beneficial property of SIR that it does not require much tuning effort would no longer exist in combination with other regularization methods. However, it remains unclear whether an appropriate tuning of the standard regularization method would already be sufficient while the same hyperparameters can be used for SIR as in the previous investigations where only SIR was applied.

Apart from the fact that the outcomes of the statistical tests are not clear over the different configurations and does not allow for profoundly general conclusions, some slight tendencies become apparent and are discussed in the following. Overall, SIR seems to be most profitable to be combined with maxnorm regularization where within no experimental configuration the addition of SIR worsened the performance on more data sets than it could improve. Moreover, the addition of SIR without target gradients improved the `LeNet5` architecture trained on the full data set on all data sets (highly significantly many under the significance level $\alpha = 1\%$ according to the sign test), and the p-values (at least from

Configuration			Sign test		Wilcoxon signed-rank test	
Data	**Arch**	**Standard regularizer**	***p*-value**	**Test statistic**	***p*-value**	**Test statistic**
10%	LeNet5	L^1 λ=0.00005	1.00000	(4/5/0)	0.91016	21
10%	ConvNet	L^1 λ=0.00001	0.28906	(2/6/1)	**0.04232** *	**3**
100%	LeNet5	L^1 λ=0.00005	1.00000	(4/5/0)	0.91016	21
100%	ConvNet	L^1 λ=0.00001	1.00000	(4/4/1)	0.83363	16
10%	LeNet5	L^2 λ=0.005	1.00000	(4/5/0)	0.49609	16
10%	ConvNet	L^2 λ=0.005	0.06250	(0/5/4)	0.05906	0
100%	LeNet5	L^2 λ=0.005	**0.00391** **	**(0/9/0)**	0.00391 **	0
100%	ConvNet	L^2 λ=0.005	0.06250	(0/5/4)	0.05906	0
10%	LeNet5	Maxnorm c=3.5	0.50781	(6/3/0)	0.25000	12
10%	ConvNet	Maxnorm c=3.5	0.72656	(5/3/1)	0.62406	14
100%	LeNet5	Maxnorm c=3.5	0.17969	(7/2/0)	0.05469	6
100%	ConvNet	Maxnorm c=3.5	0.28906	(6/2/1)	0.23395	9
10%	LeNet5	Dropout p=0.5	0.50781	(6/3/0)	0.49609	16
10%	ConvNet	Dropout p=1.0	0.17969	(7/2/0)	0.49609	16
100%	LeNet5	Dropout p=0.5	0.17969	(2/7/0)	0.35938	14
100%	ConvNet	Dropout p=1.0	0.50781	(6/3/0)	0.30078	13

(a) Target gradients and $\pi = 0.5$

Configuration			Sign test		Wilcoxon signed-rank test	
Data	**Arch**	**Standard regularizer**	***p*-value**	**Test statistic**	***p*-value**	**Test statistic**
10%	LeNet5	L^1 λ=0.00005	0.50781	(6/3/0)	0.65234	18
10%	ConvNet	L^1 λ=0.00001	0.72656	(3/5/1)	0.52861	13
100%	LeNet5	L^1 λ=0.00005	1.00000	(5/4/0)	0.49609	16
100%	ConvNet	L^1 λ=0.00001	0.28906	(2/6/1)	0.36273	11
10%	LeNet5	L^2 λ=0.005	0.50781	(6/3/0)	0.65234	18
10%	ConvNet	L^2 λ=0.005	0.37500	(4/1/4)	0.10565	1
100%	LeNet5	L^2 λ=0.005	**0.00391** **	**(0/9/0)**	0.00909 **	0
100%	ConvNet	L^2 λ=0.005	1.00000	(2/3/4)	0.41849	4
10%	LeNet5	Maxnorm c=3.5	0.28906	(6/2/1)	0.29362	10
10%	ConvNet	Maxnorm c=3.5	0.72656	(5/3/1)	0.36273	11
100%	LeNet5	Maxnorm c=3.5	**0.00391** **	**(9/0/0)**	0.00391 **	0
100%	ConvNet	Maxnorm c=3.5	1.00000	(4/4/1)	0.72629	15
10%	LeNet5	Dropout p=0.5	1.00000	(5/4/0)	0.59363	17.5
10%	ConvNet	Dropout p=1.0	0.50781	(6/3/0)	0.42578	15
100%	LeNet5	Dropout p=0.5	0.50781	(3/6/0)	0.59363	17.5
100%	ConvNet	Dropout p=1.0	0.17969	(7/2/0)	0.07422	7

(b) No target gradients and $\pi = 0.25$

Table 5.3.: Results from the statistical tests for different experimental configurations over multiple data sets. Neural networks with SIR (either **(a)** with target gradients and $\pi = 0.5$ or **(b)** without target gradients and $\pi = 0.25$) and a standard regularizer are compared to a neural network regularized solely with this standard regularizer. * and ** denote statistical significance according to the respective statistical test under a significance level of $\alpha = 5\%$ and $\alpha = 1\%$, respectively.

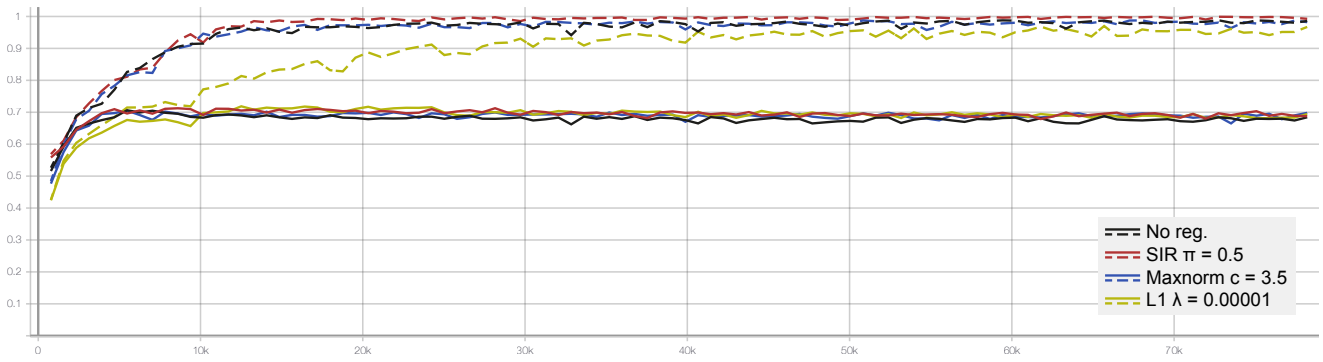


Figure 5.3.: Learning curves of test accuracy (solid line) and train accuracy (dashed line) for the `ConvNet` architecture on the CIFAR-10 data set. Asymmetric SIR was applied with $\widehat{\zeta} = 2$, $\widetilde{\zeta} = 1$ and target gradients.

the Wilcoxon signed-rank test) are almost below 0.05 if the target gradients were used. The combination of SIR and dropout does also seem relatively beneficial since the performance is mostly improved while a clear number of data sets for which the addition of SIR degrades the performance can be only found in the configuration with the `LeNet5` architecture on the full data sets. Overall the norm penalty regularization methods, L^1-norm and especially L^2-norm regularization does not seem to be profitably combinable with SIR. On the `ConvNet` architecture, the performance was worsened on most data sets when combining L^1-norm regularization with SIR while the addition of SIR to L^2-norm regularization was only able to achieve improvements on the small data set configurations for one SIR hyperparameter setting (no target gradients and $\pi = 0.25$) while in most cases no improvements or further declines partially on even highly significantly (under the significance level $\alpha = 1\%$ according to the sign test) many data sets occurred.

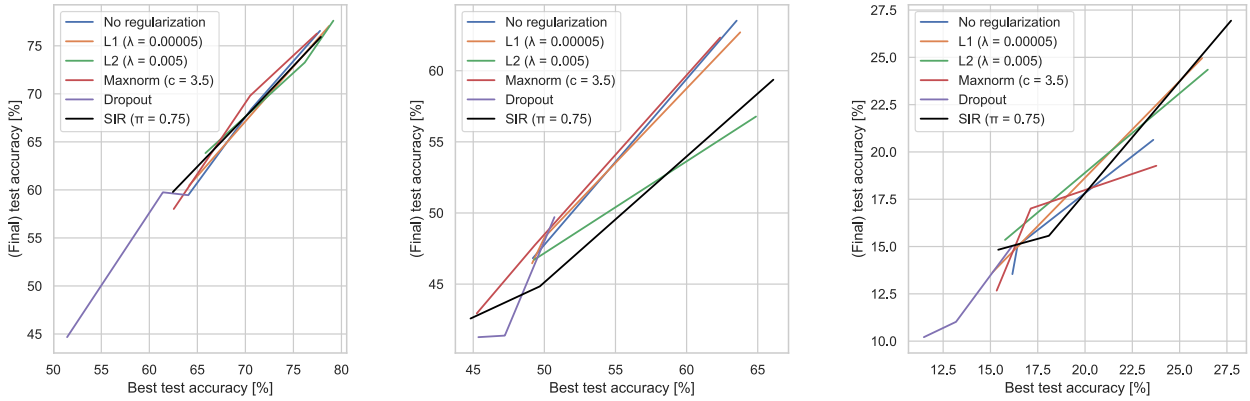
In conclusion, Hyp. IV (Complementary to standard regularization methods) cannot be solidly supported by this investigation because only the maxnorm regularization and dropout showed slight tendencies that a combination with SIR is beneficial while, in most cases, a combination with a norm-penalty regularization method failed. However, it should be noted that according to the raw performance scores in Sec. A.1 in the appendix, the regularization effect of SIR alone in terms of test accuracy is higher on many data sets than the one of applying a standard regularization method alone especially for the experimental configurations where the training data samples were reduced to 10%.

5.1.2 Overfitting in the later training course

Although SIR regularizes and can contribute to generalization significantly, as shown above, it cannot completely eliminate overfitting in the later course of training. This means that the overfitting phase (as explained in Sec. 2.1.4) can still occur.

For example, learning curves of test and train accuracy are shown in Fig. 5.3 of a self-imitation regularized and an unregularized neural network, as well as of neural networks where standard regularization methods were applied. Here, one can observe the presence of a slight overfitting phase after about the best accuracy was reached (after around 10,000 iterations). However, this phenomenon also occurs in the settings with standard regularization methods, which is why SIR cannot be claimed to be inferior to these standard regularizers regarding this problem.

In order to more thoroughly examine the (later) overfitting of SIR, the final accuracy (after finishing the training) is plotted over the best accuracy in Fig. 5.4 for neural networks trained on (subsets of)



(a) CIFAR-10, 100% training data **(b)** CIFAR-10, 10% training data **(c)** CIFAR-100, 10% training data

Figure 5.4.: Different neural network architectures trained on (subsets of) either CIFAR-10 and CIFAR-100 with SIR and standard regularization methods. Asymmetric SIR was applied with $\widetilde{\zeta} = 2$, $\widetilde{\zeta} = 1$ and target gradients. Each point represents the best accuracy (x-axis) and the final accuracy (y-axis) of a single neural network.

either CIFAR-10 or CIFAR-100 with SIR as well as standard regularizers. This visualizes the drop of accuracy from the best performing model within the course of training to the final model after 100 episodes by the difference between the value on the x-axis and the y-axis for a point (representing the performance of a neural network). In the plots in Fig. 5.4a and Fig. 5.4c one can observe the line representing SIR being similar to the lines of the standard regularizers for what reason the drop of accuracy and the impact of later overfitting in SIR is existent but of a similar extent compared to the standard regularizers on the full CIFAR-10 and CIFAR-100 (this plot is omitted) data sets, as well as on the 10% subset of the CIFAR-100 data set. In contrast, this phenomenon of later overfitting occurs much more pronounced in the plot in Fig. 5.4b where the final accuracy is more than 5% lower than the one of the best model for some models trained with SIR while most (except the L^2 regularized) neural networks barely show any difference.

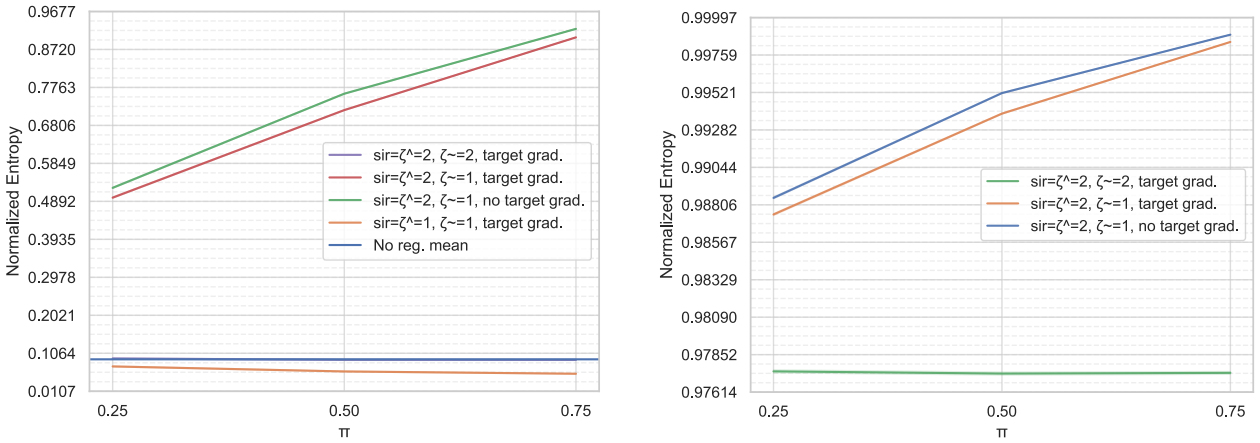
Overall, it can be said that SIR cannot prevent the overfitting phase in the later course of training at all and might tend to be more susceptible to this compared to standard regularizers according to the experimental results on the (subsets of) CIFAR-10 and CIFAR-100.

5.2 On the functioning of Self-Imitation Regularization (SIR)

Based on the experimental results, this section outlines and verifies the principal functioning of SIR. The effects of SIR on the predictions (and thus the soft targets used in for the self-imitation), on the one hand, and on the learned parameters $\boldsymbol{\theta}$ within the neural network, on the other hand, are investigated and interpreted.

5.2.1 Predictions and soft targets investigation

Obviously, SIR depends entirely on the soft targets that are used in the imitation loss (Eq. 3.1), and these, in turn, are based on the original prediction of the neural network up to a softening modification (i.e., a chained softmax function). For that reason, the characteristics of the predictions (and soft targets) being made by a neural network that was learned under the appliance of SIR in comparison to the predictions



(a) Neural network's (standard) softmax predictions

(b) (2-chain) softmax soft targets for imitation loss

Figure 5.5.: Mean normalized entropy of the distributions, **(a)** $\boldsymbol{\sigma}(\boldsymbol{z})$ and **(b)** $\boldsymbol{\sigma}^{(2)}(\boldsymbol{z})$, when querying the learned neural networks (for both the `LeNet5` and `ConvNet` architecture) on the test data set samples from CIFAR-10, CIFAR-100, MNIST and KMNIST. The entropy normalization makes the values comparable over multiple data sets with a different number of classes. Note that standard error bands are plotted but are very narrow and, therefore, barely visible.

of unregularized neural networks are examined, which provides meaningful clues on the regularization behavior of SIR.

For this purpose, the entropy of the neural network prediction (distribution) serves as a rough quality measure by telling whether the distribution is rather hard (low entropy) or soft (high entropy). A soft distribution is considered to carry much information because, in addition to the predicted class (i.e., the one with the highest probability), it also quantifies about the other classes. In the context of the self-imitation, a distinction must be made between the properties (i.e., entropy) of the (standard softmax) prediction of the neural network and the soft targets used in the imitation loss since these are different whenever asymmetric softmax imitation is used as the soft targets are a modification of the (soft) predictions. In Fig. 5.5, the mean entropy of the neural networks' soft predictions (Fig. 5.5a) as well as the soft targets (if they differ from the soft predictions; Fig. 5.5b) are shown in terms of different SIR hyperparameter settings over different values of π, as well as in comparison with an unregularized neural network.

It becomes strikingly apparent that the choice between symmetric and asymmetric imitation has a fundamental impact on the properties of the neural network's prediction (distribution). That was rather unexpected since, in the preliminary experiments with KD, no differences between symmetric and asymmetric imitation were observed, and it was rather considered as a trick to get non-equal distributions in the cross entropy in order to obtain non-zero gradients (induced by the prediction $\widetilde{\boldsymbol{\sigma}}(\boldsymbol{z})$) in the imitation loss (Eq. 3.1). As Fig. 5.5a shows, the predictions of neural networks using symmetric self-imitation are equal to or even lower in entropy than the predictions of the unregularized networks (see the orange and purple curves in comparison to the blue baseline in Fig. 5.5a). On the other hand, predictions of neural networks trained with asymmetric self-imitation show high or even very high entropy (red and green curves in Fig. 5.5a). This behavior seems to be generally valid since it was averaged over all test data samples of several data sets (CIFAR-10, CIFAR-100, MNIST and KMNIST) for different neural network architectures (`LeNet5` and `ConvNet`) and shows hardly any variance. Furthermore, this behavior can also be observed when evaluating the models on the training data as well as when the interim version of the

neural networks, when the highest test accuracy was achieved over the course of the training, is being evaluated. As expected, the effects regarding the entropy increase for all SIR settings when the imitation rate π was increased.

When considering the effect of the 2-chain softmax distributions in Fig. 5.5b, which are used as soft targets in the asymmetric imitation settings, the characteristics observed in Fig. 5.5a (standard softmax) remain (relative) unchanged, but the size of the entropy values has risen substantially. Such high entropy measurements point undoubtedly to an almost uniformly distributed probability distribution. This fact was also unexpected because whenever the soft targets are strongly uniformly distributed, there is a risk that it may lead to a loss of information regarding both the probabilities revealing the dark knowledge and the predicted class associated (and quantified in terms of confidence) with the highest probability. For a freshly initialized neural network, the same high entropy is observed when 2-chain softmax targets are produced since the neural network's pre-activations are distributed close to zero.

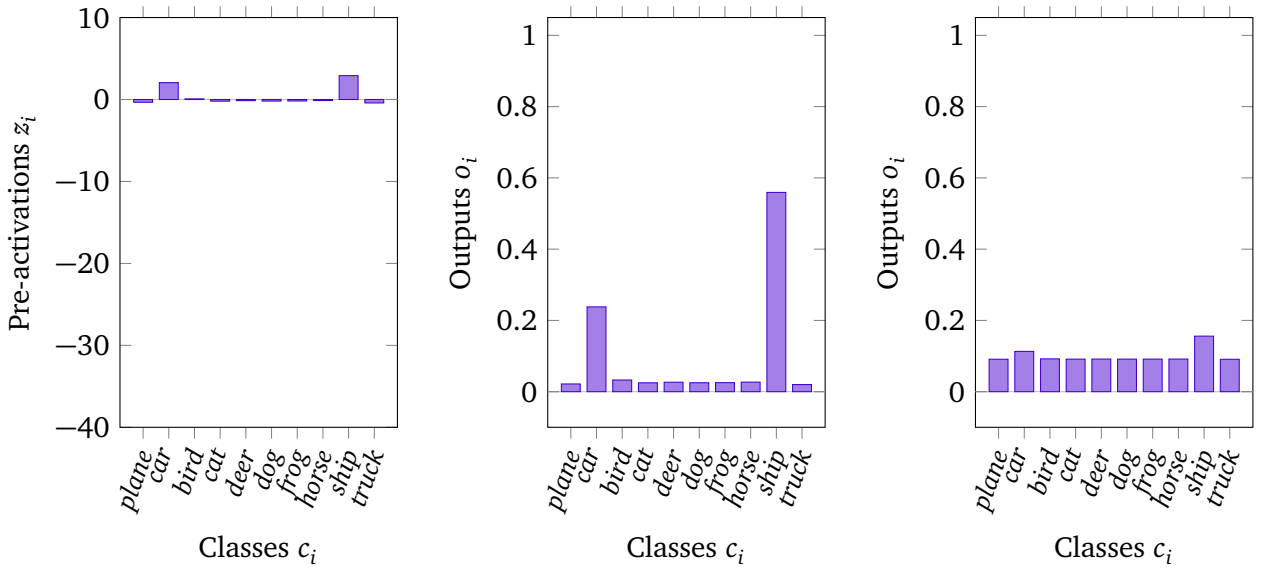
In order to pursue this further, Fig. 5.6 shows the entirely different distribution characteristics of the prediction (and correspondingly produced soft targets) being made by an asymmetric self-imitation regularized neural network (first row, Fig. 5.6a, Fig. 5.6b and Fig. 5.6c) in comparison to an unregularized one (second row, Fig. 5.6d, Fig. 5.6e and Fig. 5.6f) when evaluating the learned models (both of the `ConvNet` architecture) on the CIFAR-10 test sample shown in Fig. 5.6g, which is labeled as *ship*.

The (asymmetric) self-imitation regularized neural network's pre-activations are all close to zero, with the values for the classes, which the neural network considers to be possibly correct, standing out comparatively clearly. That leads to a relatively soft softmax distribution (Fig. 5.6b; compared to the one of the unregularized neural network in Fig. 5.6e). In sharp contrast, since the pre-activations of the unregularized neural network are of a high magnitude with high differences between the values z_i and are, further, strongly negative for the classes the neural network is certain about to be incorrect, the softmax distribution is hard (Fig. 5.6e). The two-chain softmax target (Fig. 5.6c) that were used for learning in the SIR set-up, is very soft due to applying the softmax twice but, nonetheless, still carry the information about the two classes, *ship* and *car*, the neural network was originally certain about (regarding to the pre-activations $\boldsymbol{z}$ and the softmax output $\widetilde{\boldsymbol{\sigma}}(\boldsymbol{z})$). If a 2-chain softmax target had been produced in the unregularized setting (Fig. 5.6f), any information about classes the neural network also considers (*cat* has the least difference to the highest pre-activation of *ship* in Fig. 5.6d) would have been completely lost due to the hard distribution that is already obtained after the first appliance of the softmax function (Fig. 5.6e).

Clearly, an unregularized neural network, which minimizes the data loss (Eq. 3.2) only, does not learn to predict soft (softmax) distributions as the (data) loss function implicitly favors such that are mostly hard and low in their entropy (as seen in Fig. 5.5a) as the loss is minimal if the hard data targets can be matched almost exactly. Therefore, pre-activations of high magnitude and strongly diverging z_i are produced to meet this demand.

When using symmetric softmax imitation, the imitation loss (Eq. 3.1) can also be construed as an entropy penalty in addition to the data loss (Eq. 3.2). As the data loss itself already prefers softmax distributions of low entropy, as just argued, the symmetric softmax imitation further (and explicitly) encourages the neural network in learning hard prediction distributions (i.e., low entropy) by inducing such entropy penalty in the loss. Thus, more extreme pre-activations are produced.

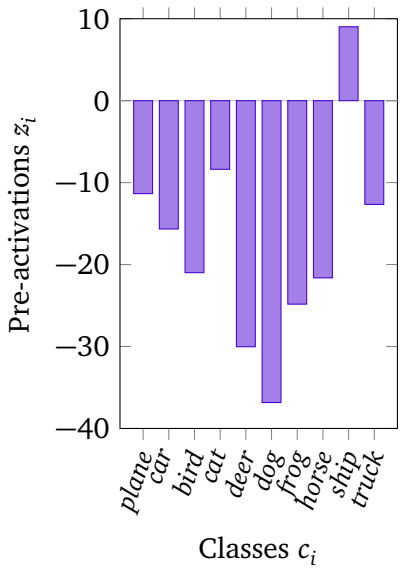
In the case of SIR using asymmetric softmax imitation, the imitation loss (Eq. 3.1) strives for an opposite goal than the data loss (Eq. 3.2). Small pre-activations will result in softer softmax distributions and reduce the imitation loss (with softened targets – formally, an asymmetry of $\widehat{\zeta} > \widetilde{\zeta}$, which is the case being considered in this thesis). Further, it can be assumed that the prediction distribution does not degenerate to a uniform distribution since salient probabilities for classes the neural network considers



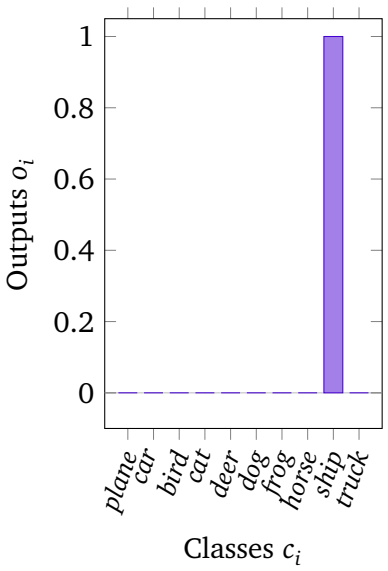
(a) SIR pre-activations z_i

(b) SIR softmax outputs $o_i = \sigma_i(\boldsymbol{z})$

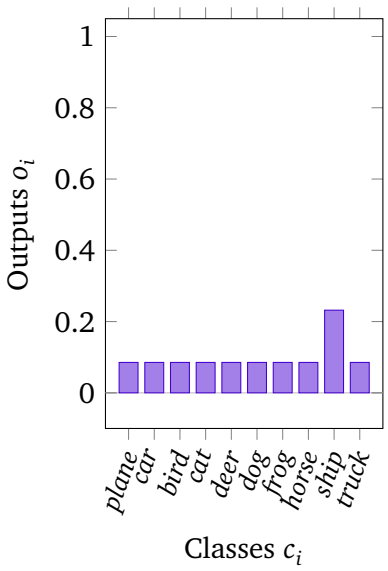
(c) SIR 2-chain softmax outputs $o_i = \sigma_i(\sigma(\boldsymbol{z}))$



(d) Unreg. neural network's pre-activations z_i



(e) Unreg. neural network's softmax outputs $o_i = \sigma_i(\boldsymbol{z})$



(f) Unreg. neural network's 2-chain softmax outputs $o_i = \sigma_i(\sigma(\boldsymbol{z}))$

(g) Predicted test sample

Figure 5.6.: Exemplary prediction distributions of two learned neural networks either **(a)** - **(c)** with using SIR while learning or **(d)** - **(f)** without regularization on **(g)** a CIFAR-10 test sample (labeled as *ship*). The hyperparameter setting $\widehat{\zeta} = 2$, $\widetilde{\zeta} = 1$, $\pi = 0.25$ with target gradients was used for training the self-imitation regularized `ConvNet` neural network.

to be possibly correct, are preserved in order to reduce the data loss (Eq. 3.2), on the one hand. This characteristic of prediction distributions (observed in asymmetric SIR) approaches an equilibrium in minimizing the multi-objective loss function L^{SIR} that SIR composes (Eq. 3.3 through the weighted average of the imitation (Eq. 3.1) and data loss (Eq. 3.2). On the other hand, it could be possible that classes that are also considered likely to be correct besides the predicted class, are preserved in the soft targets (i.e., two-chain softmax distributions) due to the target gradients (which were allowed while training the self-imitation regularized neural network that produced the results in the first row in Fig. 5.6). It was already hypothesized in the beginning that this might be the case because the target gradients point towards a "teaching behavior" that helps the student to reduce the (imitation) loss. From a mathematical perspective, the target gradients would favor a lower entropy if these gradients can be considered to be similar to the ones in the symmetric SIR set-up.

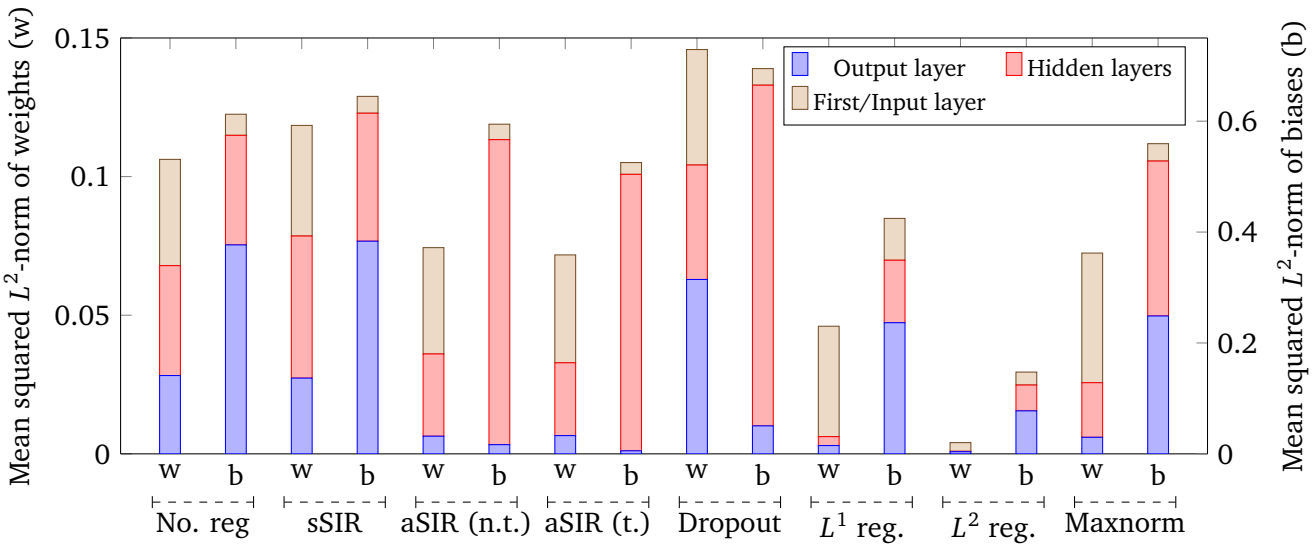
Overall, such soft softmax predictions are only possible when the differences between the pre-activation values z_i are very small. Exactly for the reason that the produced pre-activations show very small differences and are of a small size, it can be assumed that the neural network must have learned small weights (at least in the output layer) since it was regularized by SIR in order to meet the requirements of a soft output distribution (i.e., to minimize the imitation loss). The consequence of this regularization is that the generalization of the neural network improves since small weights generally contribute to generalization. This fact, in particular, motivates the use of parameter norm penalties, as explained in Sec. 2.7.1, where the regularization strategy drives the weights closer to the origin (Goodfellow et al., 2016, ch. 7, p. 227). That small weights support generalization is because overfitting gets more difficult, on the one hand, since strong and characteristic values in the weights are needed in order to capture specific input structures (e.g., which correspond to the idiosyncrasies of the training data or even memorization of certain training samples). On the other hand, small weights are less sensitive against noise and variations in the inputs since such noise is not amplified and propagated through the neural network much which is why the prediction behavior keeps to be stable on slightly different or unseen data (i.e., invariant under one or more transformations of the input variables (Bishop, 2006, ch. 5.5.3, p. 261)).

5.2.2 Parameters investigation

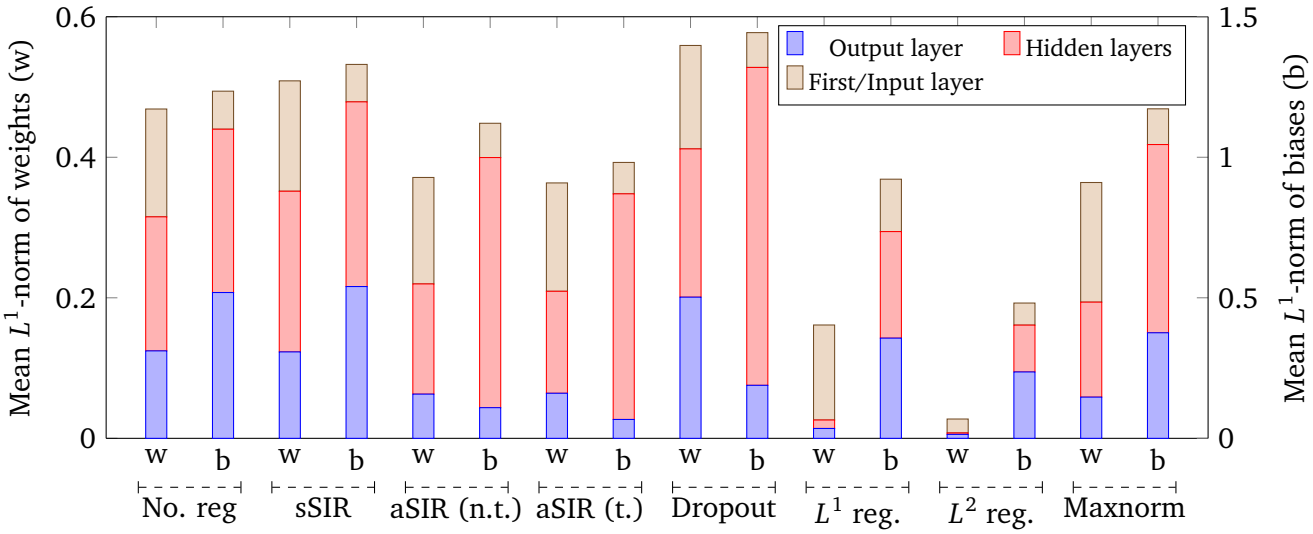
The specific properties of the parameters $\boldsymbol{\theta}$ being learned with SIR applied are investigated further in the following to support the statements claimed above. It is also of interest to what extent the learned parameters differ from those learned using other standard regularizers. From this, conclusions can be drawn about which standard regularization methods are similar to SIR and which can possibly be combined fruitfully (i.e., show complementary mechanisms).

For that purpose, the mean magnitudes of the weights $\boldsymbol{w}(\boldsymbol{\theta})$ and biases $\boldsymbol{b}(\boldsymbol{\theta})$ (per layer) are presented in Fig. 5.7. On the one hand, the use of the (squared) L^2-norm $\|\boldsymbol{\theta}\|_2^2$ as a metric (Fig. 5.7a) should show whether the parameters $\boldsymbol{\theta}$ take extreme values or are rather small and diffusely distributed. On the other hand, with the L^1-norm $\|\boldsymbol{\theta}\|_1$ as metric (Fig. 5.7b), the sparseness of the parameters $\boldsymbol{\theta}$ should be shown (i.e., the smaller the more sparse). Note that these norms are selected to show such properties in the parameters $\boldsymbol{\theta}$ as particularly that properties are aimed to be adopted when utilizing these norms in norm-penalty regularization, i.e., L^2 and L^2 regularization (see Sec. 2.7.1).

However, we should be careful that the L^1-norm does not necessarily give reliable insights into sparsity because a vector that is sparse but has large values unequal to 0 can still have a higher L^1-norm length than a diffuse vector. For instance, consider $\boldsymbol{v}_1 = (0, 2, 0, 0, 0)^\top$ and $\boldsymbol{v}_2 = (0.5, 0.5, 0.5, 0.5, 0.5)^\top$ where



(a) Mean squared L^2-norm of the learned parameters θ



(b) Mean L^1-norm of the learned parameters θ

Figure 5.7.: Mean lengths (according to either the **(a)** L^2-norm or **(b)** L^1-norm) of vectors that are being constructed by splitting the learned parameters θ (on the CIFAR-10 data set with the `ConvNet` architecture) by weights $w(\theta)$ (abbreviated with w) and biases $b(\theta)$ (abbreviated with b), and further by the type of layer (input, hidden, and output layers). The mean is computed so that for each layer type the average length per parameter is represented . Note that First/Input layer refers to the first layer containing parameters in the neural network. The hyperparameters were set as follows: sSIR: $\widehat{\zeta} = 1$, $\widetilde{\zeta} = 1$ with target gradients; aSIR (n.t.): $\widehat{\zeta} = 2$, $\widetilde{\zeta} = 1$, no target gradients; aSIR (t.): $\widehat{\zeta} = 2$, $\widetilde{\zeta} = 1$ with target gradients; Dropout was applied architecture specific; L^1 reg.: $\lambda = 0.00001$; L^2 reg.: $\lambda = 0.005$; Maxnorm: $c = 3.5$.

$\boldsymbol{v}_1$ is obviously (more) sparse, yet $\|\boldsymbol{v}_1\|_1 = 2 < 2.5 = \|\boldsymbol{v}_2\|_1$. Furthermore, Hurley and Rickard (2009) have shown that the L^1-norm does not capture all criteria of sparsity.

First of all, the conclusion from above that due to the small pre-activations and soft softmax distributions, the output parameters become smaller (in their magnitude) when asymmetric SIR is applied can be confirmed by considering the squared L^2 lengths in comparison to the unregularized neural network. Furthermore, since the reduction according to the squared L^2 metric is even more pronounced than the reduction according to the L^1 metric (in comparison to the unregularized neural network), very small (i.e., zero centered) parameter values with absolute values smaller than 1 can be deduced since then the squared L^2-norm provides smaller results than the L^1 norm, i.e., $a^2 < |a|$ applies for arbitrary $-1 < a < 1$. Thus, most weights $\boldsymbol{w}(\boldsymbol{\theta})$ in the output layer modulate the input attenuatingly.

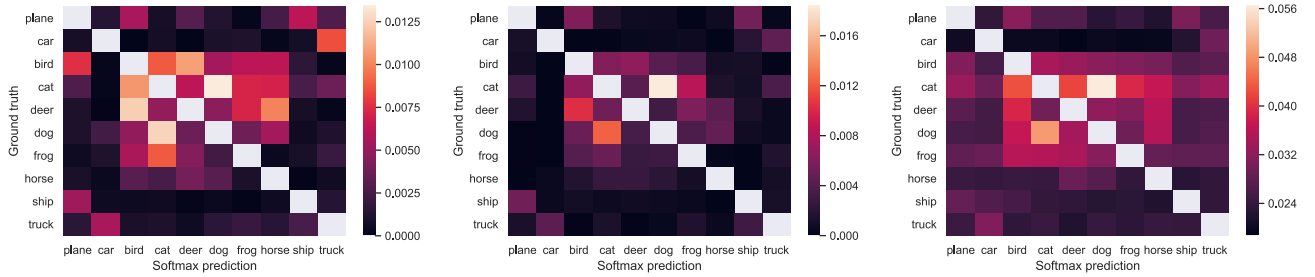
According to both squared L^2 and L^1 lengths, the weights $\boldsymbol{w}(\boldsymbol{\theta})$ in the input (i.e., the first layer in the neural network that contains parameters) and hidden layers were only marginally affected by (asymmetric) SIR which is in strong contrast to L^2 regularization where the size of the weights $\boldsymbol{w}(\boldsymbol{\theta})$ in all layers was substantially decreased. Further, the same applies to the L^1 regularization in terms of the hidden (and the output) layers. Overall, the way in which all weights $\boldsymbol{w}(\boldsymbol{\theta})$ were regularized here is most similar to the Maxnorm regularization.

It is strikingly apparent that the properties of the biases $\boldsymbol{b}(\boldsymbol{\theta})$ learned when asymmetric SIR was applied differ strongly from those of the unregularized (or symmetrically self-regularized) neural network as per the L^2-norm measures in Fig. 5.7 because of the remarkably high magnitude in the hidden layers besides being low in the output layer as discussed before. This amplifies the (output) activations of the hidden neurons independent of the actual input signals. It is unclear why SIR is calling for such behavior, but up to a minor difference in the scale, the characteristics of the biases $\boldsymbol{b}(\boldsymbol{\theta})$ are highly similar to the ones of the biases when dropout was put in while training. In addition to the asymmetric SIR, dropout is the only regularization method that produces biases $\boldsymbol{b}(\boldsymbol{\theta})$ in the hidden neurons that are of such a high magnitude.

Regarding the weights $\boldsymbol{w}(\boldsymbol{\theta})$ in all layers of the neural network, the two investigated asymmetric SIR settings in Fig. 5.7 – with the difference that the target gradients were once allowed and once not – do not show any differences but do so with respect to the biases $\boldsymbol{b}(\boldsymbol{\theta})$. According to both norms, the biases $\boldsymbol{b}(\boldsymbol{\theta})$ in the learned neural network where SIR was applied with activated target gradients, are of smaller magnitude compared to the ones where the target gradient flow was inhibited in the backpropagation.

In terms of symmetric SIR, this investigation of the learned parameters $\boldsymbol{\theta}$ does not reveal obvious insights that would enable conclusions to be drawn about possible regularization behavior besides that the magnitude of the parameters $\boldsymbol{\theta}$ increased slightly compared to the unregularized neural network, which was to be assumed due to the more diverging pre-activations z_i and harder softmax distributions (i.e., higher entropy).

In conclusion: on the one hand (asymmetric) SIR causes small-valued and diffuse parameter vectors $\boldsymbol{\theta}$ (especially in the output layer) which is generally assumed to contribute generalization (as it is the purpose for some standard regularization methods). On the other hand, SIR effectuates the parameter vector $\boldsymbol{\theta}$ in terms of the lengths (as shown in Fig. 5.7 and extensively discussed) in a way which is similar to the parameter vectors obtained when utilizing some other standard regularization methods. Both facts highly support Hyp. I (Regularization) about the regularization ability of SIR. In contrast, due to the fact that there is no similarity to some other standard regularization methods (in terms of the parameter vector lengths $\|\boldsymbol{\theta}\|$) or at least no similarity in all examined aspects to all standard regularizers, it can be assumed that the operating mechanism of SIR being complementary to that of those standard regularizers and that they can thus be combined profitably. This supports Hyp. IV (Complementary to standard regularization methods).



(a) Unreg. neural network **(b)** Symmetric SIR **(c)** Asymmetric SIR

Figure 5.8.: Average softmax distributions (over the x-axis) visualized as heat maps per class (on the y-axis) on the CIFAR-10 test data set. The neural networks were either trained **(a)** unregularized, **(b)** with symmetric SIR or **(c)** with asymmetric SIR.

In Sec. 5.3 - Sec. 5.7, the above-made statements about the regularization ability of SIR are further substantiated in the experimental results, and certain regularization properties of SIR are highlighted in discussing the results of problem-oriented experiments (such as those incorporating incorrect training data labels).

5.2.3 (Ranking) quality of the soft targets/predictions

As the soft targets are suspected as the major reason why SIR works by providing further information in addition to the (hard) data targets, this section discusses the properties and quality of the soft targets in the context of SIR.

Similarity information in the soft predictions:

Soft targets/predictions reveal the dark knowledge in the non-maximum probabilities, i.e., the probabilities of the classes that are not predicted. More specifically, information about the similarities and correlations between the classes are encoded in these probabilities. As already presented in Fig. 5.6, one can notice that the prediction probability distribution (i.e., softmax distribution of the pre-activations) of an unregularized neural network is very hard (see Fig. 5.6e), which is why such information about the other classes is not apparent as these probabilities are all near-zero. In contrast, SIR produces softer distributions (see Fig. 5.6b) that tend to contain more information. As the consideration in Fig. 5.6e is only based on a single CIFAR-10 test sample (see Fig. 5.6g), this examination is extended to all test data samples in order to investigate the general quality of the similarities and correlations between the classes for differently self-imitation regularized ($\pi = 0.25$ and either symmetrically with $\zeta = 1$ or asymmetrically with $\widehat{\zeta} = 2$ and $\widetilde{\zeta} = 1$) as well as unregularized neural networks.

The matrices/heat maps in Fig. 5.8 present the average softmax distribution (over the x-axis) per class (ground truth on the y-axis) with the `ConvNet` architecture on the CIFAR-10 test data set. Note that this visualization is not equal to confusion matrix visualizations since it does only contain the softmax distributions of correctly predicted samples because it is assumed that the quality of the similarities and correlations might not be reliable if not even the correct class could be predicted. This decision is justifiable since all considered neural networks have roughly the same test accuracy, and thus the evaluation takes place on a similar number of test data samples. Furthermore, the probabilities for the predicted/correct classes are ignored (i.e., light gray) for visualization reasons because otherwise, due to the high probability values for the correct classes, the colors of the heat maps would not be

well distinguishable for all other probabilities any more. Note that these ignored probabilities range between 95% and 99% on average for the predictions of the unregularized neural network as well as the symmetrically self-imitation regularized neural network, but between 66% and 81% when asymmetric SIR was applied.

Again, the fundamental difference between asymmetric and symmetric SIR becomes apparent in the softmax distributions being softer (see Fig. 5.8c) and harder (see Fig. 5.8b), respectively, than the ones of an unregularized neural network (see Fig. 5.8a). The hardness can be seen in the sparseness of probabilities that are not zero (i.e., not almost black). The unregularized neural network is able to express mostly meaningful general similarities but the ones of the symmetrically self-imitation regularized neural network seem slightly better concerning the quality since the sparse non-zero probabilities highlight valuable similarities. For example, symmetric SIR learned that general similarities for the class *plane* are *bird* and *ship* (first row in Fig. 5.8b), which seems meaningful because a bird and a plane are both able to fly while on images showing either a ship or a plane the color blue might dominate. However, the unregularized neural network does also point out slight similarities to the classes *deer* and *horse* (first row in Fig. 5.8a), which does not seem meaningful at all. In sharp contrast, the asymmetrically self-imitation regularized neural network produces non-zero probabilities for all classes (at least more than 2%) while the highest probabilities showing the general similarities are about 5%. Although the similarities therefore seem to be very uninformative, the difference between the highest similarity probabilities to the lowest ones is equal to the differences in the unregularized and symmetrically self-imitation regularized neural networks. Nevertheless, only considering the similarity information in the asymmetric SIR case associated with probabilities of more than 2.5%, this seems the least informative and clear since it is very diffuse but sparsity would be more meaningful here. Considering the average softmax prediction for the class *plane* again (first row in Fig. 5.8c), besides the main similarities to the classes *bird* and *ship*, similarities to the classes *cat*, *deer* and *frog* are learned with asymmetric SIR and do not seem justifiable. Therefore, the similarities are less clear than in the unregularized neural network's (average) predictions and even less so than in the symmetric SIR case.

The expression of similarities as in the symmetric SIR case (Fig. 5.8b) seems to be preferable over the other ones because these are the most sparse. Intuitively, sparse information on similarities is most akin to the way a person would express such similarity information while explaining something by focusing on the essentials to maintain high informational content without confusing the counterpart. For example, it would not make sense to enumerate all mammals when explaining to a child what a zebra is because they have slight similarities as they feed their cubs on milk. Instead, it would be more effective to limit themselves to fundamental characteristics, such as the high similarity to a horse except for the coat pattern. As symmetric SIR acts as an entropy penalty, hard distributions are preferred, which could be sparse, but it might also occur that these are near one-hot encoded predictions where all information about similarities is lost. Therefore, a trade-off between diffusely predicting similarities to all other classes and predicting no similarities (i.e., solely almost 100% to the predicted class) has to be found in order to obtain sparse and meaningful prediction distributions.

Ranking quality of the soft predictions:

To evaluate the ranking quality of the similarities and soft targets/predictions on a more thorough and measurable basis, `ConvNet` neural networks are considered in the same three settings (i.e., no regularization, symmetric SIR and asymmetric SIR) as before but on the CIFAR-100 data set. As explained in Sec. 4.2, the CIFAR-100 has 100 classes, which can be categorized into 20 super classes that express similarities so that classes in the same super class are more similar to each other than classes from different super classes. This serves a notion of whether a produced ranking for a given input image (according to

Arch.	Regularization	(Sub) class performance		Super class performance		
		Accuracy	Position err.	Accuracy	Position err.	ROC AUC
LeNet5	None	27.90%	10.34%	40.52%	4.74%	75.08%
LeNet5	Asym. SIR	28.94%	11.33%	41.03%	4.81%	72.20%
LeNet5	Sym. SIR	28.13%	10.10%	41.56%	4.69%	75.27%
ConvNet	None	28.94%	12.31%	41.00%	4.92%	73.21%
ConvNet	Asym. SIR	29.86%	20.20%	41.42%	5.46%	60.10%
ConvNet	Sym. SIR	29.38%	11.69%	41.91%	4.80%	73.57%

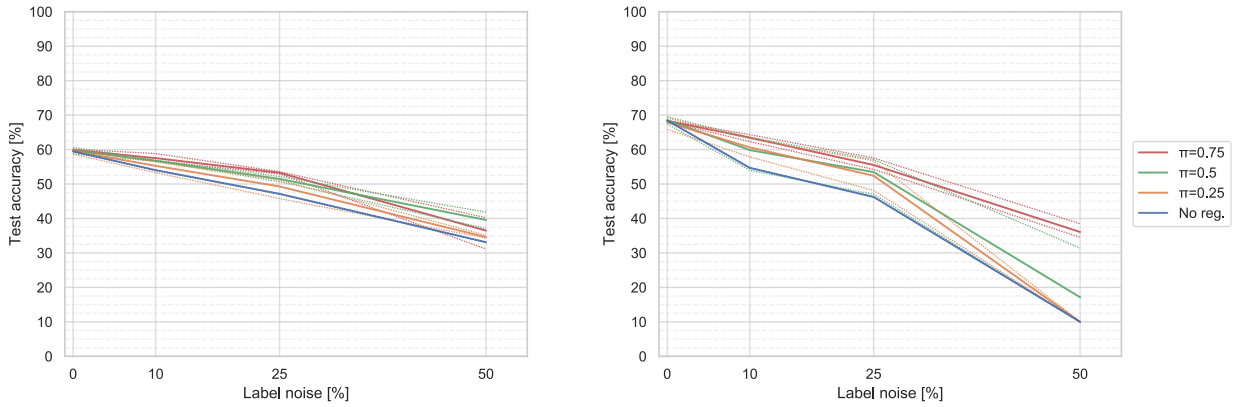
Table 5.4.: Evaluation metrics regarding the ranking quality of the soft predictions on the CIFAR-100 test set. Namely the accuracy and (normalized) position error on (sub) class basis as well as the accuracy, (normalized) position error and ROC AUC on super class basis. The best scores per architecture are printed in green whereas the worst are printed in red.

the probabilities or pre-activations assigned to each of the 100 classes) is meaningful since classes that are in the same super classes as the correct class is (according to the data labels) should be ranked above classes that are in another super class. For example, given an image showing an *apple*, the ranking is of a higher quality if classes from the super class *fruits and vegetables* are ranked at the top (got the highest probabilities assigned).

In addition to the standard test accuracy on the (sub) classes, Table 5.4 shows evaluations according to metrics measuring the ranking quality of the soft predictions on the test set with regard to the (sub) classes (i.e., normalized position error) as well as, in particular, to the super classes (accuracy, normalized position error, ROC AUC). The normalized position error is adjusted for the super classes in a way that it considers the highest rank some (sub) class obtains that is in the same super class as the data label. As described above, it is preferable that (sub) classes in the super class in which the data label is are ranked high what is expressed by the ROC AUC as it can be applied on multi-label classification via preference learning as described in Sec. 4.4.1. Here, all (sub) classes in this super class are labeled as positive and the remaining ones as negative.

The results are very clear and consistent with all previous insights that SIR generally improves the test accuracy (regarding the (sub) classes) where asymmetric SIR outperforms symmetric SIR whereas symmetric SIR is considered to produce more informational predictions (in terms of class similarities) than the unregularized neural network and especially than asymmetric SIR. The latter is expressed in the fact that symmetric SIR achieves the best scores according to all ranking evaluating metrics and asymmetric SIR the worst ones (except the super class accuracy if this is considered as a ranking evaluation measurement). That producing informational soft targets can indeed be advantageous, e.g., as symmetric SIR effectuates, is expressed in two facts: on the one hand, whenever a (test) sample is misclassified the correct label might also get a high probability assigned (according to symmetric SIR, which achieved the best (sub) class position errors). On the other hand, a wrongly predicted (sub) class is very similar to the correct (sub) class, which can be seen in symmetric SIR achieving the highest accuracy on the super classes (i.e., the wrongly predicted (sub) class is in the same super class as the correct class).

Finally, the fact that symmetric SIR is able to improve the (ranking) quality of the predictions over an unregularized neural network and consequently the quality of the soft targets used for self-imitation, this qualitative enrichment could support learning and might give a hint that Hyp. III (Learn about a class through information about other classes) might be correct in the case of symmetric SIR.



(a) `LeNet5` on CIFAR-10 **(b)** `ConvNet` on CIFAR-10

Figure 5.9.: Test accuracy of several learned neural networks either of the **(a)** `LeNet5` or **(b)** `ConvNet` architecture, which were regularized by SIR to different extends, plotted to different degrees of label noise, which were held fixed throughout training. The unregularized baseline (model) is depicted by the blue curve. When multiple hyperparameter settings are used for a regularization method, the mean performance is plotted whereas the highest and lowest performances that were achieved are plotted with thin dotted lines.

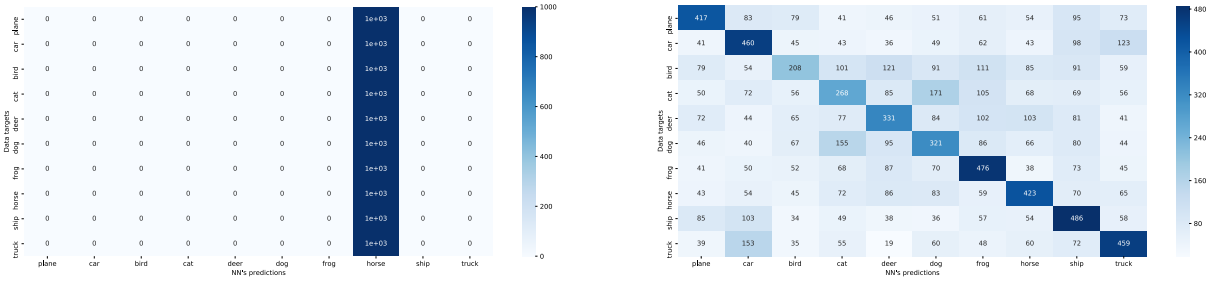
5.3 Noisy data label compensation

The experiments where the neural networks were trained on a training data set with partly noisy data labels (as specified in Sec. 4.1.1) were intended to show the ability of SIR in compensating errors in training labels in the first place. Therefore, the following discussion of the experimental results is based on experiments conducted on noisy data labels and serve to support Hyp. II (Compensate for erroneous data labels).

The ability to compensate errors can be further broken down into two core properties. First, into the robustness against such incorrect labels so that learning is influenced only to a small extent by them. This represents the intuition of the neural network is questioning the data labels "before accepting the label" by connecting this with the already acquired knowledge. Second, into the pure correcting ability of the noisy training labels when letting the (partly) learned network predict such a training instance. Being good in correcting those labels from early on in training might be especially important to achieve a superior regularizing ability and, consequently, prevent overfitting on the incorrect information in the SIR setting since the predictions are reused as "teaching" imitation targets (in the imitation loss in Eq. 3.1).

5.3.1 Robust learning despite noisy labels

In order to investigate the influence of noisy labels on the learning process when using SIR, the final test accuracy is analyzed. The question arises whether regularizing a neural network via SIR increases the robustness and improves the learning compared to the unregularized neural network can undoubtedly be affirmed according to the final test accuracy based on the experimental results of multivariate data sets from preliminary experiments and various modern CNN architectures applied on the CIFAR-10 data set, which is presented in this section.

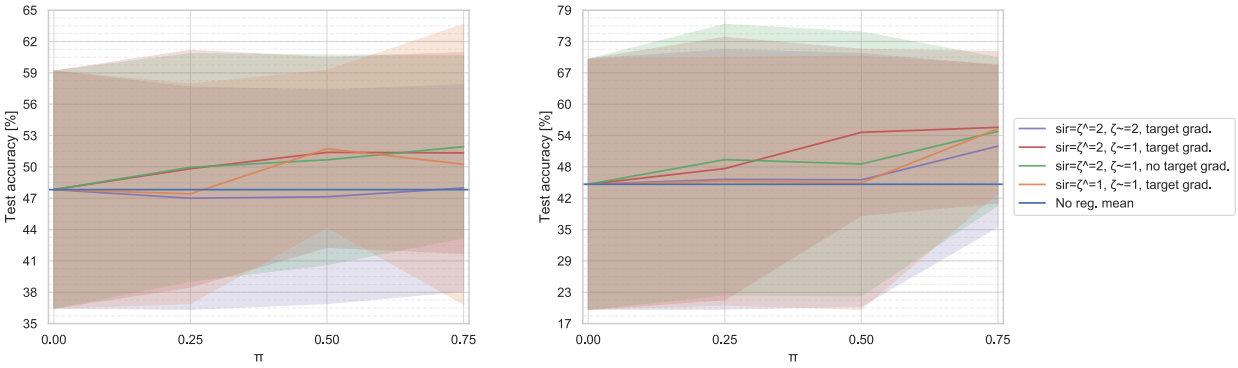


(a) Unregularized `ConvNet` **(b)** `ConvNet` with SIR ($\pi = 0.75$) applied

Figure 5.10.: Confusion matrices of the evaluation of an **(a)** unregularized and **(b)** self-imitation regularized `ConvNet` on the CIFAR-10 test data set where the neural networks were trained with 50% (training) label noise.

Fig. 5.9 shows the reduced loss of accuracy of self-imitation regularized neural networks compared to unregularized ones when label noise was applied to the training data. The SIR hyperparameter settings of the neural networks that were used to create the plots are two asymmetric softmax settings with softened imitation targets ($\widehat{\zeta} = 2$) and either allowing or inhibiting target gradient flow, and one symmetric setting with the standard softmax function (entropy penalty). By distinguishing the final test accuracy scores by different imitation rates $\pi \in \{0.25, 0.5, 0.75\}$ one can observe that a higher imitation rate effectuates a higher regularizing effect, reflected in an improved generalization, in most training cases with label noise applied for the two neural network architectures `LeNet5` and `ConvNet`. That SIR is not very susceptible to overfitting on the noisy data labels can be seen in the fact that the improvement in test accuracy achieved with SIR is even more striking on the high capacity `ConvNet` architecture (see Fig. 5.9b) since high capacity models are generally more prone to overfitting (as described in Sec. 2.1.4). Moreover, this enables the `ConvNet` neural network to sensibly learn despite highly noisy training data labels (i.e., label noise of 50%). In contrast, the unregularized neural network is not able to learn anything and only achieves accuracy of 10%, which is equal to random guessing or always predicting the same class in the ten class classification problem that the CIFAR-10 data set represents. The actual behavior of this learned unregularized `ConvNet` is always predicting the same class (i.e., class *horse*) on both the training and the test set as it can be seen in the confusion matrix in Fig. 5.10a (evaluated on the test set). To the opposite, SIR drastically improves the learning of the `ConvNet` on the CIFAR-10 training data with label noise of 50% by achieving a final test accuracy of almost 40% when, e.g., using an imitation rate of $\pi = 0.75$ and symmetric standard softmax imitation (i.e., $\tau = \zeta = 1$) while maintaining all other hyperparameters as in the unregularized training setting. This improvement is also strikingly apparent when comparing the test set confusion matrix in Fig. 5.10b with the one of the unregularized neural network (Fig. 5.10a).

For a more in-depth investigation of how the hyperparameter settings in SIR do affect the learning on data with label noise, the accuracy scores are averaged over the different degrees of label noise (10%, 25% and 50%) and plotted over the imitation rate π separately per SIR hyperparameter setting (see Fig. 5.11). The results of an additional symmetric softening softmax SIR hyperparameter setting is considered to allow for a meaningful interpretation without disregarding the dependencies of symmetry and softening softmax functions. Thus, two symmetric softmax imitation settings with either standard softmax ($\zeta = 1$) or a two-chain softmax ($\zeta = 2$) as well as two asymmetric ones with a two-chain softmax imitation target ($\widehat{\zeta} = 2$ but $\widetilde{\zeta} = 1$) where once the target gradient flow is allowed are considered.



(a) `LeNet5` on CIFAR-10 with label noise

(b) `ConvNet` on CIFAR-10 with label noise

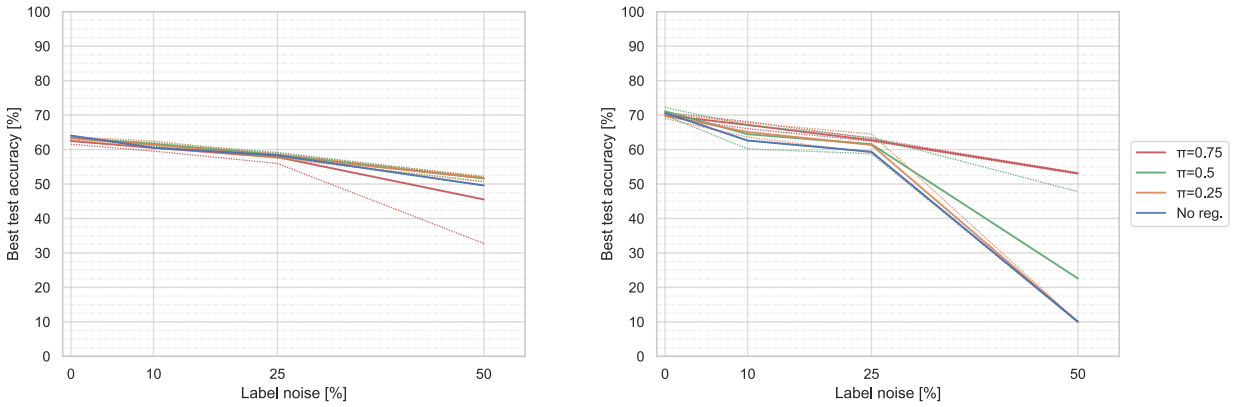
Figure 5.11.: Mean test accuracy (to different degrees of label noise) with standard error bands of the test accuracy of several learned neural networks, which were regularized by SIR to different degrees, plotted over the imitation rate π. The blue line depicts the unregularized baseline.

Again, as it is already apparent in Fig. 5.9, the intensity of the regularization is higher for a higher imitation rate π in most settings.

However, it becomes apparent in this plot that symmetric SIR and the softened symmetric imitation (i.e., $\zeta = 2$, the purple curves) is inferior to the asymmetric SIR settings and in the case of the `LeNet5` architecture (5.9a) frequently even worse than or equal to the mean of the unregularized neural network (blue horizontal line). Although the symmetric standard softmax imitation ($\zeta = 1$, the orange curve) performs better for high imitation rates ($\pi \in \{0.5, 0.75\}$) in the `LeNet5` architecture (5.9a), we cannot overall conclude that this setting is substantially superior to the symmetric softened imitation ($\zeta = 2$, the purple curves) since apart from these cases the results obtained with both settings are similar. Therefore, it can be inferred that the asymmetric imitation is more robust against noisy labels.

Considering the two asymmetric settings further, it can be observed that both work well and are robust against the label noise on average. However, the target gradients (red curves compared to the orange curves) induce a more steady proportionality to the imitation rate π, on the one hand, as the (red) curve is monotonically increasing for both architectures, `LeNet5` and `ConvNet`. On the other hand, the target gradients SIR setting seems to be more reliable in achieving improvements since the variance (visualized by the standard error bands) over the training results to various degrees of label noise is lower for most imitation rates π. Regarding the choice of π, the raw experiment results of the `LeNet5` (see Table A.11) and `ConvNet` architecture (see Table A.12) indicate the tendency that when setting $\pi = 0.25$ SIR without target gradients (nearly) outperforms SIR with target gradients and the opposite is the case when setting $\pi = 0.5$. However, such superiority of one setting is not clear for $\pi = 0.75$ because SIR without target gradients outperforms SIR with target gradients only on the `LeNet5` but vice versa regarding the `ConvNet` architecture. Further experiments on different architectures did not reveal any evident insights in this matter. Nevertheless, SIR with target gradients is slightly preferable according to the previous arguments.

Finally, it is worthwhile mentioning that SIR is not able to completely overcome the occurrence of the overfitting phase (as explained in Sec. 2.1.4) if training takes places on noisy labels since the final performance is almost always lower than the highest accuracy achieved by an interim model over the course of training. This is, however, also the case for unregularized neural networks, as shown for the `LeNet5` and `ConvNet` architectures in Fig. 5.12. The difference between the best accuracy and the final accuracy is most notable in the case of `ConvNet` (compare Fig. 5.12b to Fig. 5.9b), which is expected due to the high capacity of this architecture. However, this in no way detracts from the arguments



(a) `LeNet5` on CIFAR-10 **(b)** `ConvNet` on CIFAR-10

Figure 5.12.: Best accuracy achieved on the test set by an interim model over the course of training where either the **(a)** `LeNet5` or **(b)** `ConvNet` architecture, which were regularized by SIR to different degrees, were trained to different degrees of label noise.

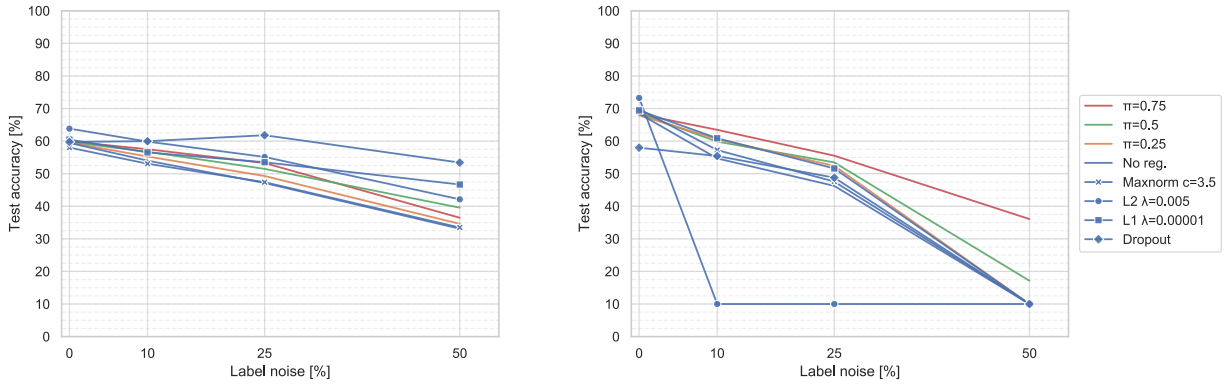
that SIR prevents overfitting as presented in the previous examinations since self-imitation regularized neural network overfits less than an unregularized. Note that no further examinations based on this best accuracy metric are made since this is not reliable and fragile to outliers that can arise if an interim neural network performs well on the test set only by chance. In order to smooth out such effects of randomness, the best model should be picked according to an additional data set (commonly referred to as validation set), which is independent of the training as well as the test set but equally distributed. Then the test set can be acquired to measure the performance of this picked model to make subsequent comparisons between the models fairer. For our purpose, measuring the interim performances on the test set is sufficient to show that the shape of the (test performance) learning curve is rough of concave form as the test curve in the conceptional visualization in Fig. 2.2 and thus demonstrates the presence of the overfitting phase.

In conclusion, it can be said that SIR is robust against label noise in terms of both (later) overfitting to the noisy labels and improved generalization despite the label noise compared to unregularized neural networks.

Comparison with standard regularizers:

The results in terms of (final) test accuracy from the experiments where the standard regularization methods, L^1-norm, L^2-norm, maxnorm and dropout, were applied on the two architectures, `LeNet5` and `ConvNet`, are shown in Fig. 5.13. This is directly compared to the mean test accuracy values the self-imitation regularized models had achieved, which were initially presented in the plots in Fig. 5.9.

Interestingly, while the standard regularization methods, especially dropout, improve the test accuracy and partially outperform SIR in the `LeNet5` architecture (Fig. 5.13a), the opposite is the case for the `ConvNet` architecture (Fig. 5.13b). Here, SIR substantially improves the test accuracy in comparison to the standard regularization methods, and no single standard regularizer surpassed SIR when label noise was applied. In fact, some standard regularizers barely improve over the baseline, whereas others completely fail on this architecture when learning on noisy labels. The fact that the L^2-norm regularization performed superior on the `ConvNet` architecture on the original data (i.e., no label noise) but entirely failed (i.e., 10% test accuracy) whenever label noise was incorporated was unexpected and does not actually speak for robustness against label noise with L^2-norm regularization.



(a) `LeNet5` on CIFAR-10 **(b)** `ConvNet` on CIFAR-10

Figure 5.13.: Test accuracy of differently learned neural networks to different degrees of label noise either of the **(a)** `LeNet5` or **(b)** `ConvNet` architecture, which were regularized by SIR as already presented in Fig. 5.9, but the mean accuracy scores are plotted for SIR. In addition, the test accuracy achieved with standard regularization methods is depicted.

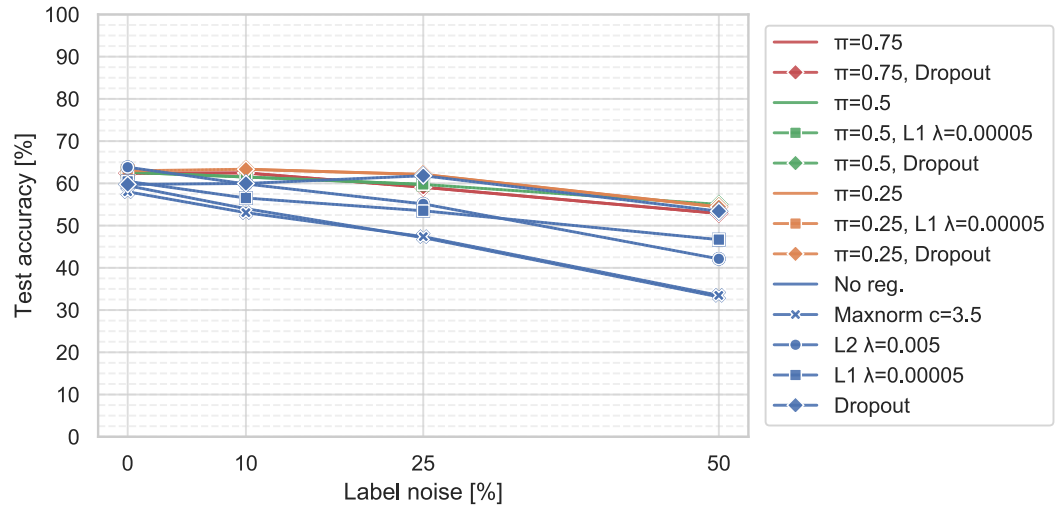
Combinations of SIR and standard regularizers:

Frequently, if SIR is surpassed by a standard regularization method according to the final test accuracy, this is not the case when consulting the best test accuracy. This shows that both regularization settings learned interim models while training that achieve the same test accuracy but the self-imitation regularized neural network tends to overfit more in the later course of training (compare the final test accuracy values shown in Fig. 5.9 to the best accuracy values in Fig. 5.12). This suggests the assumption that SIR although it is susceptible to overfitting, but contributes to generalization. Therefore, SIR might be combined beneficially with standard regularization methods that prevent the later overfitting resulting in new peak performance.

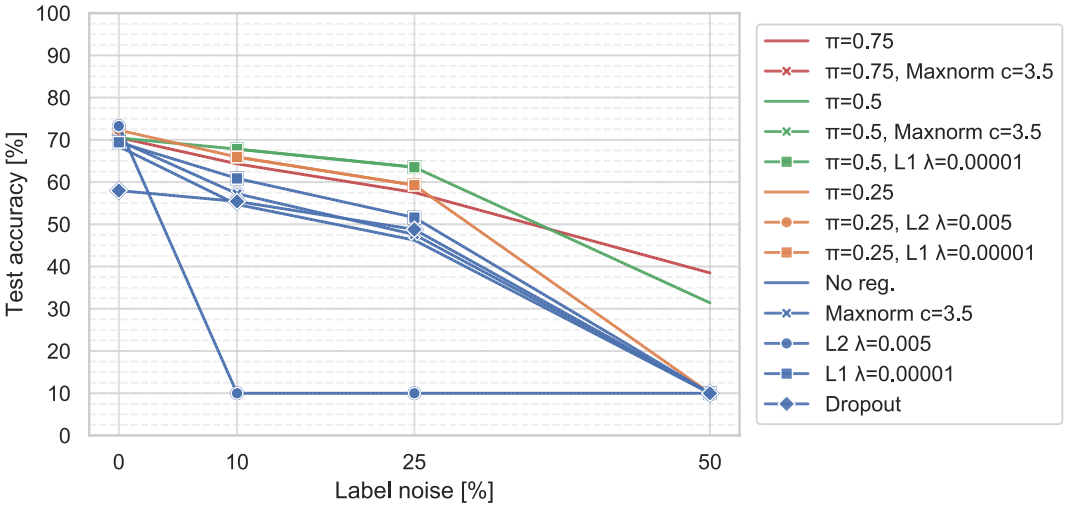
Fig. 5.14 shows the performance of the most fruitful combinations of SIR (separately per the choice of the imitation rate $\pi = \{0.25, 0.5, 0.75\}$) with standard regularizers or the performance of SIR alone if no improvement could be obtained in any combination with a standard regularization method. It becomes apparent that in almost all different label noise levels, SIR in combination with a standard regularizer was able to surpass the standard regularizers on the `LeNet5` architecture (see Fig. 5.14a) and the performance of the self-imitation regularized neural networks is further pushed if a standard regularizer was added on the `ConvNet` architecture (see Fig. 5.14b).

Again, it turns out that asymmetric SIR performs better than symmetric SIR since all profitable combinations with a standard regularization method involve asymmetric SIR (with $\widehat{\zeta} = 2$ and $\widetilde{\zeta} = 1$). More precisely, SIR without target gradients was involved in achieving the new peak performances on the `LeNet5` architecture while SIR with target gradients was applied when new peak performances have been achieved on the `ConvNet` architecture.

Considering the standard regularization methods that led to new peak performances, it turns out that most often L^1-norm, dropout and maxnorm regularization were involved. These findings are in line with the results of the statistical tests applied on the performance scores of different combinations of SIR and standard regularization over multiple data sets in Sec. 5.1.1 since the (partially slight) tendencies point out that these regularization methods might be beneficially combined with SIR. Furthermore, the results of combinations with the L^2-norm regularization do not stand out, which is consistent with the statistical significantly deterioration in some configurations in the statistical testing. Combinations with the maxnorm regularization achieved improvements as well as near-peak performance (oftentimes not

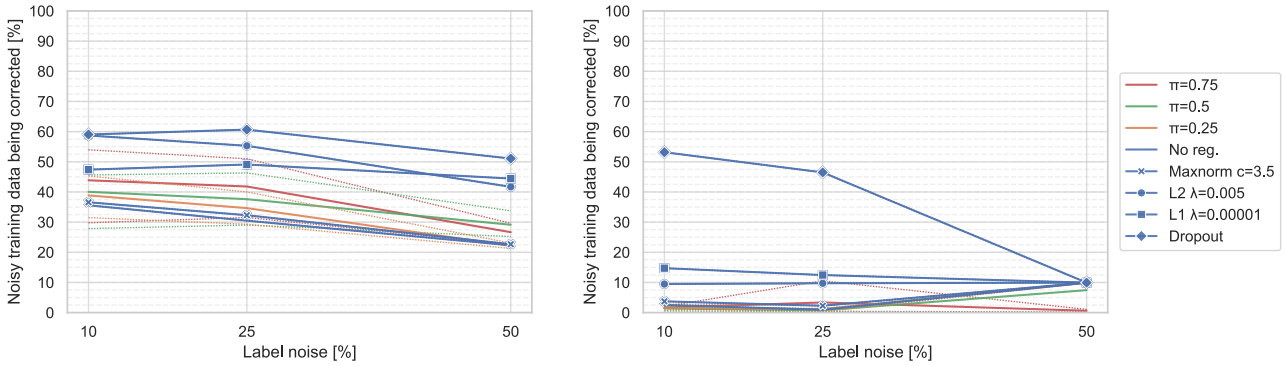


(a) `LeNet5` on CIFAR-10



(b) `ConvNet` on CIFAR-10

Figure 5.14.: Peak test accuracy performance of differently learned neural networks to different degrees of label noise either of the **(a)** `LeNet5` or **(b)** `ConvNet` architecture, which were regularized by SIR combined with an additional standard regularizer. The test accuracy achieved with standard regularized as well as unregularized neural networks is depicted (in blue).



(a) `LeNet5` on CIFAR-10 **(b)** `ConvNet` on CIFAR-10

Figure 5.15.: Measures the accuracy on the (subset of) training data samples that had noisy labels while learning but considering the original labels instead in this evaluation. The learned neural networks are either of the **(a)** `LeNet5` or **(b)** `ConvNet` architecture and were regularized by SIR to different extends, other standard regularization methods or remained unregularized, and are plotted to different degrees of label noise. For SIR different symmetric and asymmetric settings were applied and the mean performance is plotted (solid lines) while the highest and lowest performances that were achieved are plotted with thin dotted lines.

the peak-performance, which is why this is not that apparent in Fig. 5.14), which is consistent with the finding of statistically significantly frequent improvements in some configurations.

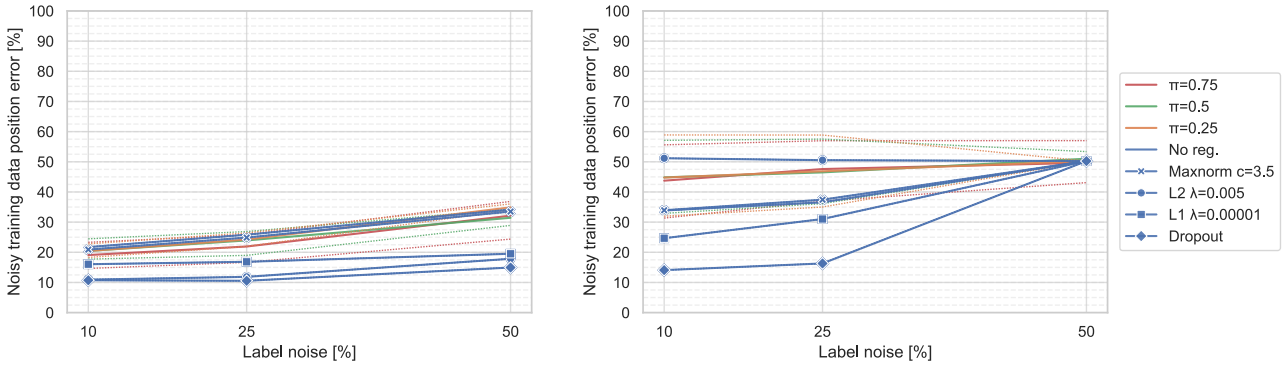
Eventually, one can conclude that SIR can be combined complementarily with other regularization methods (except the L^2-norm regularization) to achieve new peak performances on experiments with label noise applied to different degrees. This strongly supports Hyp. IV (Complementary to standard regularization methods).

5.3.2 Correcting ability of noisy training labels

The ability of correctly predicting training instances whose labels were noisified beforehand (and used over the course of training) can be observed on such noisy training samples but incorporating the original (i.e., correct) labels instead to measure the neural network's performance. Thus, the accuracy on the noisy training samples is measured and presented in Fig. 5.15, and referred to as *correction accuracy* in the following.

It becomes apparent that SIR is not able to get the neural network to correct the noisy labels after learning. For the `ConvNet` architecture (Fig. 5.15b) SIR as well as all other learned neural networks, except for the remarkable results where dropout was applied, fail to correct such training samples and completely overfit on the noisy labels (since the correction accuracy is 0%) due to the high capacity of the `ConvNet` architecture making such memorization possible. Alternatively, some other neural networks reach a correction accuracy of exactly 10% if these have not learned anything which has already been identified in Fig. 5.9b, and therefore correct noisy labels by chance.

In the case of the `LeNet5` architecture, the correction ability seems superior first, but this good correction ability can be directly contributed to the fact that the `LeNet5` capacity is too low in order to overfit on the noisy training data labels which implicitly acts as strong regularization. Therefore, most training samples with noisy labels are classified as if they had never been seen before since the peculiarity of this sample due to the noisy label cannot be memorized with the limited capacity. This is the reason why this



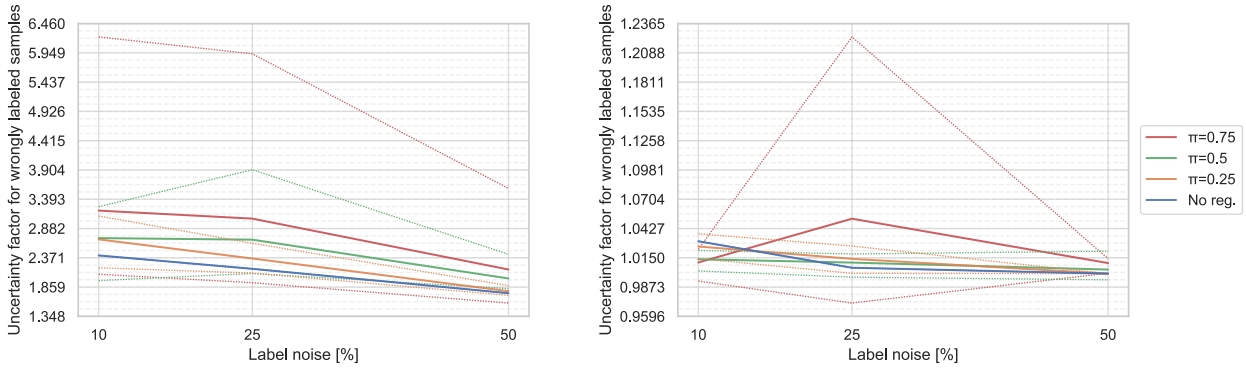
(a) `LeNet5` on CIFAR-10 **(b)** `ConvNet` on CIFAR-10

Figure 5.16.: Measures the position error on the (subset of) training data samples that had noisy labels while learning but considering the original labels instead in this evaluation. The learned neural networks are either of the **(a)** `LeNet5` or **(b)** `ConvNet` architecture where all experimental settings remain the same as in the evaluation shown in Fig. 5.15.

plot is highly similar to the one showing the results on the test accuracy (in Fig. 5.9a), and the correcting accuracy is high if it is the test accuracy. By distinguishing between the symmetric and asymmetric SIR settings on the `LeNet5` architecture the tendency reveals that symmetric (in the setting of $\zeta = 1$) outperforms asymmetric SIR ($\widehat{\zeta} = 2$, $\widetilde{\zeta} = 1$ and target gradients either activated or deactivated) regarding the correction accuracy and corresponds to the highest performances plotted with the thin dotted lines in Fig. 5.15a. Note that the symmetric SIR setting with $\zeta = 2$ does not show any conspicuous change compared to the unregularized baseline.

Since it is not only a peculiarity for SIR that the correction of incorrect training data labels is not successful and also causes problems for other regularization methods, the question arises whether the predictions in a self-imitation regularized neural network considers the correct class also to be very likely (i.e., also assigns a high probability). For example, the second highest probability might be assigned to the correct class. As the quality of a misclassification is aimed to be quantified, the (normalized) position error can be used for this purpose as described in Sec. 4.4.1. This error metric does not evaluate all errors equally, as the accuracy measure does, but gives less weight if the correct class gets a high probability, i.e., a high rank according to the probability scores.

Fig. 5.16 shows the correction ability in the same circumstances as before (in Fig. 5.15), but the normalized position error is used for evaluation instead of the accuracy and is therefore referred to as the *correction position error* in the following. Again, the tendency that the symmetric SIR setting with $\zeta = 1$ performs better than the asymmetric SIR settings on the `LeNet5` architecture can be observed in Fig. 5.16a. For the `ConvNet` this becomes strikingly apparent as the position errors differ extensively between symmetric and asymmetric SIR. Here, the position error for the asymmetric SIR settings is very high (above 50%), which means that the ranking position of the correct training data label is worse on average than for a random ranking since this would make a position error of 50% on average. On the other hand, the symmetric SIR setting with $\zeta = 1$ performs even slightly better than the unregularized neural network according to a reduced position error which means that the correct class is ranked higher than in the unregularized neural network. These findings can be seen in Fig. 5.16b where the maximum error curves (dotted curves) correspond to the asymmetric SIR setting whereas the minimum ones to the symmetric SIR setting.



(a) `LeNet5` on CIFAR-10 **(b)** `ConvNet` on CIFAR-10

Figure 5.17.: Incorrect label uncertainty factors computed by the division of the mean prediction probability corresponding to a correct by the one corresponding to a noisified training sample. The learned neural networks are either of the **(a)** `LeNet5` or **(b)** `ConvNet` architecture where all experimental settings remain the same as in the evaluation shown in Fig. 5.15.

Finally, it can be concluded that stronger regularization methods, such as dropout, are required to correct noisy training labels. This achieves such good results because of varying the neural network architecture throughout training that makes memorization difficult or even impossible. SIR cannot counteract the data loss enough to prevent memorization/ overfitting. To do this, the data loss (Eq. 3.2) must be completely canceled out or compensated by the imitation loss (Eq. 3.1) if the data label is considered to be incorrect.

More generally, in order to be able to correct, an awareness that a training data label is incorrect must be present first. In other words, the confidence of a training sample prediction must be different depending on whether there is an erroneous data label or not. This could be measured in the difference between the highest predicted probability in the case of a wrongly labeled training sample and the one for a correctly labeled sample. Note that comparing the pure probability values across differently regularized neural networks is not meaningful and would not allow drawing conclusions since the prediction distribution characteristics per se might be completely different. For example, in Sec. 5.2.1 and more specifically in Fig. 5.6 it is shown that unregularized neural networks predict with very hard softmax distributions and consequently the maximum probability is very high (near 100%) (see Fig. 5.6e) whereas an asymmetrically self-imitation regularized neural network produces softer softmax distributions where the maximum probability is not that extreme valued (see Fig. 5.6b). Therefore, the evaluation concerning the uncertainty of the correctness of a training data label is performed by dividing the mean maximum probabilities of a neural network's output distributions for correct training data samples by the mean of the maximum ones predicted for incorrect training data samples. This is denoted with *(incorrect label) uncertainty factor* where a higher uncertainty factor represents a higher sense or awareness of an erroneous data label.

Fig. 5.17 presents the uncertainty factors of SIR and unregularized neural networks on the training data where label noise was applied to different degrees (i.e., 10%, 25% and 50%). Although asymmetric SIR turned out to be often inferior to symmetric SIR and even to other regularization methods as well as to unregularized neural networks in terms of the correction ability of the incorrect training data samples (according to both the accuracy and position error), it also acquires a sense of a training label being likely to be incorrect. This sense is frequently even higher than of the unregularized baseline (i.e., a higher uncertainty factor). Nevertheless, the uncertainty factor of a symmetrically self-imitation regularized

neural network is substantially higher in most cases (highest factors in Fig. 5.17). Overall, one can conclude that SIR prevents that the neural network imprints the wrong label too strongly and thus retains a higher degree of uncertainty since the uncertainty factors of the unregularized neural networks are very small. However, it is worthwhile mentioning that other standard regularization methods are also able to achieve an increased uncertainty factor over the unregularized baseline.

5.4 Learning from other classes

In this section, the results from the experiments where a particular class was underrepresented in the training data to different degrees (as described in Sec. 4.1.3) are shown and discussed. The experiments were carried out on several architectures, and no substantial improvements could be achieved in any of them, which is against Hyp. III (Learn about a class through information about other classes). Again, substantial differences between symmetric and asymmetric SIR occurred in a way that symmetric SIR works similarly to the unregularized neural networks, but asymmetric SIR mostly worsened the performance on the underrepresented class considerably. The neural networks were trained on the CIFAR-10 data set where samples of the class *dog* were reduced on 75%, 50% and 25% of the 5,000 training data samples originally available for this class.

As an inevitable side effect, the training data set becomes imbalanced which might entail the risk that the neural network almost ignores the underrepresented class and only rarely predicts it (i.e., only if the neural network is highly certain) since it is too unlikely to be correct and, conversely, the risk is too high that it fails when predicting it. Frequently, for this underrepresented class, a high precision is achieved but to a low recall. This is the reason why the prediction performance for a single class should be evaluated by considering both the precision and the recall, e.g., by consulting the F_1 measure.

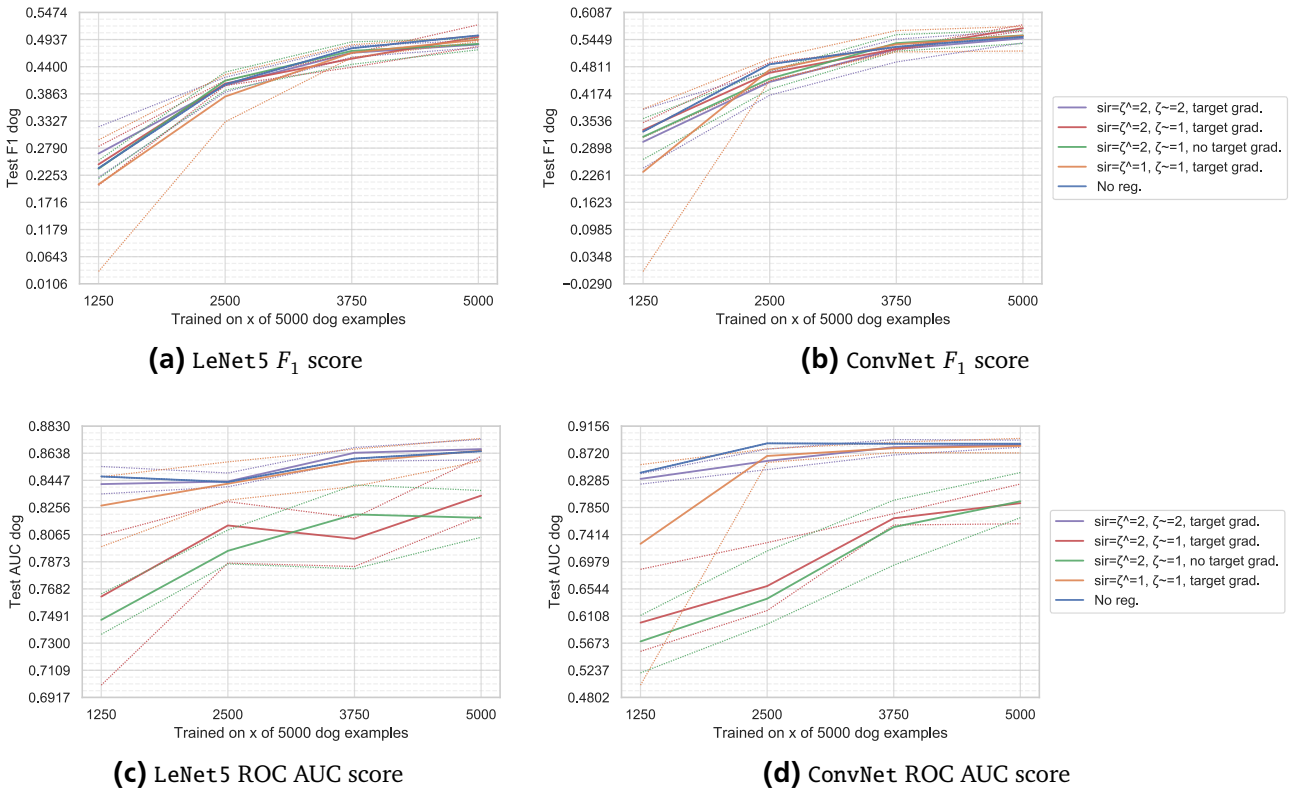
Hinton et al. (2015) reported that the pre-activation values for an underrepresented class are relatively low, which represents the statement that a class is only predicted if the neural network is very certain since the bias parameter for this class is much too low as these were learned to represent the prior associated with the imbalanced distribution of the training data. This issue is pointed out in Sec. 4.4.1 where the ROC AUC measurement is mentioned at the same time to address this problem by evaluating the discriminating power regarding the pre-activation scores (which is equal to use the corresponding softmax probability value) of a particular class isolated from the other ones.

Masko and Hensman (2015) empirically showed on the CIFAR-10 and CIFAR-100 data sets that imbalanced training data sets could have a negative impact on the overall performance of a convolutional neural network. However, according to their experiments with a single class being majorly underrepresented in the training data set, the total performance (test accuracy) was not affected and also the recall of the reduced class was only affected negligibly. Therefore, a single class can be easily reduced without the need for additional methods, such as oversampling (He and Garcia, 2009; Masko and Hensman, 2015), to overcome problems that could be induced by an imbalanced training data set.

The results achieved with the `LeNet5` and `ConvNet` architecture in terms of the F_1 and ROC AUC scores on the test set are presented in Fig. 5.18.

Compared to the unregularized baseline, no improvements can be achieved if SIR is applied according to the F_1 scores. Also among the different SIR settings, no substantial differences can be discerned.

According to the ROC AUC scores, the difference between symmetric SIR and asymmetric SIR becomes apparent as the scores achieved with the asymmetric SIR settings are clearly separated from the ones achieved with symmetric SIR or the unregularized baseline. Especially when the `ConvNet` was trained on 25% of the original *dog* training samples, SIR seriously worsened the discriminating ability of the *dog* pre-activation value since the worst ROC AUC score of 0.5 is approached, which represents an entire



(a) `LeNet5` F_1 score

(b) `ConvNet` F_1 score

(c) `LeNet5` ROC AUC score

(d) `ConvNet` ROC AUC score

Figure 5.18.: Either **(a)** & **(b)** test F_1 score or **(c)** & **(d)** test ROC AUC score of differently learned neural networks either of the **(a)** & **(c)** `LeNet5` or **(b)** & **(d)** `ConvNet` architecture for the class *dog*. The results, obtained by different symmetric SIR (purple and orange curves) and asymmetric SIR (red and green curves) settings averaged over the different imitation rates $\pi = \{0.25, 0.5, 0.75\}$ as well as by unregularized neural networks (blue curve), are plotted over different extends the class *dog* was underrepresented while learning. The highest and lowest performances that were achieved within a specific SIR setting with different imitation rates are plotted with thin dotted lines.

discriminating incapability. In other words, considering the pre-activation value for the class *dog* to form a ranking with the test samples, the probability that a randomly selected sample of the class *dog* is ranked above a sample from another class is 50%. Thus, the ranking is randomly ordered.

Overall, comparing the two asymmetric SIR settings where once target gradients are allowed and once suppressed, SIR with target gradients is superior according to most of the F_1 as well as the ROC AUC scores. Comparing the two symmetric SIR settings where they differ in the choice of applying the softmax function once or twice to produce the prediction $\widetilde{\boldsymbol{o}}$ and target $\widehat{\boldsymbol{y}}$ used for the self-imitation, it becomes apparent that the latter setting with $\zeta = 2$ (purple lines) seems to perform better on average according to both scores and is above all less sensitive to the choice of imitation rate π since symmetric SIR with $\zeta = 1$ the performance drastically drops if π had not been set properly. The question arises why an entropy penalty such as symmetric SIR represents should help in improving the performance on the class *dog*. Perhaps this helps to raise the pre-activation for the class *dog* more pronounced when predicting this class since the resulting softmax distribution is harder and favored by the entropy penalty.

5.5 Training stabilization

Hinton et al. (2015) hinted that soft targets (with high entropy) induce much less variance in the gradient between training samples in comparison to the hard data targets and hence the model can be trained by using a much higher learning rate or, conversely, training can be stabilized. Apart from that, it was not expected that the application of SIR would have a significantly stabilizing impact. However, during the experiments, it turned out that this is an astonishingly strong property of SIR. The improvement of stabilization is most often observable in SIR helping a neural network to learn anything meaningful at all (which is frequently to a great extent). Without SIR, this neural network was not able to learn anything from the data, which is reflected in the fact that it randomly predicts classes or, more likely, always predicts the same class. Frequently, even the application of standard regularization methods could not help in these situations. In this section, different scenarios in which SIR stabilized the training are gathered and briefly discussed.

Stabilization with high capacity models:

Even if the full training data set is available and the capacity of the model is high (`ConvNet` architecture), the unregularized neural network is not able to learn, e.g., from the EMNIST (comb.) and only achieved the lowest test accuracy of 2.13%. Applying (asymmetric) SIR (with $\widehat{\zeta} = 2$ and $\widetilde{\zeta} = 1$), learning is enabled, and the test accuracy is lifted to an impressive score of 86.37% (see Fig. 5.1b or Table A.1d). Only the L^1-norm regularization (Table A.3d) and dropout (Table A.6d) were able to stabilize the training whereas the L^2-norm (Table A.4d) and maxnorm (Table A.5d) regularization were not able to support learning.

Stabilization on small data sets:

SIR has already shown over various experiments that it increases data efficiency, but it is also capable of stabilizing the training on little data such as on 10% of the CIFAR-10 training data with the `ConvNet` architecture. Here, the unregularized (Table A.1b) as well as the L^2-norm regularized (Table A.4b) neural network did not even rudimentarily learn to predict for the ten classes, which results in 10% test accuracy. Although dropout (Table A.6b), L^1-norm (Table A.3b) and maxnorm (Table A.5b) regularization enabled the neural network to learn, applying (asymmetric) SIR (with $\widehat{\zeta} = 2$ and $\widetilde{\zeta} = 1$) increased the test accuracy to 52.47% (Table A.1b) – more than any other of the standard regularization methods was able to achieve in the experiments. Again this outcome supports Hyp. V (Data efficiency).

Stabilize learning from incorrect data labels:

As discussed in detail in Sec. 5.3.1, SIR can improve learning on data sets with incorrect data labels and, further, even stabilizes the training, for example, with the `ConvNet` architecture on the CIFAR-10 data set with 50% label noise (see Fig. 5.9b or the confusion matrices in Fig. 5.10). While this is able to achieve with several SIR settings, no other standard regularization method was found in the experiments that managed to support the neural network, i.e., achieve a test accuracy beyond 10% (Fig. 5.9b).

Instability through SIR:

Despite the excellent results in terms of training stabilization, it is important to note that SIR itself could cause instability so that the neural network is no longer able to learn. However, this effect was only observed when *symmetric* SIR and a high imitation rate π (i.e., $\pi = 0.75$) was used. For example, this occurred on a 10% subset of the EMNIST letter and digits training data set as well as on the full (100%) KMNIST data set (as discussed in Sec. 5.1 regarding Fig. 5.2).

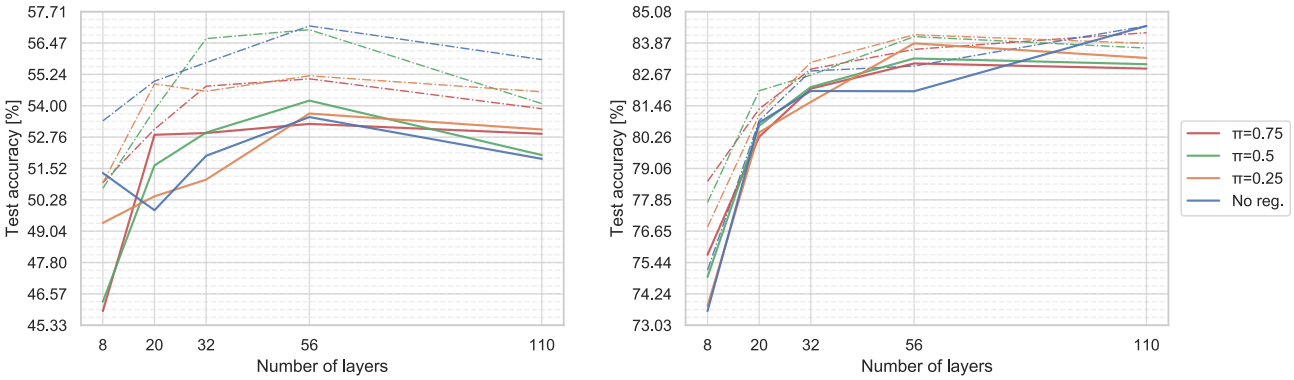
In addition, when SIR and standard regularization methods that performed well on their own were applied in combination, this could also cause instability issues. For instance, this took place on several configurations where SIR (with target gradients) was combined with L^1-norm regularization on the data sets CIFAR-100, EMNIST (comb.) and SVHN (Table A.3) or on the same data sets, except EMNIST (comb.) but EMNIST (letter) instead, when L^2-norm regularization was combined (Table A.4). Maxnorm in combination with SIR (with target gradients) never caused any instability issues (Table A.5) while dropout in combination faced such issues only on 10% of the SVHN training data with the `ConvNet` architecture (Table A.6b). On the other hand, SIR without target gradients in combination with standard regularization methods faced instability issues less often (see Table A.7, Table A.8, Table A.9 and Table A.10).

Yet, in general, it is not clear, which of the regularization methods in a combination, either SIR or the standard regularizer, is responsible for this and it is assumed that the two regularization methods are interfering with each other. Note that this issue weakens Hyp. IV (Complementary to standard regularization methods) especially for the L^1-norm and L^2-norm regularization.

5.6 Depth of a neural network

The effect of SIR is studied on very deep neural networks or, more specifically, how the effect is dependent on the depth of neural networks. Sec. 5.1 showed that SIR is able to regularize (or improve the performance) independently from the capacity of the neural network architecture. However, these differed primarily only in the number of parameters and not essentially in depth. According to the parameter investigation in Sec. 5.2.2 the effects of (asymmetric) SIR were most apparent in the output layer in terms of the weight and bias vector lengths (i.e., smaller lengths than the ones of unregularized neural networks). Any changes in terms of the lengths due to SIR, on the other hand, in the other layers are negligibly small. Therefore, it could be concluded that the regularization behavior does only affects the output layer while the other layers remain relatively unregularized. This suggests the presumption that SIR applied on very deep neural networks might, therefore, be less successful than on shallow neural networks. However, according to the experimental results, this cannot be corroborated – on the contrary – SIR was able to regularize very deep neural networks as well as shallow ones as shown in the following. This means that there has to be a substantial regularizing impact on the hidden layers or at least that excessive overfitting can be limited despite an increasing number of layers/parameters.

The experiments were conducted with the `ResNet` architecture (as presented in Sec. 4.3) on the CIFAR-10 data set where the depth of the neural network was different while keeping all other hyperparameters



(a) 10% training data

(b) 100% training data

Figure 5.19.: Neural networks with the `ResNet` architecture at different depths (i.e., plotted over the number of layers) were trained on either **(a)** 10% or **(b)** 100% of the CIFAR-10 training data set. The solid lines map the final test accuracy whereas the dash-dotted lines map the best test accuracy (i.e., the highest test accuracy achieved by an interim model while learning).

and architectural characteristics fixed. The results achieved with asymmetric SIR with $\widehat{\zeta} = 2$, $\widetilde{\zeta} = 1$ and target gradients are presented here but note that SIR with deactivated target gradients showed similar behavior. The results, plotted over the number of layers, are shown in Fig. 5.19. Here, one can observe that in most cases, SIR was able to improve the (final) test accuracy (i.e., solid lines) compared to the unregularized neural networks at different depths. Hence, SIR does not seem to have difficulties at higher depths, and its effectiveness does not depend on this.

Admittedly, the results are not particularly clear since, for example, on the full data set (Fig. 5.19b) one can observe an apparent improvement over the baseline for the neural networks of 56 layers whereas the performance does not further increase under the appliance of SIR when the number of layers is increased to 110, but the unregularized neural network evidently outperforms them. On the other hand, when training on 10% of the data only this does not happen, and most self-imitation regularized neural networks easily surpass their unregularized counterpart. Inexplicably, in this reduced training data setting, SIR performs worse than the baseline when a shallow neural network with eight layers was used.

In order to further substantiate that SIR is not affected by the depth and is also applicable to (very) deep neural networks, experiments with the `ResNet` architecture on the CIFAR-10 data set were conducted when label noise was applied on the training data set. The label noise experimental set-up is explained in Sec. 4.1.1 and several experiments were conducted with different amounts of label noise, i.e., label noise on 10%, 25% and 50% of the training data samples. The final test accuracy scores (solid lines) after learning with the same architecture (and regularization) on training data that was noisified to different degrees are averaged and depicted in Fig. 5.20 with standard error bands besides. Here it becomes strikingly apparent that (on average) SIR is able to improve on all investigated neural network depths over the unregularized counterpart and even with all the imitation rates investigated (i.e., $\pi \in \{0.25, 0.5, 0.75\}$). Further, without showing an appreciably higher variance according to the standard error bands. Apart from the fact that these results underline the strength of SIR on noisy/erroneous data labels again, it can be concluded that SIR also appears to be effective against label noise regardless of the capacity and especially the depth of neural networks as the studies in Sec. 5.3.1 were mainly confined to shallower neural network architectures of similar depth as the `LeNet5` and `ConvNet` architectures are.

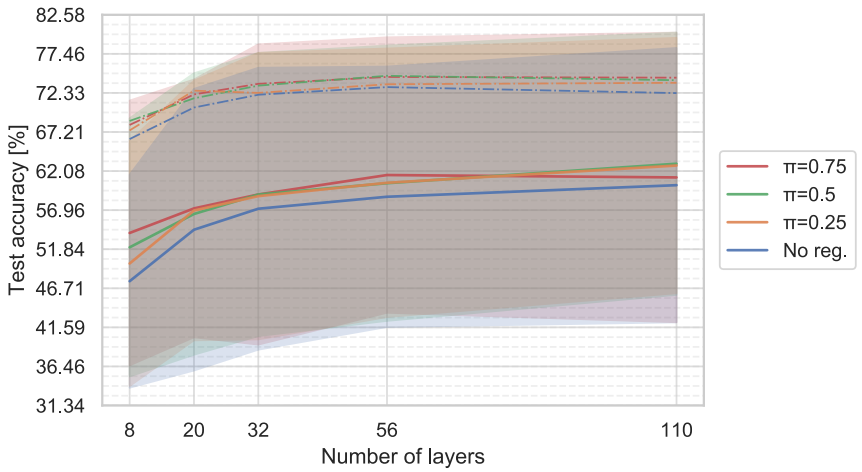


Figure 5.20.: Neural networks with the `ResNet` architecture at different depths (i.e., plotted over the number of layers) were trained on the CIFAR-10 where the training data labels were noisified at different degrees beforehand. The solid lines map the final mean test accuracy, the dash-dotted lines map the best test accuracy (i.e., the highest test accuracy achieved by an interim model while learning) and the error bands (for the final test accuracy) show the variance over the different label noise degrees.

The highest accuracy scores achieved throughout different neural network learning courses with some interim model, which are visualized by the dash-dotted lines in Fig. 5.19 as well as Fig. 5.20, are consistently comparable to the final test accuracy to some extent so that the relation between the differently regularized neural networks with different depths remains unchanged. In other words, a model that performs better according to the best test accuracy than another one also performs better according to the (final) test accuracy. An exception to this are the experiments where the neural networks learned on a smaller training subset (i.e., 10%) shown in Fig. 5.19a where the best accuracy scores seem very noisy but thus rather unreliable. Nonetheless, the gap between the best and the final test accuracy for training with SIR is comparable to the unregularized counterparts for what reason SIR seems to be equally susceptible to overfitting in the later course of training as the unregularized neural network whereby this does also not depend on the depth.

Furthermore, other standard regularizers are also unable to prevent (or at least reduce) this form of later overfitting. Besides, most of them (i.e., L^1-norm and maxnorm regularization) were not able to apparently improve over the unregularized baseline while some (i.e., especially the L^2-norm regularization) turned out to be very dependent on the depth of the neural network where the capability of beneficially regularizing substantially decreases when the depth of the neural network was increased up to resulting in an apparent worsening in deeper neural networks. This leads to two possible conclusions: first that the regularization approach of SIR seems to be superior over the standard regularization methods on the `ResNet` architecture, which is further corroborated by He et al. (2015) claiming that the thin `ResNet` architecture regularizes by design to a certain extent where the standard regularization methods do not seem to be able to further support generalization. Second, the hyperparameters of the standard regularization methods have not been extensively tuned. Therefore, this could be required to improve over the baseline or, in the case of the L^2-norm regularization worsening on the deep neural networks, the hyperparameters might have to be tuned with respect to the depth of the neural network in particular. The former fact supports Hyp. I (Regularization) and the assumption in Hyp. IV (Complementary to standard regularization methods) that SIR might yield another regularization approach than other

regularization methods whereas the latter one supports the finding that SIR seems to be very insensitive to its hyperparameters, which was already observed in other experimental evaluations.

Since the training of very deep neural networks is associated with high computing times, it was refrained from investigating the properties on deep neural networks on a large number of data sets, e.g., in statistical tests.

Overall, these results were not expected not only due to the parameter investigation but further due to the fact that KD is claimed to perform worse on deep neural networks as the student neural network (Romero et al., 2014; Yim et al., 2017). The predictions of the teacher neural network indeed show a richer "solution" than the hard data labels do but do not reveal the "solution approach" that produced these predictions. The hidden activations can be construed as gradually solving the task by processing the information through the hidden layers. Hence, it could have been expected that the same problem applies for SIR, but this presumption does not seem to be valid.

5.7 Pre-fitting

Beyond tuning the imitation rate π in SIR to a value staying fixed while training, one could also consider changing the imitation rate over the course of training. In detail, the imitation rate could be set low at the beginning of training so that the neural network can roughly fit on the data, without being irritated by the self-imitation where the imitation targets $\widehat{\boldsymbol{y}}$ come from an entirely underfitted model and are therefore not meaningful. We refer to this as *pre-fitting*. Simplifying this further, SIR is completely deactivated in the beginning (i.e., $\pi = 0.0$) and activated (i.e., $\pi \neq 0.0$) after a particular number of episodes.

The results from the experiments on the CIFAR-10 data set over various architectures have shown that such pre-fitting has indeed an effect on both the final test accuracy, which tended to improve, and the learning speed since the interim models achieved a certain test accuracy earlier at the beginning of training when (asymmetric) SIR was deactivated, but both are affected only slightly.

However, pre-fitting introduces the hyperparameter tuning to become more necessary and might also be more complicated since not only a fixed value for π must be determined but a schedule over the training course according to which the imitation rate π should be modified. Even in its simplest form as applied in the experiments described before, the period in which SIR is deactivated as well as the certain value to which π is set afterward have to be both suitably determined. For the experiments discussed above, this period was set according to the epoch where the learning curve (in terms of the test accuracy) of an unregularized neural network was at the end of the underfitting phase. Since this is frequently not known beforehand, the determination of the pre-fitting period would be even more difficult in such situations.

According to the assumption formulated in Hyp. II (Compensate for erroneous data labels) that especially at the beginning of the training the imitation targets do not contain useful information but might not irritate the learning since these soft targets do not produce any sharply misleading gradients, seems to be confirmed here since no apparent improvements could be achieved when pre-fitting the neural network before applying SIR. Furthermore, the pre-activations obtained when querying a freshly initialized neural network are distributed close to zero which consequently results in a nearly uniform distribution for the imitation target $\widehat{\boldsymbol{y}}$ especially if a chained softmax function with $\widehat{\zeta} \geq 2$ is used to produce this. Hence, the gradient introduced through the imitation loss (Eq. 3.1) in SIR does not considerably affect the data loss (Eq. 3.2) gradient, which then dominates the multi-objective loss and is primarily used for learning. Consequently, SIR can be considered automatically controlling the influence of the imita-

tion and bringing the imitation (loss) into focus (still limited by the imitation rate π) only when the predictions become more defined after a few epochs.

6 Conclusion

This section aims to give a summarizing overview of the findings on the properties and behavior of the method of Self-Imitation Regularization (SIR). On this occasion, a position is taken on each of the initial working hypotheses formulated while introducing SIR (in Sec. 3.1), and these are rejected (marked with –) or accepted (marked with ✓) if applicable. If it is not possible to make a decision, it is marked with ∼ in the following treatise.

First of all, the most unexpected finding is that symmetric SIR and asymmetric SIR strongly differ in their properties and partly even produce opposite effects. So far asymmetric SIR has only been assumed to be a trick or minor modification compared to the KD-like symmetric SIR in order to obtain non-zero gradients (even if the target gradients are disabled). Therefore, some properties or hypothesis assumptions do only apply for one of the two variants. Further unexpected findings are discussed after the hypotheses treatise below.

✓ **Hypothesis I** (Regularization)**.** SIR regularizes neural networks with different architectures and capacities learning on (visual) data sets. This occurs particularly reliably as the statistical test outcomes suggest (see Sec. 5.1) since SIR was able to regularize in most experimental configurations on significantly or even highly significantly many (visual) data sets (under a significance level of $\alpha = 5\%$ or $\alpha = 1\%$, respectively).

Besides the evaluation based on measuring the performances of the differently regularized neural networks, the parameter investigation (Sec. 5.2.2) hinted that (asymmetric) SIR regularizes by limiting the parameters in the output layer to prevent them from overfitting and leads to soft output distributions (Sec. 5.2.1). Nevertheless, this does not imply that the direct regularization of the parameters in the output layer would lead to SIR only being applicable to shallow neural networks. On the contrary, SIR achieves outstanding results and, above all, independent of the depth of the neural networks, which could be shown in experiments conducted on differently deep neural nets with up to 110 layers (see Sec. 5.6).

However, according to the parameter investigation along with the quality evaluation of the soft predictions/targets, asymmetric SIR does regularize (the parameters in) the output layer implicitly by demanding for soft output distributions due to the very soft imitation targets (caused by, e.g., applying the softmax function a second time). On the other hand, this does not necessarily contribute to generalization due to high-quality soft (imitation) targets. Moreover, these targets are worse than in symmetric SIR in terms of (ranking) quality and also worse than the (soft) predictions of an unregularized neural network. Briefly, asymmetric SIR regularizes well but the (ranking) quality of the predictions, used as imitation targets, is poor whereas symmetric SIR does not improve (i.e., regularizes in terms of generalization performance) that strong but yields predictions that are of an improved (ranking) quality (see Sec. 5.2.3). This suggests that possibly the dark knowledge encoded in the probabilities of the other classes in the imitation target is not used in the asymmetrical case or cannot even be used because it is lost too much, e.g., by the application of the chained softmax function. This does not only apply to Hyp. I (Regularization) but also to all other hypotheses based on the quality of the encoded similarities and correlations in asymmetric SIR. In particular, the presumption that the chained softmax function is responsible for this (e.g., because all classes are assigned clear non-zero probabilities, as shown in Fig. 5.6c) suggests that the temperature softmax function must also be studied in the future whether

similarity information is more distinctly preserved (e.g., since near-zero probabilities in the standard softmax distribution stay relatively low when the temperature is increased). Nevertheless, the chained softmax function is responsible for the main regularization effect that limits the parameters due to the high softness of the chained softmax distributions, which does not necessarily apply to temperature softmax distributions. Remember that another drawback of the temperature softmax function is that the tuning of the temperature hyperparameter is difficult.

In the end, Hyp. I (Regularization) can be accepted since asymmetric SIR shows a clear (and symmetric SIR moderate) regularization behavior.

✓ **Hypothesis II** (Compensate for erroneous data labels)**.** As the experiments on training data sets with label noise showed, SIR (asymmetric as well as symmetric) is robust against label noise. Moreover, on data sets where the degree of label noise was very high (i.e., 50%), SIR was able to stabilize the training with a neural network of a very high capacity while different other regularization methods (i.e., L^1-norm, L^2-norm, maxnorm and dropout regularization) have been applied but failed (see Sec. 5.3.1).

The examination of the correction ability of the noisified training samples (Sec. 5.3.2) shows that SIR cannot prevent overfitting from occurring to the extent that incorrectly labeled training samples are wrongly predicted by assigning the highest probability to the incorrect class. Therefore, SIR using hard labels as imitation loss could not steer against the incorrectness induced by the data loss since they would not differ from the incorrect hard data targets. This affirms Hyp. II **(b)** that the use of soft imitation targets is superior over the use of hard ones regarding the robustness against noisy training data labels. Independent of the information about the other classes (beside the incorrect data label class) including the correct one, that is carried in the soft imitation targets and might be of a low quality in the case of asymmetric SIR, SIR still allows a form of questioning of the data label (as presumed in Hyp. II; Compensate for erroneous data labels) even though the probability for this incorrect class is highest. This is because of the uncertainty that is implied in the probability for predicting the noisy class given a noisified training data sample. This probability is relatively smaller than the highest probability predicting on correctly labeled training samples. This is shown in the discussion about the (incorrect label) uncertainty factor in Sec. 5.3.2 where also asymmetric SIR turned out to be superior over the unregularized neural network. This relative certainty information does not seem to be lost in the imitation target after the second application of the softmax function in asymmetric SIR either. However, symmetric SIR achieved the best results regarding the correction ability (e.g., measured with the position error) because of the improved (ranking) quality standing out from the ones achieved with the unregularized neural network.

In the end, Hyp. II (Compensate for erroneous data labels) can be accepted because of the clear and remarkable results in contributing generalization on training data sets holding incorrect data labels.

– Hypothesis III (Learn about a class through information about other classes)**.** Although the quality of the soft targets in symmetric SIR is high, this does not support considerably in learning from a data set with an underrepresented class so that the performance on this class could be improved. Even worse, asymmetric SIR substantially worsened the prediction performance for this class (see Sec. 5.4).

In general, it could not be demonstrated in any experiments that a neural network can learn about a class through information about other classes if SIR is put to use. Thus, Hyp. III (Learn about a class through information about other classes) has to be rejected. Again, under consideration of the temperature softmax function, this might become possible and should become an element of future work.

~ Hypothesis IV (Complementary to standard regularization methods)**.** Combining (asymmetric) SIR with maxnorm regularization or dropout shows improving tendencies (instead of solely relying on maxnorm respectively dropout) according to the evaluation over multiple data sets in Sec. 5.1.1 and

even statistical significance in an experimental configuration with maxnorm. Further, both standard regularizers achieved improved results in combination with SIR in label noise experiments (see Sec. 5.3.1) even though the maxnorm combination achieved inferior performance compared to the combination of SIR with L^1-norm regularization in a high-capacity model.

It is very difficult to make a reliable statement about L^1-norm regularization because the outcomes vary immensely. As just mentioned, the peak performances in the label noise experiments were achieved in a high-capacity model with SIR in combination with L^1-norm regularization, but L^1-norm regularization also caused the training to become unstable (see Sec. 5.5). Moreover, in some experimental configuration, it was also found that the combination of SIR with L^1-norm regularization led to deterioration on a statistically significant number of data sets against the use of L^1-norm regularization alone (see Sec. 5.1.1).

On the other hand, combining L^2-norm regularization with SIR worsened the performance on a highly significant number of data sets in an experimental configuration for different SIR hyperparameter settings (see Sec. 5.1.1), had not achieved considerable improvements in the label noise experiments (see Sec. 5.3.1) and even destabilized the training in several scenarios (see Sec. 5.5).

Eventually, as the combination of SIR with standard regularization methods is clearly advantageous with maxnorm regularization as well as dropout, on the one hand, but is inoperable with L^2-norm regularization and rather moderate with L^1-norm regularization, Hyp. IV (Complementary to standard regularization methods) cannot be accepted. However, it should not be simply rejected since success or failure of a combination here depends on the chosen standard regularizer and not on the fact that SIR would not be combinable with any standard regularizer. Rather, it is complementary to particular standard regularization methods. Furthermore, it should be noted that the regularization effect of SIR only (in terms of test accuracy) turned out to be higher on many data sets than the one of solely applying a standard regularization method especially for the experimental configurations where the training data samples were reduced to 10% (see Sec. A.1 in the appendix).

✓ **Hypothesis V** (Data efficiency)**.** There is no doubt that SIR increases the data efficiency since the regularizing effects are clearer and more pronounced on smaller subsets of the original training data. This is expressed in the fact that improvements over an unregularized neural network occurred with a statistically significant frequency over multiple data sets on all considered SIR and experimental configurations where a reduced training set, i.e., only 10% of the original training data, was involved (see Sec. 5.1.

Furthermore, the regularization effect of SIR is substantially pronounced on reduced training data sets compared to standard regularization methods (see Sec. A.1 in the appendix) to such an extent that in some cases no other regularization method could be found that would have been able to achieve a higher result than SIR. For example, even if the training has to be stabilized when an unregularized neural network is not able to learn from a small amount of data (see Sec. 5.5).

Overall, Hyp. V (Data efficiency) is accepted since learning on a small amount of data with SIR applied did not cause any difficulties and, on the contrary, even comparatively good results, which speaks for data efficiency.

Applicable on deep neural networks:

Contrary to the presumptions made based on the parameter analyses and the popular statements that KD is not suitable for very deep student neural networks (Romero et al., 2014; Yim et al., 2017), SIR can be profitably applied in very deep neural networks. Details on the experiments with neural networks with the `ResNet` architecture with up to 110 layers are being discussed in Sec. 5.6. Summarizing, SIR

improves neural networks steadily independent of their depth and, besides full and correct training data sets, even on small subsets of the original training data as well as with label noise applied. In contrast, the performance of some standard regularization methods (e.g., L^2-norm regularization) decreases strongly with increasing depth or does not stand out much from the unregularized baseline.

Stabilizes training:

SIR turned out to show a stabilizing ability while training, which means that a neural network that is not able to learn anything from the data, is able to learn with SIR. Beyond that, in most cases, neural networks with higher performance results were learned, especially when the amount of training data was low (i.e., 10% of the original training data), compared to other standard regularization methods. Moreover, SIR is frequently able to stabilize the training even if standard regularization methods fail. See Sec. 5.5 for more details.

Ease of hyperparameter tuning:

SIR does only requires a minimal effort in hyperparameter tuning, or in other words, SIR is not very sensitive to the choice of hyperparameters.Because of replacing the temperature softmax by a 2-chain softmax function (tuning the chain length ζ is usually not necessary) eliminates the tuning of the temperature hyperparameter and, furthermore, the investigations in this thesis describe good criteria when to decide between symmetric and asymmetric SIR. Good default settings according to the experiments examined are $\widehat{\zeta} = 2$ and $\widetilde{\zeta} = 1$ (either with or without target gradients) for asymmetric SIR to achieve a high regularization effect, or $\widehat{\zeta} = \widetilde{\zeta} = 1$ for symmetric SIR to improve the (ranking) quality of the predictions. Regarding the imitation rate π, it was found that setting π to a low value (e.g., $\pi \leq 0.25$) has good guarantees not to worsen the training (e.g., in terms of stability) while setting this to a high value (e.g., $\pi \geq 0.75$) entails more pronounced regularization effects but at a higher risk.

The outcomes in the statistical testing further corroborate the statement that the hyperparameter tuning is simple since improvements were possible on significantly many data sets without individually tuning the hyperparameters for each data set, nor for the different experimental configurations (see Sec. 5.1).

On the contrary, other standard regularization methods, e.g., L^2, seem to require a more extensive tuning as the performance is dependent on various factors according to the experiments being conducted. For example, depending on the data set characteristics (e.g., the extent of label noise; see Sec. 5.3.1) or the depth of the neural network (see Sec. 5.6). Furthermore, no intuitive or reliable prior knowledge or presumptions exist according to which the hyperparameters of some standard regularization methods can be set.

6.1 Future work

In this section, further ideas to extend, improve or further investigate the method of SIR, which might become an integral part of future work, are discussed. In order to structure this clearly, this section is divided into two sections, first explaining further investigation methods and ideas, and finally presenting the ideas for extending/improving SIR.

6.1.1 Further investigations

Temperature softmax function:

As mentioned in the conclusions about Hyp. I (Regularization), the solid regularization effects when applying asymmetric SIR is mainly attributed to the application of the chained (i.e., 2-chain) softmax function. Nevertheless, it is claimed that the low (ranking) quality of the soft predictions/targets is caused by the peculiarity of assigning a clear non-zero probability to all classes in the distribution obtained from the chained softmax function, and it could be suspected that the dark knowledge about the other classes (i.e., similarity and correlation information) is annihilated. Furthermore, as concluded concerning Hyp. III (Learn about a class through information about other classes), this leads to a possible explanation of why this hypothesis could not be substantiated. As symmetric SIR without the application of the chained softmax function considerably improves this quality and the temperature softmax function is able to produce near-zero probabilities for classes associated with low pre-activation values, suggest that the temperature softmax function could increase the (ranking) quality of the soft predictions and targets, and should be investigated as a replacement for the chained softmax function.

Uncertainty evaluation:

In the thesis, the uncertainty of a prediction that can express itself relative to the other predictions in the size of the probabilities or shape of the distribution is investigated. In Sec. 5.3.2 the calculation of an uncertainty factor is proposed, and the evaluation with this shows tendencies that SIR is more able to formulate uncertainty than other counterparts, e.g., when a prediction has to be made on a training sample whose label was noisy during training.

Besides, the topic of formulating uncertainty has been ignored for quite some time, but is now being researched more intensively (Gast and Roth, 2018) since, in general, neural networks lack in reliable and appropriately formulating such uncertainty. Therefore, more sophisticated evaluation methods regarding the certainty formulation ability or, also called, calibration of a neural network have been proposed and utilized in recent years (Guo et al., 2017) such as the expected calibration errors (Naeini et al., 2015). Consequently, these metrics could be considered to use for further evaluations of SIR with regard to an adequate expression of uncertainty.

In-depth investigation of the target network variant:

As presented in Sec. 3.2.2, a target network can be used for imitation target generation, which does not change its parameters over a certain period of training iterations unlike the neural network considered in standard SIR that changes its parameters every iteration. According to the preliminary studies on this variant of SIR, no further improvements concerning the training stability became apparent, which was originally the intention of using a target network.

According to the thorough experiments conducted with (standard) SIR, SIR showed excellent stabilization behavior. However, SIR also induced instability in some experiments. Either when symmetric SIR is used at a high imitation rate (e.g., $\pi = 0.75$) on some data sets or when SIR is combined with some standard regularization methods where the regularizer as well as SIR perform well if they were applied individually. For more details, please refer to Sec. 5.5. In these cases, it should be considered to try the target network variant of SIR and analyze whether the training can be stabilized then.

Further types of data:

In the scope of the thesis, experiments were mainly conducted on visual data as well as some multivariate data sets (in preliminary some preliminary studies). In order to further corroborate the regularization capability of SIR in general, SIR should be applied to other types of data such as sequential data as it is present in natural language processing, for example. Hence, the neural network architecture of RNNs (Goodfellow et al., 2016, ch. 10; Sherstinsky, 2018) could be involved.

6.1.2 Extensions and improvements of Self-Imitation Regularization (SIR)

Sequential learning:

Sequential learning or also called *online learning* refers to learning an ML model on data samples that are not available as a batch of data beforehand, but these are arriving in a (continuous) stream where predictions must be made before all of the data samples have been seen (Bishop, 2006, ch. 3.1.3, p. 143). For example, this could be the case in real-time applications, like stock price forecasting, where the model should be able to adapt itself to current situations.

A commonly-known problem in using neural networks in a sequential learning setting is that these tend to "unlearn" or "forget" solving approaches for previously learned (sub-) tasks when these are currently not present in the data samples considered for training. It is commonly termed *catastrophic forgetting* (McCloskey and Cohen, 1989; French, 1999). As sequential learning on neural networks is common practice in the field of (deep) reinforcement learning, for example, the problem of catastrophic forgetting is strongly present there and is addressed, e.g., with methods that artificially build a data batch to learn on more balanced (including earlier) data samples turning this more into a batch learning task (Lin, 1992).

However, it is hypothesized that SIR could be able to improve learning in sequential learning settings and reduce the problem of catastrophic forgetting by making the previously acquired knowledge present in the loss. This should preserve this knowledge longer even if the current data samples describe another sub-task.

Semi-supervised learning:

The computation of the imitation loss (Eq. 3.1) in SIR does not require any data labels, so one can consider to apply SIR in a semi-supervised learning setting and using the unlabeled data for the imitation loss computation. Hinton et al. (2015) already proposed the idea that the student neural network could be trained on unlabeled data only when a strong pre-trained (on supervised data) teacher neural network is provided. Similarly, Lee (2013) proposed a method that labels the unlabeled data with the predictions of the neural network but using hard predictions instead of soft distributions.

General imitation targets per class:

In SIR as well as KD, an imitation target is generated for each training sample pointing out the similarities and correlations to other classes for this given sample. Instead, one could consider generating imitation targets that contain more general information for a class concerning similarities and correlations to other classes. An idea would be to use the predictions of the neural network on all training samples (e.g., from the previous epoch) and average them by class so that a single, mostly general, imitation target for each class is obtained. Additionally, the predictions from more distant epochs can be taken into account by a (e.g., geometrically) weighted average.

Overall, this could make the imitation targets and therefore the training more stable, on the one hand, and does incidentally solve the problem of symmetric imitation where otherwise zero gradients would

be obtained if the target gradients are not considered. These general imitation targets can be seen as a general enrichment or replacement over the one-hot encoded data targets being independent of a particular training sample.

Direct regularization of the hidden layers:

As discovered in the parameter investigation in Sec. 5.2.2, (asymmetric) SIR directly affects the parameter magnitudes in the output layer (in order to provoke the production of soft softmax distribution due to the pre-activation values being small around zero). In contrast, such an effect does not manifest itself in the hidden layers. Therefore, one could consider to directly regularize the hidden layers with SIR, e.g., by including the hidden activations in the loss function in the form of an imitation target as it is done with the output activations in SIR.

Alternatively, the method of KD is claimed to work poorly on deep student neural network (Romero et al., 2014; Yim et al., 2017) which is why several methods of extending KD to directly incorporating the hidden layers have been introduced. For example, Romero et al. (2014) and Yim et al. (2017), just to name a few. These methods could be examined to construct an adaptation for SIR.

Apart from the fact that SIR already works on deep neural networks (i.e., with many hidden layers) without losing effectiveness when the number of (hidden) layers is increased (as shown in Sec. 5.6), one could imagine that the potential for improvement and the regularization strength could be increased further by such a variant of SIR.

Acknowledgments

I would like to thank Eneldo Loza Mencía and Jinseok Nam for the valuable and helpful discussions, support in the planning and evaluation of the experiments and the opportunities for further education.
I would also like to thank Malcolm Howlett for helping me write this thesis in English.
Finally, I would like to thank my fellow students, friends and family for their support.

Lichtenberg high performance computer of the TU Darmstadt:

Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt. I would like to express my special thanks for this since without these resources it would not have been possible to carry out a comprehensive investigation of the method on the basis of numerous experiments.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A system for large-scale machine learning.

Abbott, T., Abdalla, F. B., Aleksic, J., Allam, S., Amara, A., Bacon, Balbinot, E., Banerji, M., Bechtol, K., Benoit-Levy, A., Bernstein, G. M., Bertin, E., Blazek, J., Bonnett, C., Bridle, S., Brooks, Brunner, R. J., Buckley-Geer, E., d. Burke, L., Caminha, G. B., Capozzi, Carlsen, J., Carnero-Rosell, A., Carollo, M., Carrasco-Kind, M., Carretero, J., Castander, F. J., Clerkin, L., Collett, T., Conselice, C., Crocce, M., Cunha, C. E., D'Andrea, C. B., Costa, L. N. d., Davis, T. M., Desai, S., Diehl, H. T., Dietrich, J. P., Dodelson, S., Doel, P., Drlica-Wagner, A., Estrada, J., Etherington, J., Evrard, A. E., Fabbri, J., d. Finley, A., Flaugher, B., Foley, R. J., Rosalba, P., Frieman, J., Garcia-Bellido, J., Gaztanaga, E., d. Gerdes, W., Giannantonio, T., d. Goldstein, A., Gruen, Gruendl, R. A., Guarnieri, P., Gutierrez, G., Hartley, W., Honscheid, K., Jain, B., d. James, J., Jeltema, T., Jouvel, S., Kessler, R., King, A., Kirk, Kron, R., Kuehn, K., Kuropatkin, N., Lahav, O., Li, T. S., Lima, M., Lin, H., Maia, M. A. G., Makler, M., Manera, M., Maraston, C., Marshall, J. L., Martini, P., McMahon, R. G., Melchior, P., Merson, A., Miller, C. J., Miquel, R., Mohr, J. J., Morice-Atkinson, X., Naidoo, K., Neilsen, E., Nichol, R. C., Nord, B., Ogando, R., Ostrovski, F., Palmese, A., Papadopoulos, A., Peiris, H., Peoples, J., Percival, W. J., Plazas, A. A., Reed, S. L., Refregier, A., Romer, A. K., Roodman, A., Ross, A., Rozo, E., Rykoff, E. S., Sadeh, I., Sako, M., Sanchez, C., Sanchez, E., Santiago, B., Scarpine, Schubnell, M., Sevilla-Noarbe, I., Sheldon, E., Smith, M., Smith, R. C., Soares-Santos, M., Sobreira, F., Soumagnac, M., Suchyta, E., Sullivan, M., Swanson, M., Tarle, G., Thaler, J., Thomas, Thomas, R. C., Tucker, Vieira, J. D., Vikram, Walker, A. R., Wechsler, R. H., Weller, J., Wester, W., Whiteway, L., Wilcox, H., Yanny, B., Zhang, Y., and Zuntz, J. (2016). The Dark Energy Survey: more than dark energy - an overview. *Monthly Notices of the Royal Astronomical Society*, 460(2):1270–1299.

Anderson, E. C. (1999). Monte Carlo Methods and Importance Sampling: Lecture Notes for Stat 578C: Statistical Genetics; University of California, Berkeley.

Ba, J. and Caruana, R. (2014). Do Deep Nets Really Need to be Deep? In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 2654–2662.

Bellman, R., Rand Corporation, and Karreman Mathematics Research Collection (1957). *Dynamic Programming*. Rand Corporation research study. Princeton University Press.

Benavoli, A., Corani, G., and Mangili, F. (2016). Should We Really Use Post-Hoc Tests Based on Mean-Ranks? *Journal of Machine Learning Research*, 17(5):1–10.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Information science and statistics. Springer, New York.

Boyd, S. P. and Vandenberghe, L. (2015). *Convex optimization*. Cambridge Univ. Press, Cambridge, 18. print edition.

Brodmann, K. (1909). Vergleichende Lokalisationslehre der Großhirnrinde in ihren Prinzipien dargestellt auf Grund des Zellbaues. *Leipzig: Barth*.

Bucila, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In Ungar, L., editor, *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, page 535. ACM.

Buda, M., Maki, A., and Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259.

Caflisch, R. E., Morokoff, W. J., and Owen, A. B. (1997). *Valuation of mortgage backed securities using Brownian bridges to reduce effective dimension*. Department of Mathematics, University of California, Los Angeles.

Canny, J. (2016). Berkeley University of California: Lecture CS294-129 Designing, Visualizing and Understanding Deep Neural Networks.

Chapman, P. F., Kairiss, E. W., Keenan, C. L., and Brown, T. H. (1990). Long-Term synaptic potentiation in the amygdala. *Synapse*, 6(3):271–278.

Chinchor, N. (1992). MUC-4 evaluation metrics. In Unknown, editor, *Proceedings of a conference / Fourth Message Understanding Conference, MUC-4*, page 22. Association for Computational Linguistics.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. (2018). Deep Learning for Classical Japanese Literature.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters.

Cybenko, G. (1988). Continuous valued neural networks with two hidden layers are sufficient.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(Jan):1–30.

Domingos, P. (1999). The Role of Occam's Razor in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425.

Dunnett, C. W. (1964). New Tables for Multiple Comparisons with a Control. *Biometrics*, 20(3):482.

Fawcett, T. (2004). ROC graphs: Notes and practical considerations for researchers.

Freedman, D. (2009). *Statistical models: Theory and practice*. Cambridge University Press, Cambridge, second edition edition.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135.

Fukushima, N. (1980). A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202.

Furlanello, T., Lipton, Z. C., Tschannen, M., Itti, L., and Anandkumar, A. (2018). Born Again Neural Networks.

Fürnkranz, J., Hüllermeier, E., Loza Mencía, E., and Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133–153.

Gast, J. and Roth, S. (2018). Lightweight Probabilistic Deep Networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3369–3378. IEEE.

Géron, A. (2017). *Hands-On machine learning with Scikit-learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. Safari Tech Books Online. O'Reilly, Beijing and Boston and Farnham.

Glorot, X. and Bengio, Y. (2011). Understanding the difficulty of training deep feedforward neural networks. pages 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. pages 315–323.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2014). Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout Networks. *JMLR WCP 28 (3): 1319-1327*.

Gross, C. G. (2002). Genealogy of the "grandmother cell". *The Neuroscientist : a review journal bringing neurobiology, neurology and psychiatry*, 8(5):512–518.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On Calibration of Modern Neural Networks.

He, H. and Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition.

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Wiley, New York.

Herculano-Houzel, S. (2009). The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in human neuroscience*, 3:31.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2017). Rainbow: Combining Improvements in Deep Reinforcement Learning.

Hinton, G. (2014). Dark knowledge (Lecture slides): Excellent Lecture Series at Toyota Technological Institute at Chicago (TTIC).

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67.

Hornik, K., Stinchcombe, M. B., and White, H. (1989). Multilayer feedforward networks are universal approximators. *undefined*.

Hu, Z., Ma, X., Liu, Z., Hovy, E., and Xing, E. (2016). Harnessing Deep Neural Networks with Logic Rules.

Hubel, D. H. and Wiesel, T. N. (1963). Shape and arrangement of columns in cat's striate cortex. *The Journal of physiology*, 165:559–568.

Hüllermeier, E. and Fürnkranz, J. (2005). Learning Label Preferences: Ranking Error Versus Position Error. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Famili, A. F., Kok, J. N., Peña, J. M., Siebes, A., and Feelders, A., editors, *Advances in Intelligent Data Analysis VI*, volume 3646 of *Lecture Notes in Computer Science*, pages 180–191. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hurley, N. and Rickard, S. (2009). Comparing Measures of Sparsity. *IEEE Transactions on Information Theory*, 55(10):4723–4741.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift.

Kandel, E. R., Schwartz, J. H., Jessell, T., Siegelbaum, S. A., Hudspeth, A. J., and Mack, S., editors (2013). *Principles of neural science*. McGraw-Hill Medical, New York and Lisbon and London, fifth edition edition.

Karpathy, A. and Li, F.-F. (2015). CS231n Convolutional Neural Networks for Visual Recognition Lecture Notes. `http://cs231n.github.io` accessed on 22.04.2019.

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization.

Kotlowski, W. T., Dembczynski, K., Huellermeier, E., Getoor, L., and Scheffer, T. (2011). Bipartite Ranking through Minimization of Univariate Loss.

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

LeCun, Y. (1998). The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, D.-H. (2013). Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks.

Li, H. (2018). Exploring Knowledge Distillation of Deep Neural Networks for Efficient Hardware Solutions: CS 230 Final Report.

Li, X., Chen, S., Hu, X., and Yang, J. (2018). Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift.

Lin, L.-J. (1992). Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8:293–321.

Lodato, S. and Arlotta, P. (2015). Generating neuronal diversity in the mammalian cerebral cortex. *Annual review of cell and developmental biology*, 31:699–720.

Loza Mencía, E. (2013). *Efficient Pairwise Multilabel Classification*. PhD thesis, Technische Universität.

Masko, D. and Hensman, P. (2015). The Impact of Imbalanced Training Data for Convolutional Neural Networks.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Melnychuk, S., Voronych, A., Nykolaychuk, L., and Krulikovskyi, B. (2017). The structure and components of embedded special processors for determination of entropy signals and random messages. *undefined*.

Mikołajczyk, A. and Grochowski, M. (2018). Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pages 117–122.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

Minsky, M. and Papert, S. (1988). *Perceptrons: An Introduction to Computational Geometry; Expanded edition*. MIT Press, Cambridge, Mass. and London, 3 edition.

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill series in computer science. McGraw-Hill, New York, NY, international ed. edition.

Mittenecker, E. (1979). *Planung und statistische Auswertung von Experimenten*. F. Deuticke, 9 edition.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Naeini, M. P., Cooper, G. F., and Hauskrecht, M. (2015). Obtaining Well Calibrated Probabilities Using Bayesian Binning. *Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, 2015:2901–2907.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch.

Petersen, K. B. et al. (2004). The matrix cookbook.

Provost, F. and Domingos, P. (2003). Tree Induction for Probability-based Ranking. *Machine Learning*, 52(3):199–215.

Reddi, S. J., Kale, S., and Kumar, S. (2018). On the Convergence of Adam and Beyond.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2014). FitNets: Hints for Thin Deep Nets.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Rosenblatt, F. (1962). *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books.

Rubin, T. N., Chambers, A., Smyth, P., and Steyvers, M. (2011). Statistical Topic Models for Multi-Label Document Classification.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533.

Salzberg, S. L. (1997). On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. *Data Mining and Knowledge Discovery*, 1(3):317–328.

Sasaki, Y. (2007). The truth of the F-measure. *Teach Tutor Mater*.

Schulz, P. and Aziz, W. (2018). Tutorial: Variational Inference and Deep Generative Models; University of Heidelberg: Nov 29/30 2018: Introduction slides.

Sherstinsky, A. (2018). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network.

Siegelmann, H. T. and Sontag, E. D. (1992). On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for Simplicity: The All Convolutional Net.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge Massachusetts, second edition edition.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.

Tikhonov, A. N. (1943). *Doklady Akademii Nauk SSSR*, 39(5):195–198.

Torralba, A., Fergus, R., and Freeman, W. T. (2008). 80 million tiny images: A large data set for non-parametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970.

Urban, G., Geras, K. J., Kahou, S. E., Aslan, O., Wang, S., Mohamed, A., Philipose, M., Richardson, M., and Caruana, R. (2017). Do Deep Convolutional Nets Really Need to be Deep and Convolutional?

Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Webb, G. I. (2000). MultiBoosting: A Technique for Combining Boosting and Wagging. *Machine Learning*, 40(2):159–196.

Widrow, B. and Hoff, M. E. (1960). Adaptive Switching Circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104. Institute of Radio Engineers.

Wilcoxon, F. (1950). Some rapid approximate statistical procedures. *Annals of the New York Academy of Sciences*, 52(1):808–814.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.

Yim, J., Joo, D., Bae, J., and Kim, J. (2017). A Gift from Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7130–7138. IEEE.

Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method.

Zhou and Chellappa (1993). Computation of optical flow using a neural network. In *IEEE International Conference on Neural Networks 1993*, pages 71–78 vol.2. IEEE.

A Experimental results

The (mostly) raw results collected from the conducted experiments that are of higher importance are printed in this section.

A.1 Statistical tests raw performance scores

A.1.1 Testing Self-Imitation regularized against unregularized neural networks

Target gradients and $\pi = 0.5$:

Data set	SIR	Unreg.	Diff.
CIFAR10	45.32	46.82	-1.50
CIFAR100	14.62	13.54	1.08
EMNIST (letters)	89.27	88.93	0.34
EMNIST (comb.)	81.66	79.05	2.61
EMNISTdigits	99.07	99.02	0.05
FashionMNIST	86.22	86.11	0.11
KMNIST	89.93	87.98	1.95
MNIST	98.45	98.07	0.38
SVHN	86.86	85.76	1.10

(a) `LeNet5` on 10% training data subset

Data set	SIR	Unreg.	Diff.
CIFAR10	52.47	10.00	42.47
CIFAR100	15.44	15.04	0.40
EMNIST (letters)	91.50	91.08	0.42
EMNIST (comb.)	79.82	79.62	0.20
EMNISTdigits	99.11	98.98	0.13
FashionMNIST	88.02	86.96	1.06
KMNIST	92.98	91.33	1.65
MNIST	98.46	98.50	-0.04
SVHN	19.59	19.59	0.00

(b) `ConvNet` on 10% training data subset

Data set	SIR	Unreg.	Diff.
CIFAR10	60.61	59.45	1.16
CIFAR100	30.88	27.90	2.98
EMNIST (letters)	92.26	92.74	-0.48
EMNIST (comb.)	86.22	84.50	1.72
EMNISTdigits	99.40	99.41	-0.01
FashionMNIST	89.84	89.34	0.50
KMNIST	94.72	95.06	-0.34
MNIST	98.99	99.17	-0.18
SVHN	86.86	85.76	1.10

(c) `LeNet5` on the full (100%) training data

Data set	SIR	Unreg.	Diff.
CIFAR10	68.77	68.43	0.34
CIFAR100	32.08	28.94	3.14
EMNIST (letters)	93.36	93.38	-0.02
EMNIST (comb.)	86.37	2.13	84.24
EMNISTdigits	99.30	99.54	-0.24
FashionMNIST	91.49	91.56	-0.07
KMNIST	96.20	96.53	-0.33
MNIST	99.22	99.13	0.09
SVHN	19.59	19.59	0.00

(d) `ConvNet` on the full (100%) training data

Table A.1.: The raw performance scores (in terms of test accuracy) and the pair-wise differences, which were used for statistical testing.

No target gradients and $\pi = 0.25$:

Data set	SIR	Unreg.	Diff.
CIFAR10	45.20	46.82	-1.62
CIFAR100	14.47	13.54	0.93
EMNIST (letters)	89.73	88.93	0.80
EMNIST (comb.)	81.61	79.05	2.56
EMNISTdigits	99.12	99.02	0.10
FashionMNIST	86.18	86.11	0.07
KMNIST	90.07	87.98	2.09
MNIST	98.45	98.07	0.38
SVHN	87.36	85.76	1.60

(a) `LeNet5` on 10% training data subset

Data set	SIR	Unreg.	Diff.
CIFAR10	48.64	10.00	38.64
CIFAR100	17.03	15.04	1.99
EMNIST (letters)	91.43	91.08	0.35
EMNIST (comb.)	83.31	79.62	3.69
EMNISTdigits	99.13	98.98	0.15
FashionMNIST	87.67	86.96	0.71
KMNIST	92.65	91.33	1.32
MNIST	98.52	98.50	0.02
SVHN	19.59	19.59	0.00

(b) `ConvNet` on 10% training data subset

Data set	SIR	Unreg.	Diff.
CIFAR10	59.79	59.45	0.34
CIFAR100	30.17	27.90	2.27
EMNIST (letters)	92.79	92.74	0.05
EMNIST (comb.)	86.18	84.50	1.68
EMNISTdigits	99.45	99.41	0.04
FashionMNIST	90.06	89.34	0.72
KMNIST	95.37	95.06	0.31
MNIST	99.11	99.17	-0.06
SVHN	87.36	85.76	1.60

(c) `LeNet5` on the full (100%) training data

Data set	SIR	Unreg.	Diff.
CIFAR10	69.18	68.43	0.75
CIFAR100	28.59	28.94	-0.35
EMNIST (letters)	93.58	93.38	0.20
EMNIST (comb.)	85.30	2.13	83.17
EMNISTdigits	99.54	99.54	0.00
FashionMNIST	91.11	91.56	-0.45
KMNIST	96.25	96.53	-0.28
MNIST	99.25	99.13	0.12
SVHN	19.59	19.59	0.00

(d) `ConvNet` on the full (100%) training data

Table A.2.: The raw performance scores (in terms of test accuracy) and the pair-wise differences, which were used for statistical testing.

A.1.2 Testing neural networks with SIR combined with a standard regularizer against this standard regularizer on its own

Target gradients and $\pi = 0.5$:

Data set	SIR & L^1	L^1	Diff.
CIFAR10	46.25	46.46	-0.21
CIFAR100	14.58	13.64	0.94
EMNIST (letters)	89.67	88.80	0.87
EMNIST (comb.)	82.20	79.95	2.25
EMNISTdigits	98.43	98.55	-0.12
FashionMNIST	86.39	85.90	0.49
KMNIST	88.14	88.89	-0.75
MNIST	97.65	98.17	-0.52
SVHN	19.59	87.14	-67.55

(a) `LeNet5` on 10% training data

Data set	SIR & L^1	L^1	Diff.
CIFAR10	49.15	51.74	-2.59
CIFAR100	1.00	16.02	-15.02
EMNIST (letters)	90.86	91.53	-0.67
EMNIST (comb.)	2.13	83.01	-80.88
EMNISTdigits	98.88	99.11	-0.23
FashionMNIST	87.76	87.60	0.16
KMNIST	91.62	92.16	-0.54
MNIST	98.89	98.76	0.13
SVHN	19.59	19.59	0.00

(b) `ConvNet` on 10% training data

Data set	SIR & L^1	L^1	Diff.
CIFAR10	62.63	60.47	2.16
CIFAR100	28.72	30.69	-1.97
EMNIST (letters)	93.06	93.30	-0.24
EMNIST (comb.)	87.13	86.65	0.48
EMNISTdigits	99.03	99.30	-0.27
FashionMNIST	90.70	89.90	0.80
KMNIST	94.60	94.13	0.47
MNIST	98.64	99.12	-0.48
SVHN	19.59	87.14	-67.55

(c) `LeNet5` on 100% training data

Data set	SIR & L^1	L^1	Diff.
CIFAR10	68.41	69.42	-1.01
CIFAR100	34.10	25.92	8.18
EMNIST (letters)	94.16	94.20	-0.04
EMNIST (comb.)	88.31	86.72	1.59
EMNISTdigits	99.46	99.45	0.01
FashionMNIST	91.49	91.56	-0.07
KMNIST	96.41	96.65	-0.24
MNIST	99.25	99.03	0.22
SVHN	19.59	19.59	0.00

(d) `ConvNet` on 100% training data

Table A.3.: The raw performance scores (in terms of test accuracy) and the pair-wise differences between SIR combined with L^1-norm reg. and solely L^1-norm reg. (used for statistical testing).

Data set	SIR & L^2	L^2	Diff.
CIFAR10	49.61	46.71	2.90
CIFAR100	1.00	15.35	-14.35
EMNIST (letters)	88.42	88.17	0.25
EMNIST (comb.)	81.55	79.81	1.74
EMNISTdigits	97.58	98.26	-0.68
FashionMNIST	83.96	85.59	-1.63
KMNIST	86.86	87.22	-0.36
MNIST	97.72	97.63	0.09
SVHN	19.59	83.25	-63.66

(a) `LeNet5` on 10% training data

Data set	SIR & L^2	L^2	Diff.
CIFAR10	10.00	10.00	0.00
CIFAR100	1.00	1.00	0.00
EMNIST (letters)	3.85	90.96	-87.11
EMNIST (comb.)	2.13	2.13	0.00
EMNISTdigits	97.90	98.70	-0.80
FashionMNIST	85.84	87.49	-1.65
KMNIST	88.95	89.64	-0.69
MNIST	97.89	97.94	-0.05
SVHN	19.59	19.59	0.00

(b) `ConvNet` on 10% training data

Data set	SIR & L^2	L^2	Diff.
CIFAR10	53.33	63.83	-10.50
CIFAR100	1.00	29.61	-28.61
EMNIST (letters)	89.17	91.04	-1.87
EMNIST (comb.)	82.69	84.57	-1.88
EMNISTdigits	97.75	98.44	-0.69
FashionMNIST	86.33	89.00	-2.67
KMNIST	89.01	93.07	-4.06
MNIST	97.79	98.49	-0.70
SVHN	19.59	82.91	-63.32

(c) `LeNet5` on 100% training data

Data set	SIR & L^2	L^2	Diff.
CIFAR10	10.00	73.26	-63.26
CIFAR100	1.00	1.00	0.00
EMNIST (letters)	3.85	3.85	0.00
EMNIST (comb.)	2.13	2.13	0.00
EMNISTdigits	98.23	98.68	-0.45
FashionMNIST	86.34	89.48	-3.14
KMNIST	91.41	94.10	-2.69
MNIST	98.62	99.01	-0.39
SVHN	19.59	19.59	0.00

(d) `ConvNet` on 100% training data

Table A.4.: The raw performance scores (in terms of test accuracy) and the pair-wise differences between SIR combined with L^2-norm reg. and solely L^2-norm reg. (used for statistical testing).

Data set	SIR & Maxnorm	Maxnorm	Diff.
CIFAR10	44.91	47.74	-2.83
CIFAR100	15.07	12.67	2.40
EMNIST (letters)	89.49	88.33	1.16
EMNIST (comb.)	82.01	79.21	2.80
EMNISTdigits	98.85	99.09	-0.24
FashionMNIST	85.73	85.85	-0.12
KMNIST	89.99	88.04	1.95
MNIST	98.42	98.04	0.38
SVHN	87.35	86.43	0.92

(a) `LeNet5` on 10% training data

Data set	SIR & Maxnorm	Maxnorm	Diff.
CIFAR10	50.25	42.94	7.31
CIFAR100	14.57	17.01	-2.44
EMNIST (letters)	92.18	91.54	0.64
EMNIST (comb.)	78.67	80.29	-1.62
EMNISTdigits	99.07	99.25	-0.18
FashionMNIST	88.08	87.42	0.66
KMNIST	92.74	92.13	0.61
MNIST	98.63	98.44	0.19
SVHN	19.59	19.59	0.00

(b) `ConvNet` on 10% training data

Data set	SIR & Maxnorm	Maxnorm	Diff.
CIFAR10	60.31	58.01	2.30
CIFAR100	30.90	29.25	1.65
EMNIST (letters)	92.71	92.77	-0.06
EMNIST (comb.)	86.53	85.16	1.37
EMNISTdigits	99.50	99.45	0.05
FashionMNIST	89.05	89.67	-0.62
KMNIST	95.51	94.79	0.72
MNIST	99.08	98.64	0.44
SVHN	87.35	86.43	0.92

(c) `LeNet5` on 100% training data

Data set	SIR & Maxnorm	Maxnorm	Diff.
CIFAR10	70.46	69.81	0.65
CIFAR100	31.37	28.68	2.69
EMNIST (letters)	93.50	93.00	0.50
EMNIST (comb.)	85.88	2.13	83.75
EMNISTdigits	99.64	99.59	0.05
FashionMNIST	91.36	91.24	0.12
KMNIST	95.98	97.02	-1.04
MNIST	99.04	99.25	-0.21
SVHN	19.59	19.59	0.00

(d) `ConvNet` on 100% training data

Table A.5.: The raw performance scores (in terms of test accuracy) and the pair-wise differences between SIR combined with maxnorm and solely maxnorm (used for statistical testing).

Data set	SIR & Dropout	Dropout	Diff.
CIFAR10	50.86	49.71	1.15
CIFAR100	15.99	14.99	1.00
EMNIST (letters)	91.09	91.45	-0.36
EMNIST (comb.)	84.12	83.77	0.35
EMNISTdigits	99.20	99.28	-0.08
FashionMNIST	87.46	87.00	0.46
KMNIST	90.47	90.35	0.12
MNIST	98.69	98.65	0.04
SVHN	87.08	89.70	-2.62

(a) `LeNet5` on 10% training data

Data set	SIR & Dropout	Dropout	Diff.
CIFAR10	49.54	41.27	8.27
CIFAR100	7.09	10.21	-3.12
EMNIST (letters)	91.96	89.69	2.27
EMNIST (comb.)	83.94	81.45	2.49
EMNISTdigits	98.83	97.84	0.99
FashionMNIST	87.32	87.20	0.12
KMNIST	91.15	90.67	0.48
MNIST	98.73	98.32	0.41
SVHN	19.59	83.03	-63.44

(b) `ConvNet` on 10% training data

Data set	SIR & Dropout	Dropout	Diff.
CIFAR10	60.95	59.75	1.20
CIFAR100	27.05	24.18	2.87
EMNIST (letters)	91.31	93.69	-2.38
EMNIST (comb.)	85.98	87.09	-1.11
EMNISTdigits	99.43	99.61	-0.18
FashionMNIST	90.38	91.34	-0.96
KMNIST	94.99	96.04	-1.05
MNIST	99.14	99.35	-0.21
SVHN	86.68	89.70	-3.02

(c) `LeNet5` on 100% training data

Data set	SIR & Dropout	Dropout	Diff.
CIFAR10	61.67	57.99	3.68
CIFAR100	9.47	11.39	-1.92
EMNIST (letters)	93.02	83.81	9.21
EMNIST (comb.)	79.54	76.66	2.88
EMNISTdigits	98.67	96.17	2.50
FashionMNIST	88.98	87.57	1.41
KMNIST	94.44	95.09	-0.65
MNIST	99.01	97.06	1.95
SVHN	19.59	83.03	-63.44

(d) `ConvNet` on 100% training data

Table A.6.: The raw performance scores (in terms of test accuracy) and the pair-wise differences between SIR combined with dropout and solely dropout (used for statistical testing).

No target gradients and $\pi = 0.25$:

Data set	SIR & L^1	L^1	Diff.
CIFAR10	44.87	46.46	-1.59
CIFAR100	14.45	13.64	0.81
EMNIST (letters)	89.84	88.80	1.04
EMNIST (comb.)	82.24	79.95	2.29
EMNISTdigits	98.57	98.55	0.02
FashionMNIST	85.93	85.90	0.03
KMNIST	89.58	88.89	0.69
MNIST	97.64	98.17	-0.53
SVHN	85.97	87.14	-1.17

(a) `LeNet5` on 10% training data

Data set	SIR & L^1	L^1	Diff.
CIFAR10	10.00	51.74	-41.74
CIFAR100	13.70	16.02	-2.32
EMNIST (letters)	91.45	91.53	-0.08
EMNIST (comb.)	84.03	83.01	1.02
EMNISTdigits	99.23	99.11	0.12
FashionMNIST	88.41	87.60	0.81
KMNIST	91.86	92.16	-0.30
MNIST	98.50	98.76	-0.26
SVHN	19.59	19.59	0.00

(b) `ConvNet` on 10% training data

Data set	SIR & L^1	L^1	Diff.
CIFAR10	62.30	60.47	1.83
CIFAR100	30.01	30.69	-0.68
EMNIST (letters)	93.72	93.30	0.42
EMNIST (comb.)	87.44	86.65	0.79
EMNISTdigits	99.26	99.30	-0.04
FashionMNIST	90.45	89.90	0.55
KMNIST	94.93	94.13	0.80
MNIST	98.78	99.12	-0.34
SVHN	85.97	87.14	-1.17

(c) `LeNet5` on 100% training data

Data set	SIR & L^1	L^1	Diff.
CIFAR10	69.33	69.42	-0.09
CIFAR100	1.00	25.92	-24.92
EMNIST (letters)	94.09	94.20	-0.11
EMNIST (comb.)	88.07	86.72	1.35
EMNISTdigits	99.41	99.45	-0.04
FashionMNIST	91.35	91.56	-0.21
KMNIST	96.35	96.65	-0.30
MNIST	99.22	99.03	0.19
SVHN	19.59	19.59	0.00

(d) `ConvNet` on 100% training data

Table A.7.: The raw performance scores (in terms of test accuracy) and the pair-wise differences between SIR combined with L^1-norm reg. and solely L^1-norm reg. (used for statistical testing).

Data set	SIR & L^2	L^2	Diff.
CIFAR10	48.64	46.71	1.93
CIFAR100	1.00	15.35	-14.35
EMNIST (letters)	89.32	88.17	1.15
EMNIST (comb.)	81.96	79.81	2.15
EMNISTdigits	98.05	98.26	-0.21
FashionMNIST	85.87	85.59	0.28
KMNIST	88.53	87.22	1.31
MNIST	98.11	97.63	0.48
SVHN	19.59	83.25	-63.66

(a) `LeNet5` on 10% training data

Data set	SIR & L^2	L^2	Diff.
CIFAR10	10.00	10.00	0.00
CIFAR100	1.00	1.00	0.00
EMNIST (letters)	91.22	90.96	0.26
EMNIST (comb.)	2.13	2.13	0.00
EMNISTdigits	98.58	98.70	-0.12
FashionMNIST	87.73	87.49	0.24
KMNIST	90.58	89.64	0.94
MNIST	98.64	97.94	0.70
SVHN	19.59	19.59	0.00

(b) `ConvNet` on 10% training data

Data set	SIR & L^2	L^2	Diff.
CIFAR10	60.65	63.83	-3.18
CIFAR100	26.43	29.61	-3.18
EMNIST (letters)	90.31	91.04	-0.73
EMNIST (comb.)	84.01	84.57	-0.56
EMNISTdigits	98.11	98.44	-0.33
FashionMNIST	88.15	89.00	-0.85
KMNIST	91.17	93.07	-1.90
MNIST	98.24	98.49	-0.25
SVHN	19.59	82.91	-63.32

(c) `LeNet5` on 100% training data

Data set	SIR & L^2	L^2	Diff.
CIFAR10	72.31	73.26	-0.95
CIFAR100	1.00	1.00	0.00
EMNIST (letters)	3.85	3.85	0.00
EMNIST (comb.)	2.13	2.13	0.00
EMNISTdigits	98.71	98.68	0.03
FashionMNIST	88.18	89.48	-1.30
KMNIST	94.18	94.10	0.08
MNIST	98.93	99.01	-0.08
SVHN	19.59	19.59	0.00

(d) `ConvNet` on 100% training data

Table A.8.: The raw performance scores (in terms of test accuracy) and the pair-wise differences between SIR combined with L^2-norm reg. and solely L^2-norm reg. (used for statistical testing).

Data set	SIR & Maxnorm	Maxnorm	Diff.
CIFAR10	44.81	47.74	-2.93
CIFAR100	13.79	12.67	1.12
EMNIST (letters)	89.72	88.33	1.39
EMNIST (comb.)	81.55	79.21	2.34
EMNISTdigits	99.09	99.09	0.00
FashionMNIST	84.98	85.85	-0.87
KMNIST	90.37	88.04	2.33
MNIST	98.43	98.04	0.39
SVHN	87.75	86.43	1.32

(a) `LeNet5` on 10% training data

Data set	SIR & Maxnorm	Maxnorm	Diff.
CIFAR10	43.92	42.94	0.98
CIFAR100	11.20	17.01	-5.81
EMNIST (letters)	91.43	91.54	-0.11
EMNIST (comb.)	83.65	80.29	3.36
EMNISTdigits	99.08	99.25	-0.17
FashionMNIST	87.94	87.42	0.52
KMNIST	93.14	92.13	1.01
MNIST	98.81	98.44	0.37
SVHN	19.59	19.59	0.00

(b) `ConvNet` on 10% training data

Data set	SIR & Maxnorm	Maxnorm	Diff.
CIFAR10	60.33	58.01	2.32
CIFAR100	30.67	29.25	1.42
EMNIST (letters)	93.03	92.77	0.26
EMNIST (comb.)	86.86	85.16	1.70
EMNISTdigits	99.50	99.45	0.05
FashionMNIST	90.06	89.67	0.39
KMNIST	95.70	94.79	0.91
MNIST	99.24	98.64	0.60
SVHN	87.75	86.43	1.32

(c) `LeNet5` on 100% training data

Data set	SIR & Maxnorm	Maxnorm	Diff.
CIFAR10	68.26	69.81	-1.55
CIFAR100	32.12	28.68	3.44
EMNIST (letters)	93.46	93.00	0.46
EMNIST (comb.)	84.88	2.13	82.75
EMNISTdigits	99.50	99.59	-0.09
FashionMNIST	90.93	91.24	-0.31
KMNIST	97.05	97.02	0.03
MNIST	99.07	99.25	-0.18
SVHN	19.59	19.59	0.00

(d) `ConvNet` on 100% training data

Table A.9.: The raw performance scores (in terms of test accuracy) and the pair-wise differences between SIR combined with maxnorm and solely maxnorm (used for statistical testing).

Data set	SIR & Dropout	Dropout	Diff.
CIFAR10	50.61	49.71	0.90
CIFAR100	16.11	14.99	1.12
EMNIST (letters)	91.21	91.45	-0.24
EMNIST (comb.)	83.81	83.77	0.04
EMNISTdigits	99.20	99.28	-0.08
FashionMNIST	87.68	87.00	0.68
KMNIST	90.58	90.35	0.23
MNIST	98.61	98.65	-0.04
SVHN	88.79	89.70	-0.91

(a) `LeNet5` on 10% training data

Data set	SIR & Dropout	Dropout	Diff.
CIFAR10	45.56	41.27	4.29
CIFAR100	6.40	10.21	-3.81
EMNIST (letters)	91.18	89.69	1.49
EMNIST (comb.)	82.32	81.45	0.87
EMNISTdigits	99.08	97.84	1.24
FashionMNIST	87.00	87.20	-0.20
KMNIST	92.93	90.67	2.26
MNIST	98.57	98.32	0.25
SVHN	81.30	83.03	-1.73

(b) `ConvNet` on 10% training data

Data set	SIR & Dropout	Dropout	Diff.
CIFAR10	60.60	59.75	0.85
CIFAR100	26.85	24.18	2.67
EMNIST (letters)	93.52	93.69	-0.17
EMNIST (comb.)	86.41	87.09	-0.68
EMNISTdigits	99.56	99.61	-0.05
FashionMNIST	91.39	91.34	0.05
KMNIST	95.31	96.04	-0.73
MNIST	99.22	99.35	-0.13
SVHN	88.79	89.70	-0.91

(c) `LeNet5` on 100% training data

Data set	SIR & Dropout	Dropout	Diff.
CIFAR10	61.61	57.99	3.62
CIFAR100	11.85	11.39	0.46
EMNIST (letters)	84.96	83.81	1.15
EMNIST (comb.)	79.04	76.66	2.38
EMNISTdigits	98.22	96.17	2.05
FashionMNIST	89.88	87.57	2.31
KMNIST	94.42	95.09	-0.67
MNIST	98.59	97.06	1.53
SVHN	81.30	83.03	-1.73

(d) `ConvNet` on 100% training data

Table A.10.: The raw performance scores (in terms of test accuracy) and the pair-wise differences between SIR combined with dropout and solely dropout (used for statistical testing).

A.2 Noisy data labels results

Label Noise	π	Chained target softmax	Target grad	Test acc	Test pos err	Train acc	Best test acc
0.10	0.00	1.00	No	54.01	15.03	78.47	60.54
0.10	0.25	1.00	Yes	53.29	15.69	78.90	60.54
0.10	0.25	2.00	No	56.43	14.04	75.25	62.49
0.10	0.25	2.00	Yes	56.10	14.82	78.97	61.21
0.10	0.50	1.00	Yes	56.69	14.15	76.08	62.01
0.10	0.50	2.00	No	56.61	15.17	79.40	61.90
0.10	0.50	2.00	Yes	56.91	14.97	79.51	61.05
0.10	0.75	1.00	Yes	58.84	12.68	68.72	59.55
0.10	0.75	2.00	No	56.98	16.01	76.85	60.53
0.10	0.75	2.00	Yes	56.71	15.48	76.84	61.36
0.25	0.00	1.00	No	47.14	19.09	68.66	58.31
0.25	0.25	1.00	Yes	45.78	20.14	72.44	58.60
0.25	0.25	2.00	No	50.92	17.71	66.15	57.77
0.25	0.25	2.00	Yes	51.16	17.62	65.85	57.54
0.25	0.50	1.00	Yes	52.18	16.49	60.51	57.59
0.25	0.50	2.00	No	51.55	18.37	69.43	58.74
0.25	0.50	2.00	Yes	50.70	20.24	71.81	59.17
0.25	0.75	1.00	Yes	53.59	15.11	53.43	55.97
0.25	0.75	2.00	No	52.93	18.55	66.12	58.94
0.25	0.75	2.00	Yes	53.27	18.33	64.19	58.52
0.50	0.00	1.00	No	33.11	28.76	51.67	49.58
0.50	0.25	1.00	Yes	34.36	28.42	52.26	50.62
0.50	0.25	2.00	No	35.13	29.17	56.63	52.26
0.50	0.25	2.00	Yes	34.31	29.77	57.30	52.26
0.50	0.50	1.00	Yes	41.84	24.75	50.56	52.03
0.50	0.50	2.00	No	37.09	28.55	55.24	52.01
0.50	0.50	2.00	Yes	39.79	25.60	46.80	50.74
0.50	0.75	1.00	Yes	31.14	22.48	25.56	32.76
0.50	0.75	2.00	No	40.21	27.25	49.54	51.85
0.50	0.75	2.00	Yes	38.06	30.29	54.62	51.97
0.00	0.00	1.00	No	59.45	11.04	84.61	64.05
0.00	0.25	1.00	Yes	58.67	11.31	85.24	62.92
0.00	0.25	2.00	No	59.79	12.81	86.99	63.34
0.00	0.25	2.00	Yes	60.18	12.86	88.24	63.73
0.00	0.50	1.00	Yes	58.71	11.22	83.69	63.54
0.00	0.50	2.00	No	59.91	12.67	84.19	63.29
0.00	0.50	2.00	Yes	60.61	12.73	85.66	63.34
0.00	0.75	1.00	Yes	59.87	10.91	74.28	61.55
0.00	0.75	2.00	No	60.09	13.72	85.53	63.43
0.00	0.75	2.00	Yes	59.78	12.99	82.69	62.45

Table A.11.: Performance scores of the `LeNet5` neural network training with label noise involved. "Acc" denotes the accuracy on either the test or train(ing) data set, "pos err" the position error and "best test acc" the highest accuracy achieved on the test set over the course of training.

Label Noise	π	Chained target softmax	Target grad	Test acc	Test pos err	Train acc	Best test acc
0.10	0.00	1.00	No	54.69	14.70	96.52	62.62
0.10	0.25	1.00	Yes	57.88	13.76	98.26	63.63
0.10	0.25	2.00	No	63.13	14.56	99.37	67.76
0.10	0.25	2.00	Yes	60.88	15.92	99.95	63.90
0.10	0.50	1.00	Yes	53.99	15.46	97.52	60.19
0.10	0.50	2.00	No	61.98	15.06	99.60	66.02
0.10	0.50	2.00	Yes	63.56	14.50	99.81	67.18
0.10	0.75	1.00	Yes	62.31	12.55	98.32	66.05
0.10	0.75	2.00	No	63.75	15.20	99.62	67.24
0.10	0.75	2.00	Yes	64.32	15.31	99.60	68.12
0.25	0.00	1.00	No	46.25	19.05	97.69	59.35
0.25	0.25	1.00	Yes	48.14	18.53	96.78	60.44
0.25	0.25	2.00	No	56.73	18.60	99.79	64.52
0.25	0.25	2.00	Yes	52.31	20.72	99.69	59.07
0.25	0.50	1.00	Yes	47.03	19.23	98.02	58.79
0.25	0.50	2.00	No	56.25	18.94	99.86	63.48
0.25	0.50	2.00	Yes	57.10	18.44	99.86	62.44
0.25	0.75	1.00	Yes	54.78	19.49	91.99	62.52
0.25	0.75	2.00	No	54.23	22.12	99.63	62.42
0.25	0.75	2.00	Yes	57.56	18.18	99.74	63.20
0.50	0.00	1.00	No	10.00	50.00	10.13	10.00
0.50	0.25	1.00	Yes	10.00	50.00	10.13	10.00
0.50	0.25	2.00	No	10.00	50.00	10.00	10.00
0.50	0.25	2.00	Yes	10.00	50.00	10.00	10.00
0.50	0.50	1.00	Yes	10.00	50.00	10.09	10.00
0.50	0.50	2.00	No	10.00	50.00	10.00	10.00
0.50	0.50	2.00	Yes	31.40	33.68	99.90	47.79
0.50	0.75	1.00	Yes	38.49	25.54	97.34	53.22
0.50	0.75	2.00	No	35.19	34.76	99.59	52.87
0.50	0.75	2.00	Yes	34.54	34.93	99.57	53.22
0.00	0.00	1.00	No	68.43	8.68	98.59	70.62
0.00	0.25	1.00	Yes	65.86	9.66	98.70	69.00
0.00	0.25	2.00	No	69.18	11.03	99.96	70.06
0.00	0.25	2.00	Yes	68.72	11.49	99.95	70.46
0.00	0.50	1.00	Yes	69.52	8.79	98.41	72.19
0.00	0.50	2.00	No	67.38	12.08	99.84	69.91
0.00	0.50	2.00	Yes	68.77	11.63	99.27	71.26
0.00	0.75	1.00	Yes	68.09	9.05	98.76	69.53
0.00	0.75	2.00	No	68.43	11.95	99.82	69.56
0.00	0.75	2.00	Yes	68.41	12.29	99.51	70.77

Table A.12.: Performance scores of the `ConvNet` neural network training with label noise involved. "Acc" denotes the accuracy on either the test or train(ing) data set, "pos err" the position error and "best test acc" the highest accuracy achieved on the test set over the course of training.

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, J. Jäger, die vorliegende Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum / Date: Unterschrift / Signature:

_____ _____