University Regensburg
Faculty of Philosophy
Language, Literature and Culture
Institute of Information and Media, Language and Culture (I:IMSK)
Department of Media Informatics

3D Pose Estimation Problem and a Convergence Analysis of Adaptive Gradient Descent Algorithms

Dissertation

submitted by

Sebastian Bock

supervised by

First examiner: Prof. Dr. phil. habil. Christian Wolff

Second examiner: Prof. Dr. rer. nat. Martin G. Weiß

Regensburg, January 10, 2023

Contents

Relationship to Published Work

The chapters in this thesis describe work that has been published in the following conferences:

- Chapter 2 - 3
  - U-Shape Phenomenon with Gaussian Noise and Clipped Inputs.
    Sebastian Bock, Philipp Schwarz and Martin G. Weiß. In: *International Congress on Information and Communication Technology* **(ICICT)**, 2024 ([BSW24])
- Chapter 4
  - Rotationsermittlung von Bauteilen basierend auf neuronalen Netzen.
    Sebastian Bock. Master's Thesis, 2017 ([Boc17])
  - A Proof of Local Convergence for the Adam Optimizer
    Sebastian Bock and Martin G. Weiß. In: *International Joint Conference on Neural Networks* **(IJCNN)**, 2019 ([BW19a])
  - Non-Convergence and Limit Cycles in the Adam Optimizer.
    Sebastian Bock and Martin G. Weiß. In: *International Conference on Artificial Neural Networks* **(ICANN)**, 2019 ([BW19b])
  - An improvement of the convergence proof of the ADAM-Optimizer.
    Sebastian Bock, Martin G. Weiß and Josef Goppold. In: Clusterkonferenz, 2018 ([BWG18])
- Chapter 5
  - Local Convergence of Adaptive Gradient Descent Optimizers.
    Sebastian Bock and Martin G. Weiß. In: *https://arxiv.org/abs/2102.09804* **(Arxiv)**, 2021 ([BW21])

The publications below describe work that is loosely related to this thesis but not described in the thesis:

- Rotation Detection of Components with Convolutional Neural Networks.
  Sebastian Bock and Martin G. Weiß. 2020 ([BW20])

Acknowledgements

First of all I want to thank my supervisors Prof. Dr. Christian Wolff and Prof. Dr. Martin G. Weiß. Both support and advice me in writing this thesis. In particular, I would like to thank Prof. Dr. Martin G. Weiß for discovering me during my mathematics degree and enabling me to write my dissertation.

Thanks also go to Baumann Automation GmbH, which made it possible for me to write the dissertation with an uncommon half-time job in the image processing department.

I also want to thank everyone involved in research cluster RAKS, research project LEAP and research project PRISMA. The cooperation and support was an important building block of this work.

Most of all I want to thank my girlfriend Carina and my parents Petra and Jürgen. Even if they do not understand what I am doing, they were interested and asked questions. Thanks for supporting me.

1 Introduction and Background

Nowadays it is quite easy to train neural networks. With frameworks like Tensorflow [AAB+16] or PyTorch [Pas+19], even beginners can train networks in just a few lines of code. Other methods of artificial intelligence (AI) can also be easily integrated, and that is a good thing, in order to bring these promising technologies into many areas of everyday life. Many people already use AI, for example in image processing or speech recognition, without necessarily even knowing the meaning of AI. And as AI is expected to help solve many of the main problems of our time (i.e. climate change [Mah+22] or demographic change (see [Mio+17] as an example in healthcare)), such intersections will become more common. In principle, it is not a problem if the user is not an AI expert. After all, not every car driver knows the functional principle of an engine. However, programmers should have enough knowledge about AI to be able to assess the consequences and dangers for users, and this is where a problem arises. Many applications that already work in practice have hardly been researched mathematically. One example is the Adaptive Moment Estimation (ADAM) optimizer from [KB15], which is often used to train neural networks but has no proof of convergence and even has a counterexample [RKK19] several years after its publication.

Such differences between application and research should not be underestimated in such a fast growing field as neural networks. It is primarily the rapidly growing computing power that has made more and more applications possible in recent years. Nevertheless, the theory behind it should not be neglected, since, for example, security-relevant tasks (i.e. autonomous driving) are also processed with it.

Therefore, in this work we will first start with a topic born from practice, 3D pose estimation using neural networks. In the course of the work, more and more mathematical gaps in the theory have become apparent, so that in the second part of the thesis we will mainly consider the convergence behavior of individual gradient descent algorithms. Since this work contains many different topics, we would like to formulate various research questions and briefly discuss the contribution made.

Research question 1: *Are neural networks suitable for determining rotations or poses of objects in industrial environments?*

In Chapter 2 we observe pose estimations of objects in automated production conditions. Here we focus first on the rotation in the two-dimensional case and then on the

three-dimensional case with three degrees of freedom. We create an overview of suitable architectures and present noticeable tricks. The expectation of this research question is to obtain a more cost effective tool for pose estimation. Furthermore, a better generalization is expected, which is helpful in case of disturbances in the images.

Research question 2: *How can we explain the U-shapes that appear in tests of neural networks with different noise?*

In Chapter 2 and related work (see [Kra21] and [Sta21]), we discovered parabolic curves when testing networks with different perturbations. In Chapter 3, we trained neural networks with a fixed noise rate σ_0 (mostly we use a Gaussian noise). The expectation when testing with different values for σ would be to get the smallest test error with the smallest disturbance or with the trained disturbance. However, this was not the case in the experiments shown. We then developed theories and experiments to describe this behavior.

Research question 3: *How can we describe the convergence behavior of the ADAM optimizer?*

While trying to better understand the training behavior of neural networks in Chapters 2 and 3, the weaknesses of the convergence proof of one of the most important optimizer in the field of neural networks were again noticed. We address this issue in Chapter 4, where we prove local convergence and global non convergence in batch mode. At the end of Chapter 4 we also give a convergence proof for the ADAM in incremental mode.

Research question 4: *Can the previously developed methods be applied to other adaptive gradient descent methods?*

In Chapter 5, we address the question of how we can apply the insights gained in Chapter 4 to other adaptive gradient descent methods. In doing so, we can achieve similar results for Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMSProp), Adaptive Gradient Algorithm (AdaGrad), AdaDelta and AMSGrad.

2 Learning Pose - LEAP[1]

2.1 Introduction to LEAP

A rigid object in a three-dimensional space has six degrees of freedom. Three coordinates describe the position (x, y, z) and the other three the rotation (α, β, γ) of the object in space. To relate such coordinates to others, you need an underlying coordinate system. In applications like these, the world coordinate system is often agreed upon. This is an arbitrary but fixed coordinate system in our space. Only e.g. robot applications require special coordinate systems in flange or gripper.
With a given coordinate system and appropriate coordinates we can describe the pose of our object. Are all coordinates equal zero, we are talking about the zero position of an object. As mentioned above, the coordinates can be divided into a translational and a rotational part. Together and taking the zero position into account, this defines the *6D-Pose* of an object.

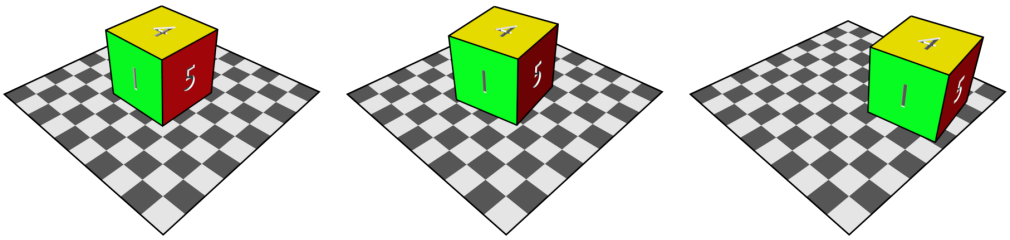


Figure 2: reference pose (left); rotation (middle); rotation and translation (right)

In the research project Learning Pose (LEAP) we are addressing the *6D pose estimation* problem. Thereby we try to find a transformation between reference pose and object pose by using image data. This will be realized by machine learning algorithms. We can divide this problem into two subproblems. On the one hand we have the determination of the translation. In particular, we are looking for the center coordinates of the object, around which the rotation is performed. On the other hand we have the determination of the rotation. Here we are looking for the rotation around the center of the object, which rotates our object back to the reference position. In this work we will discuss only the rotation estimation and therefore a *3D pose estimation*.

The problem is applied in robotics, aerospace and many automated productions, which makes the problem increasingly important for many industries. Currently, classical image processing is often used for this purpose because it provides high accuracy and results

[1]This chapter expands the results and presents proofs that are referenced in [BSW24].

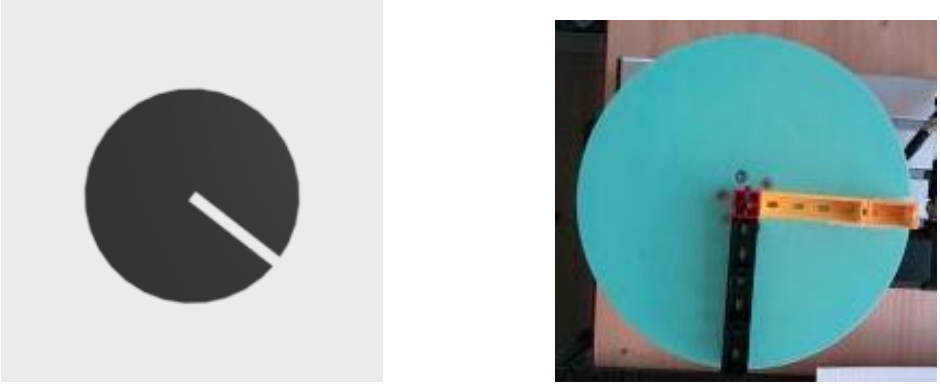
in the millisecond range. However, this requires expensive and highly qualified personnel to select and parametrize the algorithms. Furthermore, such systems are strongly dependent on environmental influences. Therefore, a high effort is often made to avoid e.g. solar radiation or headlights of forklifts. In some systems, however, this is not possible at all, because e.g. at collaborating workplaces, people need access to the component or they suffer health damages when using ultraviolet light. In addition, every small change to a component requires an expert to readjust the system.

Therefore, the research project LEAP aims to solve the problem of 6D pose estimation with machine-learning algorithms. The main focus is on the aspects of accuracy and cost savings compared to classical image processing. The training of neural networks requires labeled data, which can be created with little qualification. A complete automation by a turntable would be desirable, which is why we carry out a corresponding experiment in Section 2.6. In case of problems in pose determination or modified components, these pictures can be used for retraining. The goal of the project is to generate a pre-trained network architecture for common components of a company. This architecture will then be trained by using train images of a concrete object. The user should neither have to adapt the architecture of the network nor the hyper parameters of the training procedure.

Furthermore, neural networks are expected to generalize better. Due to the possibility of training noisy input data, such disturbances are known to the networks. This is why tasks are often considered which are very confusing or unclear. In [Xia+18] and [ZSI19], for example, highly crowded tables are used where the 6D pose of a specific component is to be identified. In [FDB15], rotations of everyday images are predicted, which are not always clearly defined and in [Liu+16] and [Li+18], rotations of ships are predicted using satellite images. All these predictions are very inaccurate because often bounding boxes are used to simplify the objects, we do not have an exact label or metrics are used which are subsequently classified by a threshold depending on the diameter of the object (see [Hin+12a]). This approach is nevertheless useful in the papers, as the task often does not allow for a more detailed analysis. Here, however, the present work differs from those mentioned above; our task in the industrial environment is very clear and little disturbed by environmental influences as described above.

2.2 Dataset

To prove the ability of predicting rotations of two dimensional objects, we create a simple dataset with clear contours and no disturbing stuff beside (comparable to industrial



(a) Example image for the dataset slitdisc (b) Example image for the dataset turntable

Figure 3: Example images

conditions). Therefore we use a nearly black disc on a nearly white background. To identify the rotation, the disc has a white slot (see Figure 3a). Each image has the size of 200×200 pixels. In the train set we have 3600 images evenly distributed over 360°. Thus, we have an image available every 0.1°. In the test set, the angles are randomly selected.

For the above experiment, we also created a real experiment. For this, we used a camera that is pointed at a turntable. On this turntable there is a coordinate system, which clearly defines the rotation of the turntable (see Figure 3b). We resize the images to 175×166 pixels and intentionally omit the robot cell lighting here to get tremendous lighting effects. More details will be given in Section 2.6.

For the experiments regarding the 3D pose estimation, we use a dice. Each side of the dice has a unique number engraved and is colored in a unique color (see Figure 4). All numbers are painted black. Moreover, we simulate six different camera positions in this setting. The working distance to the dice is always constant, only the position changes (see Figure 4). This setting corresponds to a robot-guided camera. In industrial applications, a constant focal length is often used, which means that the working distance must be constant in every image position. Possible camera positions are therefore completely on a sphere around the dice. In addition, this sphere is limited by environmental conditions (e.g. underground). However, we will neglect these in the following experiments.

The cube can rotate $\pm 60°$ in each axis. The test set has a step size of 8° and therefore 4095 different poses. In the test set we use a random pose in the range of $\pm 60°$ in each axis. Background disturbances or lighting inhomogeneities were avoided.

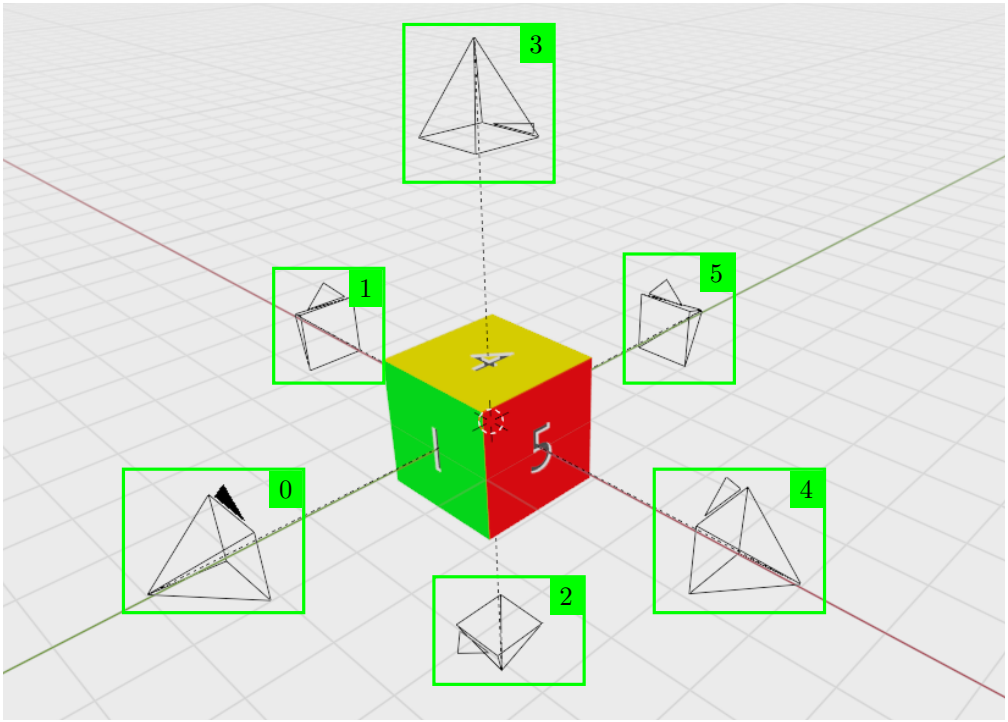


Figure 4: Dice and the camera positions

2.3 Network Architectures

In the following neural network approaches, we use different established neural network architectures. We chose Residual Network (ResNet)50 [He+16], Mobilenet [How+17] and MobilenetV2 [San+18] to cover very deep architectures (see Table 1).

Table 1: Architecture settings

Name	Parameters in million	Depth (amount of layers)
VGG16	57.7	24
VGG mod	47.7	25
ResNet50	233.7	180
Mobilenet	88.7	91
MobilenetV2	108.	159
PoseCNN	91.0	32
CNN	46.6	15

ResNet50 and the two Mobilenet architectures also differ significantly in the number of parameters, which can also reveal further differences. Note that the majority of the parameters in the Mobilenet architectures come from the dense layers added at the end.

Since these networks may be too complex for this simple task, we test also a VGG16, a modified VGG (abbr. VGG Mod) and a simple Convolutional Neural Network (CNN). Since the latter two architectures are self-designed, we briefly discuss them in more detail. Both are produced using blocks (x, y), where x describes the number of filters and y the number of convolutions. After each block, a MaxPooling layer is applied. With this definition, CNN is generated by $(32, 2), (32, 2)$ and VGG Mod by $(32, 2), (64, 2), (128, 2), (256, 2), (512, 2)$. Subsequently, both architectures receive dense layer, with $(8, 4)$ and $(1024, 512, 4)$ neurons, respectively. As an activation function, we use the *'relu'* function only in the last layer a linear function.

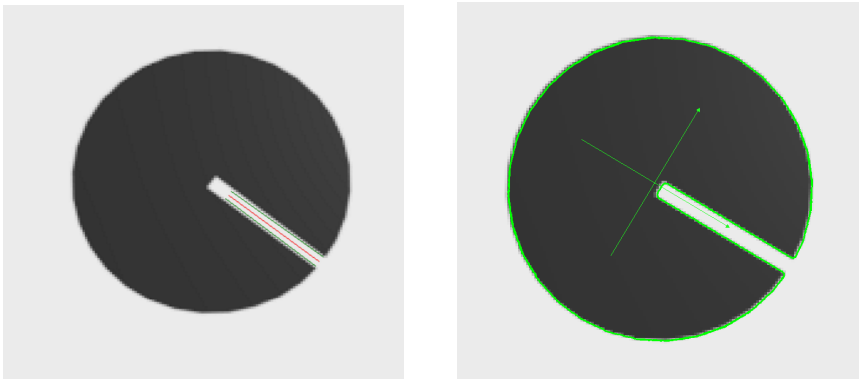
We also extract the rotation part from the PoseCNN [Xia+18] and test it under the name PoseCNN. This architecture was designed in [Xia+18] especially for rotation detection and is therefore tested as well.

In Section 2.5, other architectures were used due to the number of cameras. These are specifically explained at the beginning of that section.

2.4 Two Dimensional Object with One Degree of Freedom

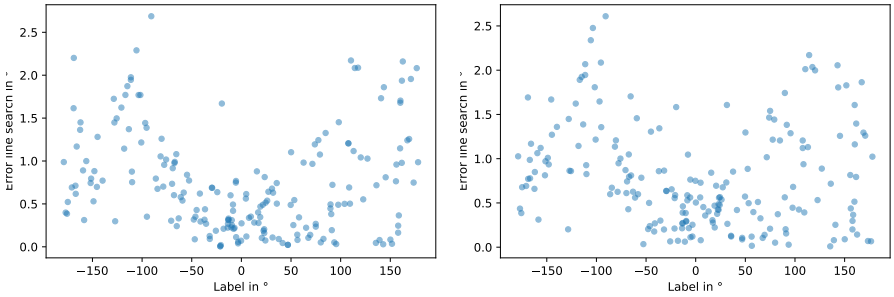
2.4.1 Traditional Image Processing

We would like to compare our results not only neural networks internally but also with currently used image processing algorithms. Thus, the possible industrial benefit shall also be explored. For this purpose we use a rudimentary method, which is especially adapted to the shape of the slitdisc (in the following called line search) and a more open pattern matching method, where only strong contrasts are necessary (in the following called PatMax$^{\text{TM}}$ [SWW]).

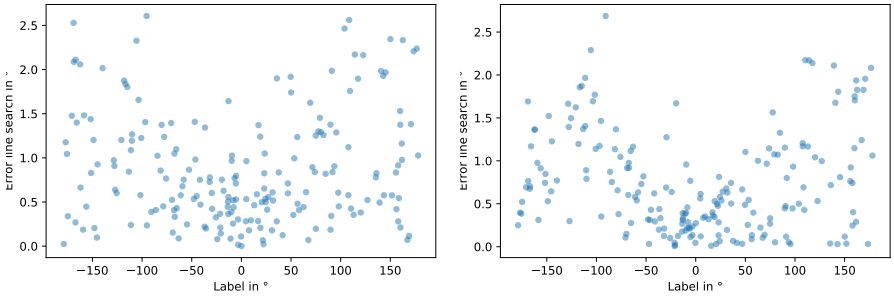


(a) Line search algorithm (b) PatMax algorithm

Figure 5: Traditional image processing algorithms



(a) Error with Gaussian noise $\sigma = 0$ (b) Error with Gaussian noise $\sigma = 0.01$

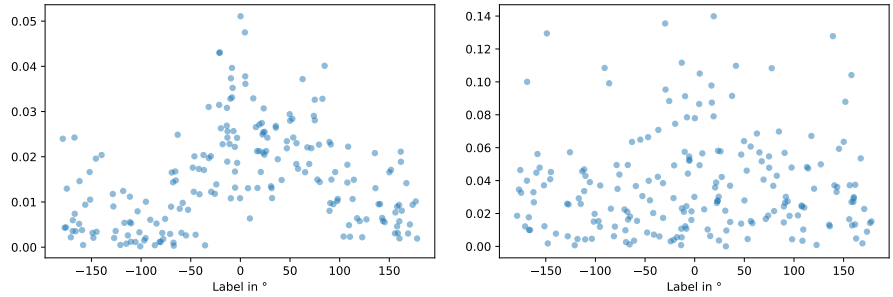


(c) Error with Gaussian noise $\sigma = 0.05$ (d) Error with SnP noise $p = 0.005$

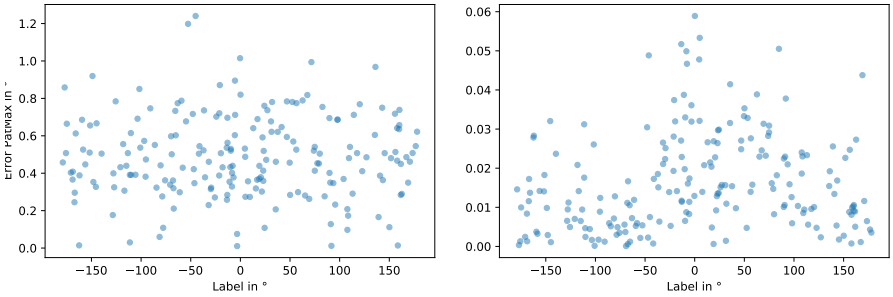
Figure 6: Error distribution with line search

In the line search algorithm we are searching for two lines around the black circle. One of these lines should have the polarity black on white and the other exactly opposite polarity. Thus we find the notch of the disc and can calculate the angle via the middle line of these two lines. Visually, this is described in Figure 5a. The two green lines are the result of the line search with the respective polarity and the red line is the measured center of the notch. The system was calibrated once to zero degrees and then tested.

In the PatMax algorithm, we use the pattern matching algorithm from Cognex Corp.. Here, one must first train an initial contour, which is then searched for. The contrast of the image is the most important factor for the PatMax. In Figure 5b, for example, the found contour is drawn in green. Again, only one image was trained, which was then used to calibrate the system.



(a) Error with Gaussian noise $\sigma = 0$ (b) Error with Gaussian noise $\sigma = 0.01$



(c) Error with Gaussian noise $\sigma = 0.05$ (d) Error with SnP noise $p = 0.005$

Figure 7: Error distribution with the PatMax

In our experiments we use 200 randomly rotated slitdisc images with Gaussian noise ($\sigma \in \{0, 0.01, 0.05\}$) and Salt and Pepper (SnP) noise. In Figure 6 and 7 we can see the error distribution as a function of the angle. It is noticeable that the Gaussian noise hardly influence the measurements (see Table 2). The experiments with the PatMax algorithm showed better rotation detection than the line search algorithm. Furthermore, no systematic outliers were detected. However, the Gaussian noise increases the average error enormously from 0.015° to 0.488° (see Table 2). This can be attributed to the contrast dependence mentioned before. The SnP noise has hardly influenced both methods.

Table 2: Average error in traditional image processing

	$\sigma = 0$	$\sigma = 0.01$	$\sigma = 0.05$	SnP $p = 0.005$
Line search	0.732°	0.797°	0.873°	0.739°
PatMax	0.015°	0.037°	0.488°	0.016°

These results are now compared with the experiments based on neural networks.

2.4.2 Neural Networks Approach

To tackle the rotation estimation with neural networks we use the architectures described in Section 2.3. We train each architecture with two different losses and four different disturbances on the train images. In addition, each variant was trained twice to minimize unwanted one-off effects. Labels were given in quaternions to prevent modulo effects. Mean Squared Error (MSE) and the PoseLoss (PLOSS) [Xia+18] were used as loss. Training disturbances are 'Without noise', SnP noise with $p = 0.005$ and Gaussian noise with $\sigma = 0.01$ and also random $\sigma \in [0.0, 0.05]$.

The detailed test results can be found in Appendix C.1. Here we only summarize the most important findings.

1. No matter which noise was tested with, the CNN architecture achieved the lowest absolute error. The PoseCNN architecture delivered a similarly good result, which is always among the best ten variants several times.

2. The line search method is always less accurate than the best networks. PatMax, on the other hand, is more accurate for smaller disturbances; only for large disturbances are both variants comparable.

3. Deep networks such as ResNet50 or Mobilenet, on the contrary, always scored the worst.

4. With the best losses, there are no striking differences. Only when testing without noise, a weakness of the PLOSS can be seen.

5. Compared to traditional image processing, many networks can perform better than the line search attempt. So the networks are always preferable to this. In comparison with the PatMax, comparable networks only emerge at a very high level of noise.

Especially in the tests with a large noise, the experiments with neural networks were competitive with traditional image processing. In addition, known noise can be trained directly with neural networks, which is difficult with traditional image processing. Thus, the pose can also be detected in case of disturbances in industrial applications such as metallic surfaces or inhomogeneous background. Already in these experiments, the U-shape phenomenon was noticed, which is examined in more detail in Chapter 3.

This experiment is very relevant for industry e.g. in conveyor belt applications where only one direction of rotation is unknown or specifically the gripping of choke plates where the positions X, Y, Z and the rotation around X and Y is known by the robot

Table 3: Summary of the most important findings of the neural networks approach

Test noise	Without noise	Salt and pepper	Gauss $\sigma = 0.01$	Gauss random σ
Best architecture $\varnothing abs$ in deg	CNN 0.210°	CNN 0.300°	CNN 0.348°	CNN 0.349°
Top 5 architectures	2× CNN 3× PoseCNN	2× CNN 1× PoseCNN 1× Mobilenet 1× VGG Mod	2× CNN 1× PoseCNN 1× Mobilenet 1× VGG Mod	2× PoseCNN 2× VGG Mod 1× CNN
Top 10 architectures	5× CNN 5× PoseCNN	3× CNN 3× PoseCNN 2× Mobilenet 1× ResNet50 1× VGG Mod	4× CNN 3× PoseCNN 2× VGG Mod 1× Mobilenet	3× CNN 4× PoseCNN 3× VGG Mod
Top 5 generalizer	3× CNN 2× PoseCNN	4× PoseCNN 1× CNN	4× PoseCNN 1 × CNN	5× PoseCNN
Top 10 losses	6× PLOSS 4× MSE	6× MSE 4× PLOSS	6× PLOSS 4× MSE	6× PLOSS 4× MSE
Worst 10 losses	10× PLOSS	6× PLOSS 4× MSE	5× PLOSS 5× MSE	6× PLOSS 4× MSE
Worst 10 architectures	6× ResNet50 4× Mobilenet	4× ResNet50 1× VGG Mod 2× Mobilenet	6× ResNet50 3× VGG Mod 1× Mobilenet	5× VGG Mod 4× ResNet50 1× Mobilenet

position but the Z rotation has to be determined. For this reason, we will perform this experiment in Section 2.6 with a real data set. The focus will be on a minimalist image processing setting, so that image processing experts are not needed. For the time being, however, we want to stay with artificial data sets and consider objects with three degrees of freedom.

2.5 Three Dimensional Object with Three Degrees of Freedom

With the knowledge of the rotation experiments, we will now look at 3D pose estimation. Thereby we do not use distance sensors or similar, the image format does not change for three-dimensional objects. The object now has three degrees of freedom but our image remains in $(width, high, 3)$. Note, the task could also be solved with grey scale images due to the numbers on the dice. However, this possibility is not considered in more detail so that objects with purely color differences are not excluded.

Based on the results of the rotation experiments, we will limit ourselves to smaller architectures and only use the MSE as a loss in the next experiments. Note that the ShapeMatch-Loss (SLOSS) [Xia+18] can be useful for objects with symmetries. However, since the objects used do not contain any symmetries, this loss was omitted. In the following experiments, we will mainly focus on the different input possibilities and the benefits of multiple cameras.

2.5.1 Multiple Images as Input

If we want to identify a pose of an object from multiple cameras, we have several ways to tackle this problem. For all follows, there is one pose of the object and $c \in \mathbb{N}$ different cameras. Every pose of the cameras is well known. This also reflects the industrial reality. Either it is constructively known through permanently installed cameras or indirectly through the robot and cameras attached to the gripper. In both cases, the camera pose can be determined. This means that all camera poses are reproducible. We obtain an image with dimension $(width, high, 3)$ for each camera pose. In terms of network architectures and multiple input images, we have the following two options.

- *Stack* (One image): We combine all images into one image by extending the dimension of the image.

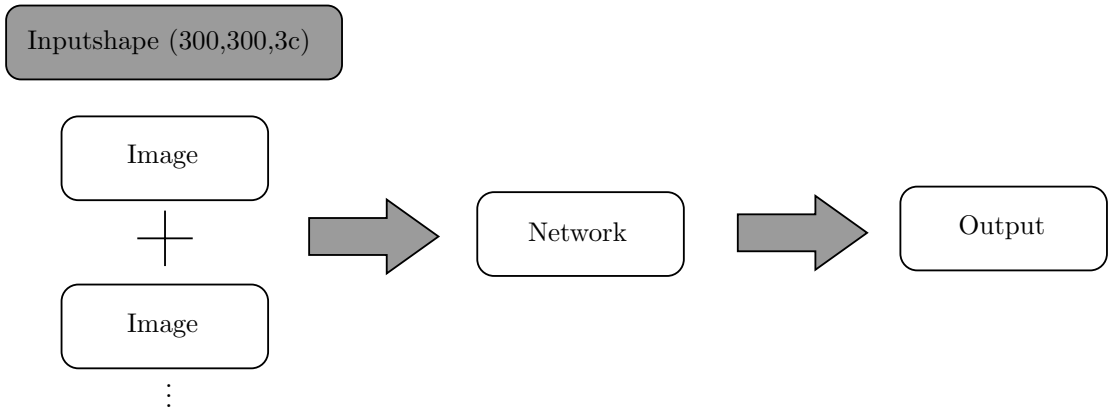


Figure 8: Stack architecture

- *Concatenate* (All images separately): Each input image initially receives its own neural network, but these are constructed in the same way, since the same task is only performed with a different camera pose. Then these outputs are concatenated to each other and calculate with another network the object pose.

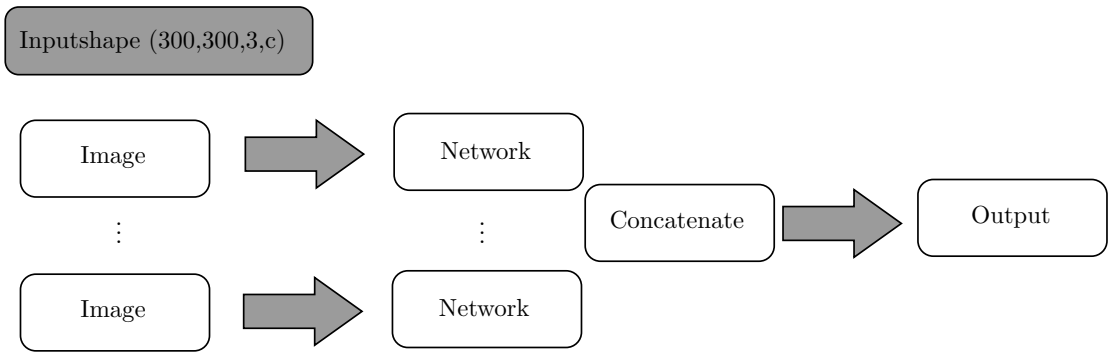


Figure 9: Concatenate architecture

Both architectures are strongly influenced by the number of input images. Changing the number of input images is only possible with retraining. A new camera pose can only be learned by retraining.

At this point we would like to point out that we did not consider the camera pose as a direct input for an architecture. Possible architectures would then determine the object pose on the basis of one image and one camera pose. With a constant camera pose, however, this corresponds to the procedure in Section 2.4 and random camera poses are nonsensical in an industrial environment.

As could already be seen in Section 2.4, very deep architectures were not advantageous. Deep architectures such as ResNet50 [He+16] or VGG16 [SZ15] performed very poorly in comparison. They behaved similarly in the 3D pose estimation (see Table 4), so to save time we neglect these architectures in the comparisons in this section.

Table 4: Result of the tests with some deep architectures.

Architecture	Avg. train error in deg	Avg. test error in deg	Architecture type
ResNet	32.00°	29.96°	*Stack*
VGG16	32.00°	29.94°	*Stack*
ResNet	32.02°	30.05°	*Concatenate*
VGG16	32.00°	29.94°	*Concatenate*

We focus on the well-performing CNNs. In the experiments, we look at a smaller (called small cnn) and a larger (called big cnn) architecture. Using the block notation from Subsection 2.4.2, the small cnn is defined by $(32, 3), (64, 3), (64, 3)$ and the big cnn by $(32, 3), (64, 3), (128, 3), (256, 3)$. In the *stack* variant, small cnn receives an additional pooling layer with a $(4, 4)$ kernel and both architectures have three dense layers with $1024, 512$ and 4 neurons after flattening. In the *concatenate* variant, there are the above-

mentioned CNNs per input image. The resulting output is then concatenated and folded again with $(256, 3)$. The neurons are then flattened and the network receives three more dense layers with $1024, 512$ and 4 neurons respectively. The number of neurons in the different variants *stack* and *concatenate* thus differs essentially only by a convolution layer after the concatenating and the amount of networks after each input image. Labels are again given in quaternions.

In this chapter, we only discuss the main aspects of the experiments. Details can be found in Appendix C.2. It was initially noticeable that the *stack* architecture performed worse than the *concatenate* architecture with a comparable number of parameters (see Table 5). Note that *big/small* refers to the depth of the network. The strongly varying number of parameters is mainly due to the different convolution output.

Table 5: Trainable parameters with two cameras

Architecture	Parameters in million
small / stack	4.9
small / concatenate	61.3
big / stack	60.7
big / concatenate	11.2

Figure 10 shows the mean abs error as a function of the architecture. The test images were undisturbed and the colored marking shows the disturbance of the train images.

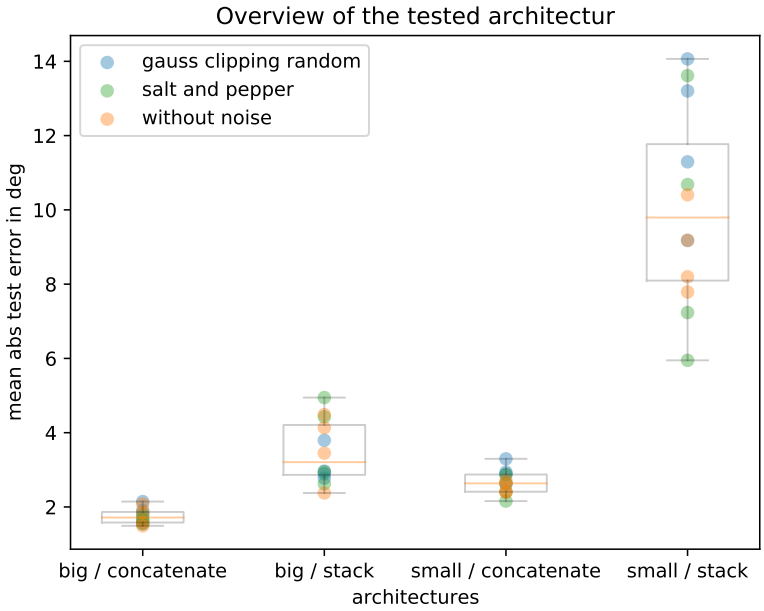


Figure 10: Architecture overview

Specifics between the different noises could not be found. The *concatenate* variant is clearly preferable to the *stack* variant. This becomes clear when one looks at the average error of the *small/concatenate* variant and the deeper network *big/stack*. The average error of the *small* variant with *concatenate* was 2.664° whereas the *big* variant with *stack* has an average error of 3.486° (see Figure 10). We were able to achieve the smallest error with the *big/concatenate* architecture and two cameras. Here, the average absolute error was 0.491° in contrast to 8.0° train grid.

An initial single viewing of the images in the network (concatenate variant), produces a minor loss.

Remarkably, the test error does not decrease with a higher number of cameras. For example, if we take entry 1 and 4 from Table 21 in the appendix (see an extract in Table 6), we can see that the absolute error of 1.492° deteriorates slightly to 1.589° even with four cameras. Also in the absolute error in the individual axes, is no advantage recognizable. Since we only use cameras in the X and Y directions in the experiments with two cameras and only the third camera is in the Z direction, the expectation was that the error in the Z-axis would decrease. But again, no significant improvement could be observed.

Table 6: Extract from Table 21

Architecture	cameras	abs error in deg	abs error X-axis in deg	abs error Y-axis in deg	abs error Z-axis in deg
big cnn concat	2	1.492	0.489	0.512	0.491
⋮					⋮
big cnn concat	4	1.589	0.656	0.439	0.494

Applications based on the Gauss-Newton method unfortunately did not yield useful results and are therefore not compared. Other approaches would only have been possible with additional equipment (3D scanner or similar) and were therefore not used.

Based on this result, we created a more detailed experiment in the next subsection regarding multiple cameras.

2.5.2 Multiple Cameras

To test the effectiveness of multiple cameras, we create an experiment with the same framework. We use big cnn *concatenate* as architecture and no noise is used. Train and test set is constructed as in Section 2.2. To avoid one-time effects, we also train each network twice independently.

We create experiments with cameras $[0], [0, 1], [0, 1, 2]$ and $[0, 5]$. Cameras $0 - 2$ are to test the different axes and camera 5 is like camera 0 on the X-axis and is thus to represent a redundant case (see Figure 4). In the experiments (see Table 7), we can see that run 1 and run 2 of each experiments are always close to each other, indicating that there are no exotic one-time effects. As expected before, the network with cameras on the X-, Y- and Z-axis is the best. However, the difference e.g. to the worst network is only $0.284°$. Direct projection to the respective axis error is not possible. E.g. experiment with cameras $[0, 1, 2]$ contains a camera in the Z-axis. Nevertheless, the average error from both runs is lower for the experiment with the cameras $[0, 1]$.

For industrial applications, this means an estimation between the increased accuracy and the resulting increased time, e.g. due to the increased training time, the increased cycle time due to the robot movement between the individual positions or the commissioning of the more numerous cameras.

Furthermore, care must be taken not to use redundant cameras. As can be seen experiment with cameras $[0, 5]$ in Table 7, this experiment performed the worst. Not considered

Table 7: Camera experiment

cameras	l2 mean	l2 std	abs error X-axis in deg	abs error Y-axis in deg	abs error Z-axis in deg	abs error in deg	train time (h)
0, 1, 2	1.119	1.498	0.710	0.474	0.495	0.560	4.45
0, 1, 2	1.136	1.398	0.594	0.426	0.672	0.564	4.40
0, 1	1.163	1.312	0.744	0.472	0.522	0.579	2.97
0, 1	1.272	1.107	0.833	0.498	0.577	0.636	2.96
0	1.406	2.211	0.760	0.685	0.716	0.720	1.49
0	1.566	2.695	0.889	0.722	0.802	0.804	1.49
0, 5	1.636	2.451	0.828	0.881	0.840	0.850	2.92
0, 5	1.647	2.651	0.839	0.843	0.851	0.844	2.91

in the experiments were symmetries in the objects. For such objects, multiple camera positions can have an enormous impact on accuracy.

2.6 Turntable Application

In order to test the results in real conditions, we create a suitable data set for a real 2D rotation within the bachelor thesis of [Orh22]. We used a camera[2] that is attached to a robot.[3] The camera is looking at a turntable [igu22] with a two-dimensional coordinate system ($100\ mm \times 100\ mm$) on their 200 mm plate. The turntable has a reverse backlash of $< 0.5°$, so we can define the first accuracy limit at $0.5°$.
The camera takes 1280×720 pixel images. We resize them to the size 320×180 pixel and crop them to 175×166. By intentionally making the lighting conditions vary greatly, we get a natural disturbance in the image (see Figure 11). This is much more pronounced than in most industrial applications. The pixel accuracy for a 175×166 pixel image is $0.6903°$. Below that we are in the sub-pixel range. Thus, a network with an average accuracy of $< 0.5°$ or $< 0.7°$ would be desirable.

As in the section above we compare the combinations of architectures $\in \{$*'VGG16'*, *'PoseCNN'*, *'ResNet50'*, *'Mobilenet'*, *'MobilenetV2'*, *'VGG mod'*$\}$ and artificial noise $\in$ $\{$*'Without noise'*, *'Salt and Pepper'*, *'Gaussian noises with* $\sigma_0 = 0.01$*'*$\}$. In addition, we use each experiment in a regularized version and without regularization. Therefore, we regularize the last layer in the corresponding architecture. The hope is to minimize errors in redundant values e.g. in the constant Y-rotation.

[2] Intel® RealSense™ Depth Camera D455 [Int22]

[3] The robot was not important for this application, since the component did not move and no further processing took place. The camera could just as well have been fixed to a rigid fixture.

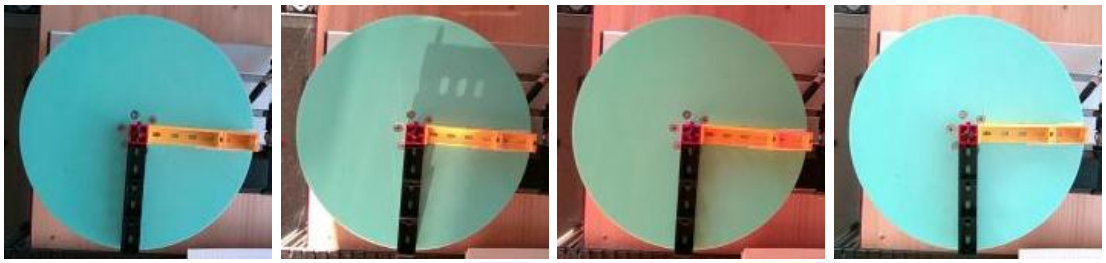


Figure 11: Different light influences in the data set

Table 8: Best networks on the turntable dataset with different light influences

Architektur	Exp. regularized	l2 mean	l2 std	abs error in deg	max abs error Z-axis in deg	Noise
VGG16	False	0.323	0.328	0.356	4.265930	Without noise
VGG16	False	0.398	0.425	0.432	7.267123	Without noise
VGG16	True	0.431	0.304	0.475	3.327395	Without noise
PoseCNN	False	0.445	0.354	0.484	2.897656	Gaussian noise with $\sigma_0 = 0.01$
PoseCNN	True	0.433	0.351	0.485	1.733418	Salt and Pepper
VGG16	False	0.443	0.406	0.486	3.981858	Salt and Pepper
ResNet50	False	1.792	21.363	0.565	5.945046	Salt and Pepper
VGG16	True	0.537	1.003	0.566	16.901430	Salt and Pepper
CNN	True	0.565	1.218	0.574	26.469033	Salt and Pepper
PoseCNN	True	1.160	15.128	0.576	22.047687	Without noise
Mobilenet	False	1.164	15.122	0.582	3.349986	Without noise
PoseCNN	False	0.536	0.495	0.598	3.259047	Salt and Pepper

Table 8 shows only the best combinations for this experiment sort by the absolute error of X-, Y- and Z-axis in degree (The complete table can be found in the Appendix C.3.). Again, the VGG architectures are the best for this task. In the top five are only VGG architectures (four times pure VGG16, one time VGG based PoseCNN). The average absolute error in the top six is below 0.5° and thus smaller than the accuracy of the turntable. For a typical pick-and-place application and the 100 mm object used, this would be a displacement of the central gripping position of $\leq$ 0.436 mm in the average. Such a tolerance of error is permissible in most pick-and-place applications. The maximum displacement in the top one combination would be 3.730 mm and in the best PoseCNN combination 2.530 mm. There are applications in which such a maximum value is too high, but in our application (well grippable object and wide open gripper jaws) it would be completely sufficient. The required accuracy depends on the application

and can therefore only be considered here as an example. The best 15 combinations also predict in the subpixel range with an abs error of less than 0.7°. If we restrict ourselves purely to the Z-coordinate, there are even 20 combinations.

However, we also see that the ResNet50 architecture delivers good results, which was often not the case in the previous experiments. No ResNet combination, has an average absolute error in the Z-axis greater than 2°. This also suggests that the task was more complex than the previous ones. ResNet often did not learn anything in the other experiments because the architecture was too deep and the task was not complex enough. The task seems too difficult for the Mobilenet architectures and their few parameters. They almost exclusively form the worst combinations in this experiment (see Table 22). Two VGG16 combinations still stand out with an average absolute error greater than 88°. However, these combinations have not learned anything through training (see the systematic error in Figure 12) and can therefore be neglected. Our self-created CNN was also able to accomplish the task relatively well. It was only noticeable that many errors occurred close to 0°. Systematic errors can therefore not be completely ruled out.

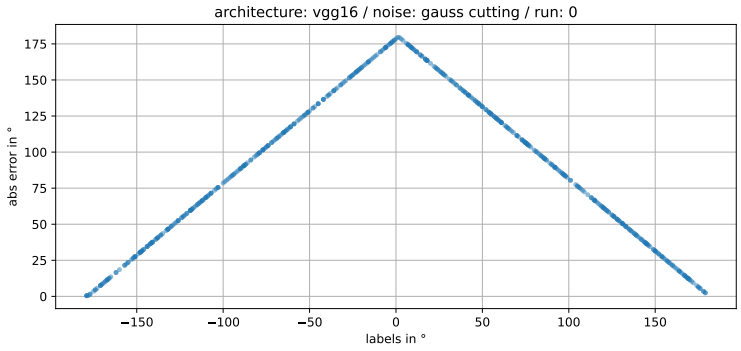


Figure 12: Broken VGG16 combination

In Figure 13 we see all the architectures lined up again in a box plot. Outliers above 20° were neglected. We can see the good performance of the VGG architectures mentioned above from the low average error (orange line). In addition, the clear outliers of the Mobilenet architectures become obvious. When comparing regularized and not regularized combinations, the first thing to notice is that there are more not regularized combinations among the outliers (See red dots e.g. in the Mobilenet architecture in Figure 13).

In a direct comparison of the mean values, we obtain an average advantage of 0.7238° across all architectures for the regularized variants (see Table 9). Although there can also be minimal deterioration (see MobilenetV2 in Table 9), regularization is still to be

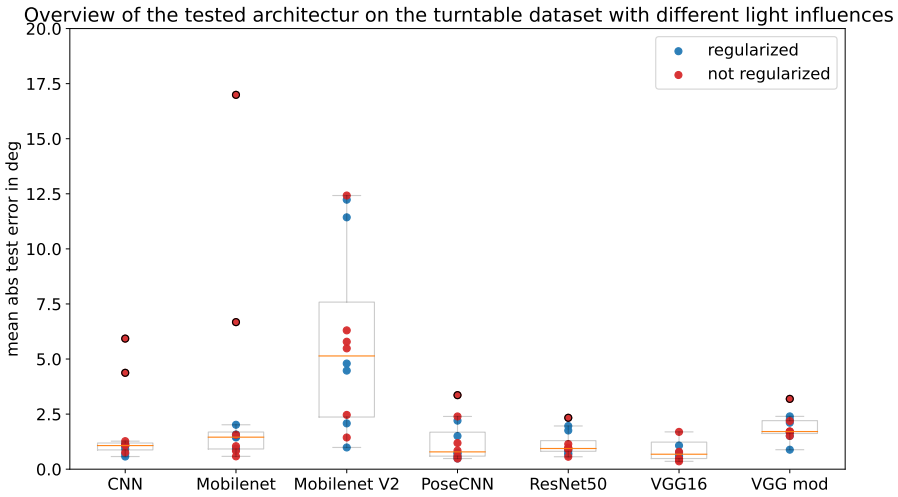


Figure 13: Architecture comparison

Table 9: Regularization vs. no regularization

Architecture	mean abs error with regularization	mean abs error without regularization
VGG mod	1.8015°	2.0008°
VGG16	15.3852°	15.4242°
ResNet50	1.1451°	1.1437°
Mobilenet	1.3866°	4.6132°
MobilenetV2	5.9997°	5.6490°
CNN	0.9098°	2.4237°
PoseCNN	1.0374°	1.4773°

preferred due to the high average advantage.

In terms of artificial disturbances, "Without noise" and "SnP" were convincing. Since the test data of Appendix C.3 are undisturbed, "Without noise" is no surprise. The reason for the undisturbed test data is that in a real application no artificial noise is applied to the input image. Unfortunately, "SnP" does not show any decisive advantage over "Without noise", which means that similarly good results are achieved in this application even without artificial noise.

2.7 Conclusion

In summary, we were able to give an overview of rotation detection by neural networks in this chapter. We were able to give some architectures and combinations, which partly outperform even classical image processing. The pose estimation with one degree of freedom was then rounded off with a real experiment in which we were able to slightly undercut both the pixel accuracy and the machine accuracy.

We also examined three-dimensional objects with three degrees of freedom. Here we showed experimentally that a *concatenate* architecture with several cameras is to be preferred and that redundant cameras can even worsen the results.

In all these results, some U-shapes were created when testing with different σ in Gaussian noise. This phenomenon will be examined in more detail in the next chapter.

3 U-shape[4]

3.1 Problem Formulation

In this work as well as in the unpublished bachelor thesis of Philipp Kramer [Kra21] using neural networks and Andreas Stautner [Sta21] using Support Vector Machine (SVM) the following problem was studied, with similar effect.

We consider a grey scale or RGB image of a slit disc (see Figure 3a) and want to determine the angle relative to a reference position. The training images $I_1, \dots, I_N$ with rotation $\varphi_1, \dots, \varphi_N$ are generated from a reference image $I \in \mathbb{R}^{N_x \times N_Y}$ using rotation and additive Gaussian noise with zero mean and standard deviation σ_0. Then a neural network or SVM was trained with these data to predict the angle φ, resulting in a function $f_{\sigma_0} : \mathbb{R}^{N_x \times N_Y} \times \mathcal{W} \to \mathbb{R}$, $(I, W) \mapsto \varphi$ with the weights collected in W. The subscript σ_0 emphasizes the noise model on the data.

Then the regression function was tested on data generated with different standard deviations. This approach corresponds to a machine learning algorithm trained on synthetic data, and then tested with data from sensors with different noise. The MSE was plotted against σ, resulting in a U-shaped curve in some – not all – cases.

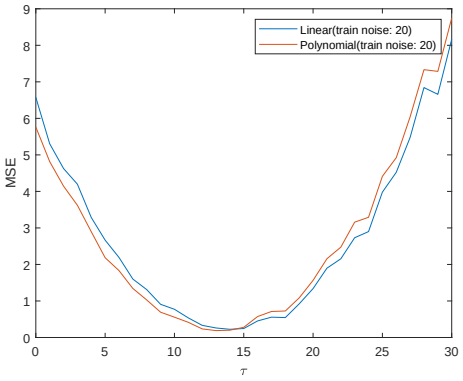


Figure 14: SVMs with various kernels and train noise $\tau_{train} = 20$ (see [Sta21], p. 38)

It sounds reasonable that an algorithm trained on data with train noise σ_0 behaves well for test data with the same noise. For $\sigma \nearrow 0$ the algorithm should also behave better because test data are less noisy. Apparently this reasoning is not correct: The MSE increases towards 0. In the extreme case $\sigma_0 = 0$ we expect an increasing behavior of $\sigma \mapsto \mathrm{MSE}(\sigma)$.

One purpose of this chapter is to explain the U-shape phenomenon which did not seem plausible to us, and to locate the minimum of the MSE curve. It turns out that

[4]This chapter expands the results and presents proofs that are referenced in [BSW24].

the qualitative behavior is the consequence of clipping pixel values to a finite interval like $[0, 1]$ or $[0, 255]$ in combination to an unbalanced number of pixel near the edge of the pixel range. This significantly destroys the Gaussian noise structure.

Clipping the noise in images with unbalanced number of pixel near the edge of the pixel range is the reason for the U-shape.

This emerging U-shape will be due to the bias-variance trade off (see e.g. [LS11]). Papers like [MM19] and [Bel+19] have already considered this effect in more detail depending on the network parameters. In contrast, we will slightly modify the test images and thus also obtain and analyze an U-shape via the bias-variance trade off.

The problem is relevant e.g. for images in astronomy, tomography, spectroscopy [Don+92] and industrial image processing (back lighting [DSW99]). In industrial image processing, a back light setup is often used for measurements. Here, the component under consideration is between the illumination and the camera. This creates a very high contrast, which is ideal for measurement tasks. However, the resulting grey values are often close to the edge of the grey value scale, which can cause the problem described above.

3.2 U-shape with Uniform Distribution

If we choose an uniformly distributed noise, we can go deeper in the analysis of the U-shape. $X_i \in \mathbb{R}$ describes the Input and we define our noisy input and the clipping function on the edges of the gray scale i.e. $[0, 1]$. Therefore the clipped noisy input is:

$$\tilde{X}_i = \Pi_{[a,b]}(X_i + \epsilon) \qquad \text{(Clipped Noisy Input)}$$

with $\epsilon \in \mathbb{R}, \epsilon \sim \mathcal{U}_{[-\frac{\sigma_0}{2}, \frac{\sigma_0}{2}]}$ and

$$\Pi_{[a,b]}(x) = \begin{cases} a, & x < a \\ x, & a \leq x \leq b \\ b, & x > b \end{cases}$$

In this application, we want to determine the undisturbed input. Therefore our label is X_i. Thus, we can reformulate the error sum in case of a neural network with a single

neuron and the identity function as activation.

$$
\begin{aligned}
&\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}(w\tilde{X}_i + \vartheta - X_i)^2] \\
=&\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}(w\tilde{X}_i)^2 + 2w\tilde{X}_i(\vartheta - X_i) + (\vartheta - X_i)^2] \\
=&\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}(w\tilde{X}_i)^2] + \mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}2w\tilde{X}_i(\vartheta - X_i)] + \mathsf{E}_X[\sum_{i=1}^{N}(\vartheta - X_i)^2] \\
=&w^2\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\tilde{X}_i^2] + 2w(\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\tilde{X}_i\vartheta] - \mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\tilde{X}_iX_i]) + \vartheta^2 \\
&- \mathsf{E}_X[\sum_{i=1}^{N}2\vartheta X_i] + \mathsf{E}_X[\sum_{i=1}^{N}X_i^2] \\
=&w^2\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\tilde{X}_i^2] + 2w\vartheta\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\tilde{X}_i] - 2w\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\tilde{X}_iX_i] + \vartheta^2 \\
&- 2\vartheta\,\mathsf{E}_X[\sum_{i=1}^{N}X_i] + \mathsf{E}_X[\sum_{i=1}^{N}X_i^2] \\
=&\vartheta^2 - 2\vartheta\bar{X} + w^2\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\Pi_{[a,b]}(X_i + \epsilon)^2] \\
&+ 2w\vartheta\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\Pi_{[a,b]}(X_i + \epsilon)] \\
&- 2w\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\Pi_{[a,b]}(X_i + \epsilon)X_i] + \mathsf{E}_X[\sum_{i=1}^{N}X_i^2]
\end{aligned}
$$

Now we assume an imbalance towards the black pixels with $X \in [0, 0.2]$ uniformly distributed.

$$
\begin{aligned}
=&\vartheta^2 - 2\vartheta\bar{X} + w^2\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\Pi_{[0,1]}(X_i + \epsilon)^2] \\
&+ 2w\vartheta\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\Pi_{[0,1]}(X_i + \epsilon)] \\
&- 2w\,\mathsf{E}_{X,\varepsilon}[\sum_{i=1}^{N}\Pi_{[0,1]}(X_i + \epsilon)X_i] + \mathsf{E}_X[\sum_{i=1}^{N}X_i^2]
\end{aligned}
$$

In addition, we fix the train noise to e.g. $\sigma_0 = 0.1$, to achieve an interpretable term.

$$\approx \vartheta^2 - 0.2\vartheta + 0.0133 + 0.0141w^2 + 0.202w\vartheta - 0.0266w$$

Thus, for a network without bias ($\vartheta = 0$) we get $w_{min} = 0.9433$. We choose $\vartheta = 0$ because the minimum is also close to zero and thus we can better explain the values. In doing so, we can see that the estimator tends to underestimate our system, since for a noise with an expected value of zero, $w_{min} = 1$ should apply. This shift of w_{min}, provides a worse prediction in case of noised data with $\sigma < \sigma_0$. Which explains the increase in the MSE for $\sigma \nearrow 0$.

With an imbalance to white pixels, we see a similar shift. An analytical calculation of the expected values and the extraction of σ_0 was unfortunately not possible. Therefore, in the next section we will switch to a high level formulation.

3.3 Analysis with High Level Formulation

In this section we analyze the U-shape using typical expressions from statistics. For this we will use an equation from [HTF09], which we will derive briefly and then convert to our problem. Let $X_1, \ldots, X_n$ be iid realizations of a real valued random variable $X_\varphi \in \mathbb{R}^n$ with parameter or label values $\varphi_1, \ldots, \varphi_n$, and $\hat{f} = \hat{f}(\cdot; X_1, \ldots, X_n))$ an estimator for the label φ of X_φ. So we can plug in X as an argument for $\hat{f} = \hat{f}(X)$ and get a real valued random variable as well, we assume that the mean value $\mathsf{E}[\hat{f}(X)]$ exists.

This is the typical setting in machine learning applications: An estimator for a quantity depending on the observations, here with the interpretation of a parameter φ that can be seen as a generator of the random variable $X = X(\varphi)$, should be constructed. In machine learning terms, a neural network or similar device shall be learned, with training data $(X_1, \varphi_1), \ldots, (X_n, \varphi_n)$. This results in an learning device which can be used for (nonlinear) regression $X \mapsto \varphi$. There are similarities and differences between machine learning and linear regression or nonlinear regression: In both cases a parameter is computed based on data. However regression computes a single parameter $\hat{\varphi}$ based on a multitude of data (x_i, y_i) for the same assumed parameter $\varphi_\star$. Machine learning constructs a function $\hat{f}$ between x and y, using a multitude of data (x_i, y_i) for different parameters φ_i, that can be used to compute parameters φ for single observations x.

Think of $X \sim \mathcal{U}_{[-\frac{\sigma}{2}, \frac{\sigma}{2}]}$ and $\hat{f}(X) = \frac{1}{n} \sum_{i=1}^{n} X_i$. Then we can calculate the error corre-

sponding to the random variable φ:

$$\begin{aligned}
\mathsf{E}_\varphi[(\hat{f}(X)-\varphi)^2] &= \mathsf{E}_\varphi[(\hat{f}(X)-\mathsf{E}_\varphi[\hat{f}(X)]+\mathsf{E}_\varphi[\hat{f}(X)]-\varphi)^2] \\
&= \mathsf{E}_\varphi[(\hat{f}(X)-\mathsf{E}_\varphi[\hat{f}(X)])^2] \\
&\quad + 2\,\mathsf{E}_\varphi[(\hat{f}(X)-\mathsf{E}_\varphi[\hat{f}(X)])\cdot(\mathsf{E}_\varphi[\hat{f}(X)]-\varphi)] \\
&\quad + \mathsf{E}_\varphi[(\mathsf{E}_\varphi[\hat{f}(X)]-\varphi)^2]
\end{aligned}$$

In the middle term $(\mathsf{E}_\varphi[\hat{f}(X)]-\varphi)$ is a scalar which can be moved outside the expectation. Then the remaining factor gives $\mathsf{E}_\varphi[\hat{f}(X)-\mathsf{E}_\varphi[\hat{f}(X)]]=0$, so the middle term disappears an we get

$$\begin{aligned}
\mathsf{E}_\varphi[(\hat{f}(X)-\varphi)^2] &= \mathsf{E}_\varphi[(\hat{f}(X)-\mathsf{E}_\varphi[\hat{f}(X)])^2]+\mathsf{E}_\varphi[(\mathsf{E}_\varphi[\hat{f}(X)]-\varphi)^2] \\
&= \mathsf{E}_\varphi[(\hat{f}(X)-\mathsf{E}_\varphi[\hat{f}(X)])^2]+\mathsf{E}_\varphi[\mathsf{E}_\varphi[\hat{f}(X)]-\varphi]^2 \\
&= \mathsf{E}_\varphi[(\hat{f}(X)-\mathsf{E}_\varphi[\hat{f}(X)])^2]+(\mathsf{E}_\varphi[\hat{f}(X)]-\varphi)^2 \\
&= \mathsf{Var}_\varphi[\hat{f}(X)]+\mathsf{Bias}_\varphi[\hat{f}(X)]^2
\end{aligned}$$

Let us now consider the U-shape application. Therefore we define the disturbed and clipped inputs $\tilde{X} := \Pi_{[0,1]}(X+\varepsilon) \in \mathbb{R}^n$ with $\varepsilon \in \mathbb{R}^n, \varepsilon \sim N(0,\sigma^2)$. Remark, that also our estimator $\hat{f}$ is trained by disturbed and clipped inputs $\tilde{X}$ with a fixed noise σ_0. This results in the error function, where the expectation is first computed over the distribution of ε and afterwards φ:

$$\begin{aligned}
\mathsf{Err}[X] &= \mathsf{E}_{\varphi,\varepsilon}[(\hat{f}(\tilde{X})-f(X))^2] \\
&= \mathsf{E}_{\varphi,\varepsilon}[(\hat{f}(\tilde{X})-\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]+\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]-f(X))^2] \\
&= \mathsf{E}_{\varphi,\varepsilon}[(\hat{f}(\tilde{X})-\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})])^2] \\
&\quad + 2\,\mathsf{E}_{\varphi,\varepsilon}[(\hat{f}(\tilde{X})-\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})])(\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]-f(X))] \\
&\quad + \mathsf{E}_\varphi[(\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]-f(X))^2] \\
&= \mathsf{Var}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]+2\,\mathsf{E}_{\varphi,\varepsilon}[(\hat{f}(\tilde{X})-\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})])(\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]-f(X))] \\
&\quad + \mathsf{E}_\varphi[\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]-f(X)]^2+\mathsf{Var}_\varphi[\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]-f(X)] \\
&= \mathsf{Var}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]+2\,\mathsf{E}_{\varphi,\varepsilon}[(\hat{f}(\tilde{X})-\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})])(\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]-f(X))] \\
&\quad + \mathsf{Bias}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]^2+\mathsf{Var}_\varphi[f(X)]
\end{aligned}$$

X is not a random variable here, so $\mathsf{Var}_\varepsilon[f(X)]$ is constant. Thus the essential term for

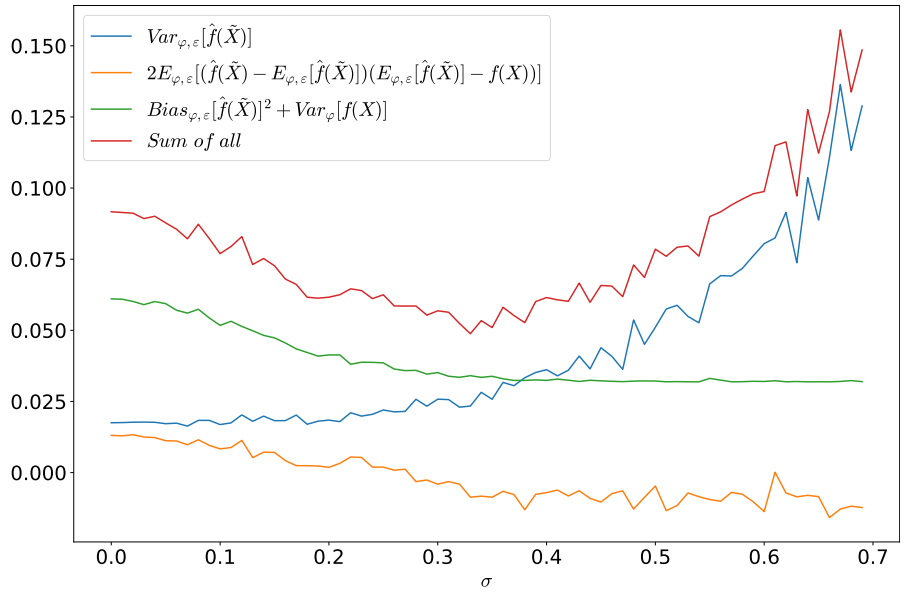


Figure 15: Overview of the different terms of equation (3.1)

the U-shape is:

$$\begin{aligned}\mathsf{Err}[X] = \mathsf{Var}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})] + \mathsf{Bias}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]^2 + \mathsf{Var}_{\varphi}[f(X)] \\ + 2\,\mathsf{E}_{\varphi,\varepsilon}[(\hat{f}(\tilde{X}) - \mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})])(\mathsf{E}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})] - f(X))]\end{aligned} \tag{3.1}$$

$\mathsf{Var}_{\varphi}[\hat{f}(\tilde{X})]$ is monotonically increasing with increasing test noise σ in our experiments. This curve is also fitting to the consideration that the error increases with increasing noise σ. The U-shape is created by the other two terms. Especially in the case of $\mathsf{Bias}_{\varphi,\varepsilon}[\hat{f}(\tilde{X})]^2$, a clear U-shape with a minimum at σ_0 can be seen (see Figure 15 for an example of the different terms). This bias, which is contained in both terms once in the square and once without, creates the U-shape. For $\sigma < \sigma_0$, the estimator tends to underestimate the system. For $\sigma > \sigma_0$, it overestimates the system. In both cases, the error increases the further away we are from the trained σ_0. Therefore, the minimum occurs at approximately σ_0. However, $\mathsf{Var}_{\varphi}[\hat{f}(\tilde{X})]$ shifts the minimum in the error sum.

For better visualization, the individual terms were plotted in Figure 15. These graphs emerge from an experiment with ten input neurons, a hidden layer with 124 neurons and the sum of the ten input neurons as the label. In addition, the input neurons were uniformly distributed on $[0, 0.2]$ and we train with $\sigma_0 = 0.5$.

If we use uniformly distributed pixel on the whole pixel range $[0, 1]$ the bias will not affect the sum and alternate around zero. Therefore you can see Figure 16, which describes several experiments with different amount of test noise σ and different distributed pixel.

Pixel range in this context always means the upper limit for the uniformly distributed pixels between zero and the upper limit.

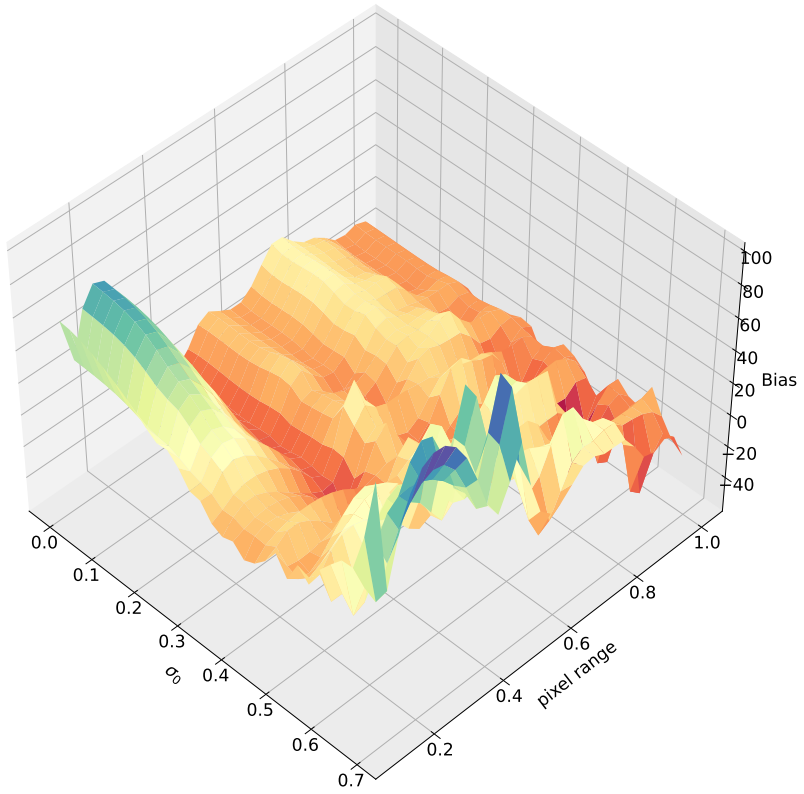


Figure 16: Overview of the bias dependent on σ and pixel range

Also, if we look more closely at the bias, we see a U-shape. However, this is not recognizable due to the scaling in Figure 15 and also plays a minor role in the error sum (3.1).

We suspect that the U-shape is a special case of the bias-variance trade off (e.g. described in neural networks applications in [LS11] or in general for statistical methods in [Jam+17]). This describes that the variance can be reduced by increasing the bias. High bias can lead to important relations between inputs and outputs not being established. Thus, the network would be more susceptible to fluctuations in the data, as important connections may not have been trained. The modified σ in combination with the clipping, is probably too complex for our network trained in this way and thus lets the bias in the error calculation increase.

In summary, we can say that the clipped and unbalanced pixels cause a bias-variance trade off, which leads to a U-shape in the error function. The analysis for the U-shape in the real case (e.g. in the application with the slitdisc) remains open.

Salt and Pepper Noise

Similar considerations can be made with SnP Noise. Here, however, the first question is whether a U-shape occurs and what could be the reason for this. Therefore, we first define the SnP noise

$$Z(x_{i,j}, r) = \begin{cases} a, & \text{with probabilty } \frac{1-r}{2} \\ b, & \text{with probabilty } \frac{1-r}{2} \\ x_{i,j}, & \text{with probabilty } r \end{cases}$$

with the grey value of the pixel $x_{i,j}$ and the noise probability $r \in [0, 1]$. Again we want to minimize

$$\arg\min_{w \in \mathbb{R}} \frac{1}{N} \sum_{i=1}^{N} (w - Z_i)^2$$

for a set of realizations Z_i and a single neuron with the weight $w \in \mathbb{R}$. Therefore we can write

$$\begin{aligned} \mathsf{E}[\sum_{i=1}^{N} (w - Z_i)^2] &= (w-a)^2 P\left(\frac{1-r}{2}\right) + (w-b)^2 P\left(\frac{1-r}{2}\right) + P(r)(w - x_{i,j})^2 \\ &= \frac{1-r}{2}\left(w^2 - 2wa + a^2 + w^2 - 2wb + b^2\right) + r(w-x)^2 \\ &= w^2 - 2xrw - (a+b)(1-r)w + rx^2 + (1-r)\left(\frac{a^2+b^2}{2}\right) \end{aligned}$$

To minimize the expected value we search for $w_{\min}$. Therefore we derive to w.

$$\frac{\partial}{\partial w} \mathsf{E}[\sum_{i=1}^{N} (w - Z_i)^2] = 2w - 2xr - (1-r)(a+b) \stackrel{!}{=} 0$$

Thus follows

$$w_{\min} = xr + (1-r)\frac{a+b}{2}$$

Then we derive the expected value with $w_{\min}$ set to r and set it equal to zero.

$$r\left(x^2(1-2r_0) + 2xr_0 - x - \frac{1}{2}r_0\right) + w_{\min}^2 - w_{\min}(a+b) + \frac{a^2+b^2}{2} = 0$$

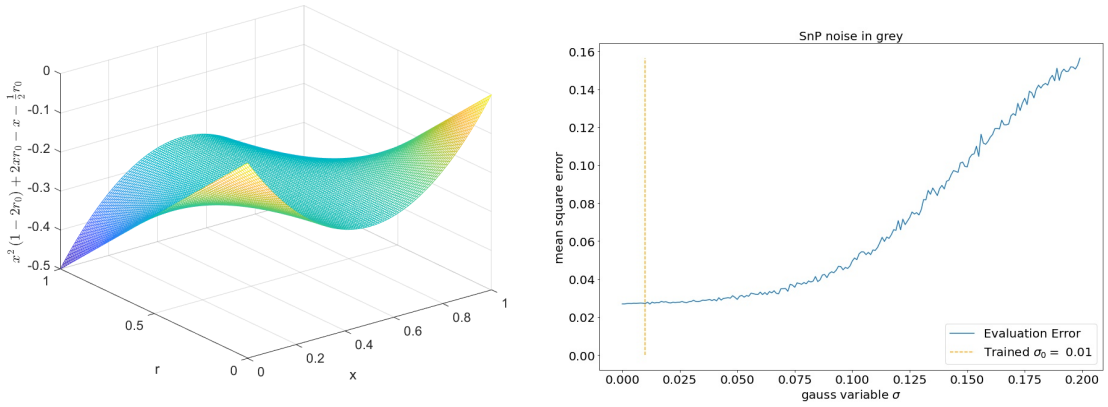


Figure 17: Multiplier of r (left) Experiment with ResNet and SnP noise(right)

Because $\left(x^2(1-2r_0)+2xr_0-x-\frac{1}{2}r_0\right)$ is always negative (see Figure 17 on the left), $r_{\min}=1$ is valid for all r_0. This means that no matter what disturbance r the network was trained with, the network performs best without disturbance ($r=0$). Thus, there should not arise a U-shape, which was also the case in the experiments (see Figure 17 on the right). This result also fits with the previous consideration of the Gaussian noise. Since we do not get any clipping in the SnP noise, there is also no U-shape.

3.4 U-shape Experiments with Different Edge Behavior

To observe the U-shape more detailed, we trained some network architectures with the same hyper parameters but different behavior on the edges of the grey value scale. We identify four different methods *clipping*, *reflect*, *scale* and *without edges*. In the following we explain these four methods and in addition they are visualized in Figure 18. However, this is not based on the slit disc images but on two artificially created Gaussian bells (drawn blue in each figure). The histograms of the slit disc images were rather unsuitable for visualization due to two spikes from the white and dark area.

clipping: Cuts all grey values above 255 and under 0 and set them to 255 or 0 correspondingly. In the upper left corner of Figure 18 we see an example pixel at 230 and depending on whether the pixel and the disturbance stays smaller than 255 or gets larger, it keeps its value or is set to 255. This sets the grey area of the disturbance completely to the maximum 255, which explains the enormous spike. Analogous is the behavior near zero. This method is probably the most realistic one, because with a fixed aperture (Typically, lenses with a fixed aperture and focal length are used in the industry) and an outlier the camera also uses the minimum and maximum of the grey scale, respectively. Some cameras consider these outliers as errors and use the average

of the surrounding pixels instead. However, we will neglect such technical smoothings in this work.

reflect: Mirrors all values over 255 and under 0 by 255 or 0. Thus, we do not neglect any part of the Gaussian bell, but merely mirror it. In the upper right corner in Figure 18, we see again an example pixel at 230. Here the grey part of the disturbance is mirrored by 255 so that the right part of the red Gaussian bell is shorter but taller than the left part.

scale: Rescales the image. Thus the maximum pixel is set to 255 and the minimum to 0. Intermediate values are scaled accordingly. This method completely maps the Gaussian bell. In the lower left corner in Figure 18, we visualize this behavior with a Gaussian noise ($\sigma = 0.01$)[5]. The disturbed grey values are then scaled so that the maximum 353 corresponds to 255 and the minimum -87.72 to 0. The grey value scale is thus compressed (red histogram). With this variant, the disturbance is recorded evenly and undistorted in the image. However, the contrast decreases due to the compression.

without edges: Here we reduce the range of the grey scale by e.g. 15% on each side (the new grey values are in $[38, 217]$). Now it is very unlikely that the ends 0 and 255 are reached by Gaussian noise. But if it is reached, *clipping* is applied. However, this variant requires a high contrast in the image. In the lower right corner of Figure 18 we can see in red the histogram of the compressed image. This gives us a margin of 38 pixels to each side.

In the following experiments, we first focused on grey-scale images. We train a neural network with a ResNet architecture and identical hyper parameters[7]. In addition, the training images are perturbed with a Gaussian noise ($\sigma_0 = 0.01$). Only the behavior at the edge differs in the individual experiments as described above. We then tested the resulting neural network with a random test set and the corresponding sort of noise. We repeat this process with different σ and calculate the MSE in each case. These different errors depending on σ, are shown in Figure 19.

In the upper left corner of Figure 19, we see the *'clipping'* variant. Here we identify again a U-shape and the trained $\sigma_0 = 0.01$ is far from the minimum $\sigma_{min} = 0.023$. Remark this is strongly counter intuitive, since thus the error in the neural network is larger for smaller disturbances or for the trained disturbance. The graph for variant *'without edges'* is similar to that of variant *'clipping'*. This probably results from the

[5]It should be noted that we apply the disturbance to the normalized image in the interval $[0, 1]$.

[7]The hyper parameters are: Epochs=200, Learning rate=0.001, Loss= MSE, Batch size=20, Optimizer: Adam with standard parameters

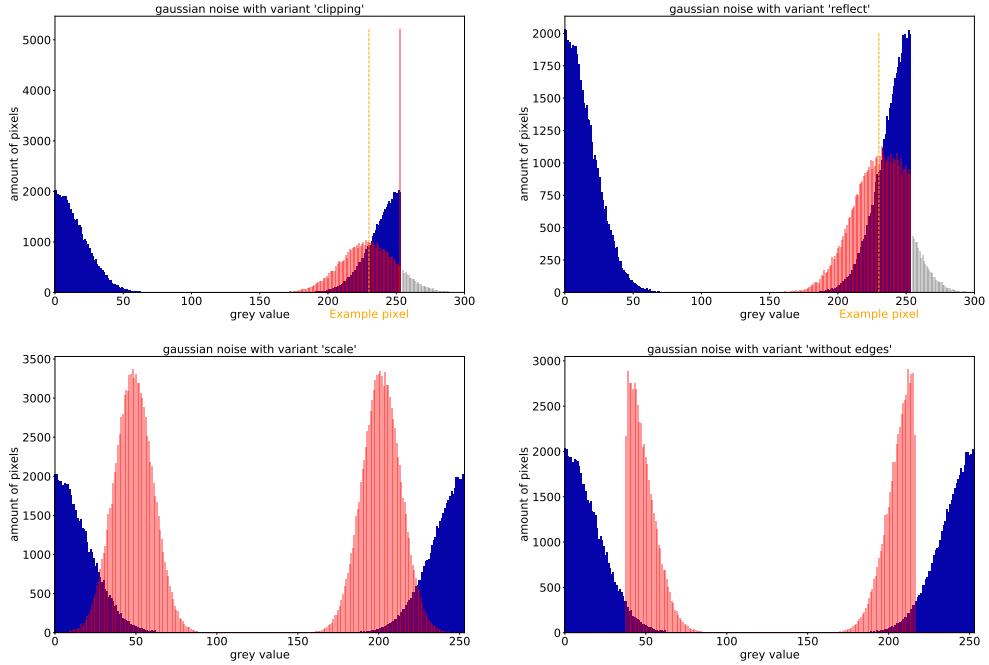


Figure 18: Illustration of the different Gaussian noise variants[6]

fact that variant *'clipping'* is used if the edge of the grey scale is reached. If the noise σ is large enough, this is unavoidable. This also explains the larger σ_0. In variant *'scale'*, minimum σ_{min} and train noise σ_0 are closer to each other. However, this is only the case because the U-shape is more compressed than in variant *'clipping'*. This compression is probably due to the fact that the neural network cannot deal with the new scaling of the pixels. This is why the error increases faster than in comparison with the other variants. The only variant without a directly recognizable U-shape is variant *'reflect'*. We assume that the reason for this is the more shifted expected value. As visualized in Figure 18, the shape of the Gaussian curve is almost completely preserved, but the expected value shifts more significantly. We assume that due to the strongly shifted expected value the network tolerates grey value fluctuations more strongly and thus concentrates better on the problem case of rotation determination. Experiments in which the exact grey value is important (i.e. experiments like in Figure 15) are therefore completed more poorly with this variant. The increase at very high σ, as with any variant, is due to the blurring transition between white and grey pixel. In addition, minimum σ_{min} and train noise σ_0 are closest together here. However, the networks hardly differ in accuracy. The training errors are close and even in an experiment with mixed sigmas and noise variants there were hardly any differences. Therefore, the results were written in the appendix for the

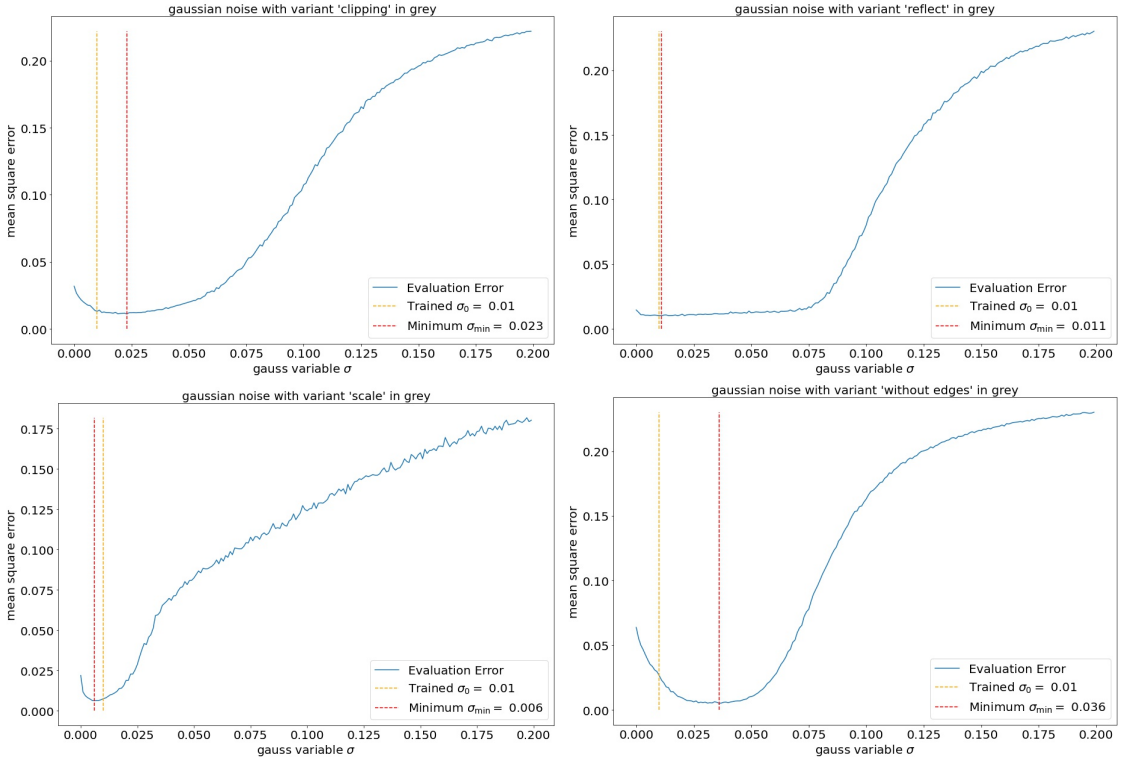


Figure 19: Experiment with a ResNet architecture and a trained $\sigma_0 = 0.01$

sake of completeness (see Appendix C.4).

In the experiments above we always use the ADAM Optimizer to train the neural networks. To better understand its influence on both accuracy and U-shape, we now look a bit closer at its step size. Therefore we need to define the optimal step size in a gradient descent algorithm.

Definition 3.1. ***(Optimal step size)***
Let $f : \mathbb{R}^n \to \mathbb{R}, x_t \in \mathbb{R}^n$ and $g_t \in \mathbb{R}^n$ be a descent direction for f in x_t with the time steps $t \in \mathbb{N}$. Then we define the optimal step size α_{opt} with

$$f(x_t + \alpha_{opt} g_t) \leq f(x_t + \alpha g_t) \quad , \forall \alpha \geq 0.$$

Logically, this optimal step size is unknown and optimizers try to approximate it as best they can. Anyway with the optimal step size we can estimate the distance to the minima recursive.

Theorem 3.2. *(Convergence with optimal step size)*
Let $f : \mathbb{R}^n \to \mathbb{R}$ *with* $f(x) = \frac{1}{2}x^\perp Qx + b^\perp x + a$, *positive definite matrix* $Q \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n, a \in \mathbb{R}$ *and an unique global minima* $x_\star = -Q^{-1}b$. *Also let* $\{x_t\}_{t\in\mathbb{N}}$ *be a row with optimal step size* $\alpha_{opt,t} > 0$ *in the point* x_t *in the descent direction* $g_t := -\nabla f(x_t)$ *and* $\nabla f(x_t) \neq 0, \forall t \in \mathbb{N}$ *with respect to*

$$x_{t+1} := x_t + \alpha_{opt,t} g_t.$$

Then applies

$$f(x_{t+1}) - f(x_\star) \leq \left(\frac{\frac{\lambda_{max}(Q)}{\lambda_{min}(Q)} - 1}{\frac{\lambda_{max}(Q)}{\lambda_{min}(Q)} + 1} \right)^2 (f(x_t) - f(x_\star)) \,. \tag{3.2}$$

Proof. For a proof see i.e. [RHG13] p. 103 □

For reasons of clarity we abbreviate $\kappa := \kappa(Q) = \frac{\lambda_{max}(Q)}{\lambda_{min}(Q)}$ and call it the *spectral condition number*. To proof empirically, weather the ADAM converges against the optimal step size, we make some experiments.

Experiment 3.3. (Optimal step size by a quadratic function):[8]
We use the quadratic function

$$f : \mathbb{R}^2 \to \mathbb{R} \text{ with } f(x) = \frac{1}{4}(x_1 - 5)^2 + (x_2 - 6)^2 \tag{3.3}$$

with the global minima $f(x_\star) = 0$ at $x_\star = (5, 6)^\perp$. We start at $x_0 = (9, 7)^\perp$ with an abort tolerance $||\nabla f(x^k)||_2 \leq 10^{-6}$. Our example function $f(x_t)$ corresponds to $Q = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 2 \end{pmatrix}$ and therefore the eigenvalues

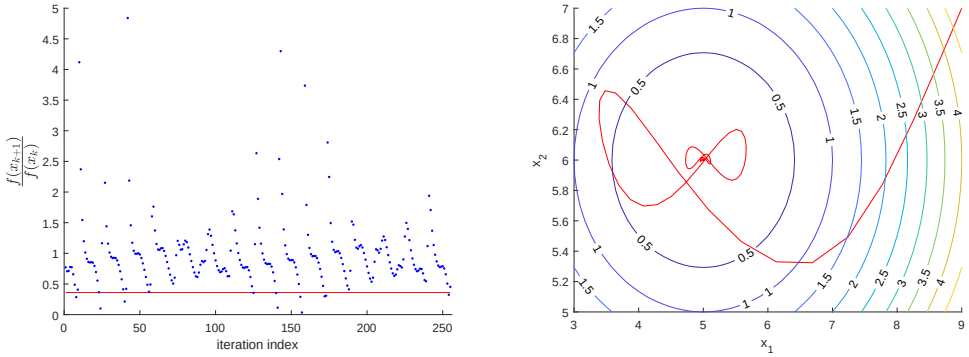
$$\lambda_{max} = 2 \text{ and } \lambda_{min} = \frac{1}{2}$$

Accordingly, $\kappa = 4$ follows. With inequality (3.2) we can estimate the convergence factor C_k for an optimal step size.

$$\frac{f(x_{t+1}) - f(x_\star)}{f(x_t) - f(x_\star)} := C_t \leq \left(\frac{\kappa - 1}{\kappa + 1} \right)^2 = 0.36 \tag{3.4}$$

Optimizers with an optimal step size, have a convergence factor $C_k \leq 0.36$ for this

[8] Reinhardt in [RHG13] p. 130 did the same experiment but without ADAM.



(a) Blue dots convergence factor of the ADAM; red line optimal step size

(b) Behavior of x_1 and x_2 in contrast to the contour of function (3.3)

Figure 20: Experiment 3.3 with quadratic function (3.3) and ADAM

example. Remark the convergence factor depends on the starting point and can be lower than 0.36. Reinhardt did the same experiment with the gradient descent algorithm and an optimal step size (see [RHG13] p. 130). There we can see that the upper barrier is reached for $x_0 = (9, 7)^{\perp}$ but never topped. However, calculating the optimal step size is very computationally intensive. Therefore in practice we use optimizers, which converge at least against the optimal step size but are not that computationally intensive (for more experiments with other optimizers see [RHG13]).

At this point, the convergence factor will be illustrated in more detail. The larger the factor is, the longer the algorithm needs to reach the minimum. Whenever the factor is greater than one, it even moves away from the solution.

We focus on the ADAM optimizer used in the experiments in Section 2.4. Therefore we look at the convergence factor of the ADAM in the experimental setting above. Due to the fact that the ADAM is widely and frequently used, it is to be expected that the convergence factor at least converges towards the upper bound of inequality (3.4). However, the tests show that the ADAM requires many more iterations and no convergence behavior is evident. We use the hyper parameters suggested by Kingma and Ba [KB15] ($\alpha = 0.1, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$).

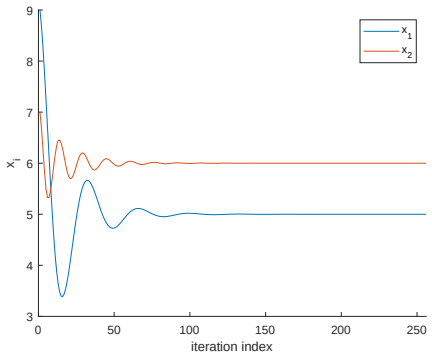


Figure 21: Behavior of x_1 and x_2

With these hyper parameters and the experimental setting above, we need 256 iterations (i.e. [RHG13] with the gradient descent and optimal step size only needs 30 iterations with the same abort tolerance and start point) to reach the abort tolerance. Moreover, the convergence factor does not seem to converge against the optimal step size (see Figure 20a). On the contrary, it seems that the convergence factor oscillates around one, sometimes stronger, sometimes weaker (mean value of the convergence factor is 1.0601). This oscillation also becomes clear when we look at the curves of x_1 and x_2 (see Figure 21).

We see an oscillating behavior of each parameter around the minimum coordinates $(5, 6)^{\perp}$. This oscillating behavior is due to the momentum term in ADAM. To better understand this behavior, we study different optimizers (and especially on the ADAM) and their convergence behavior in the rest of this work.

4 ADAM Optimizer

One of the key problem of machine learning is minimizing a function f.

$$\min_x f(x) \tag{4.1}$$

If $\nabla f(x)$ can be computed, we can use Gradient Descent (GD) to solve problem (4.1). GD is one of the oldest methods for minimizing differentiable functions and works for a given function $f(x)$ as described in Algorithm 1. GD is a first-order iterative optimization

Algorithm 1 Gradient Descent GD

1: **while** x not converged **do**
2: $\quad x_{t+1} = x_t - \alpha \nabla f(x_t)$
3: $\quad t = t + 1$
4: **end while**

algorithm for finding a local minimum of differentiable functions. The general idea of GD is to minimize a function f by following a descent direction with the step size α. Every algorithm which is familiar with GD uses the negative gradient or variations thereof as descent direction. All the following algorithms are based on the GD, but since a lot of research has already been done on it, it will not be discussed further here.

In some settings it is possible to divide the function f into several subsequences l_i.

$$\min_x f(x) = \min_x \frac{1}{n} \sum_{i=1}^{n} l_i(x) \tag{4.2}$$

Especially in the area of machine learning, problem (4.2) is quite useful. Instead of minimizing the whole training set, we minimize a randomly chosen subset l_i of the whole training set. l_i is an approximation of $f(x)$ and helps to minimize $f(x)$ after only $|l_i|$ train data. The subset l_i is called minibatch. Optimizing this special setting with GD is called SGD (first published in [RM51]). SGD is very effective on large training

Algorithm 2 Stochastic Gradient Descent SGD

1: **while** x not converged **do**
2: $\quad x_{t+1} = x_t - \alpha \sum_{i=1}^{n} \frac{\nabla f(x_t)}{n}$
3: $\quad t = t + 1$
4: **end while**

sets, where each epoch is very time-consuming. Because x is updated only every n steps, this procedure can be easily parallelized. Based on the SGD Rumelhart et. al.

introduce SGD with momentum in [RHW86]. This algorithm remembers the update $\nabla f(x_t)$ at each iteration t and calculates the following update as a linear combination of the current gradient and the previous updates. This type of algorithm became a

Algorithm 3 Momentum algorithm

1: **while** x not converged **do**
2: $m_{t+1} = \beta m_t - \alpha \nabla f(x_t)$
3: $x_{t+1} = x_t + m_{t+1}$
4: $t = t + 1$
5: **end while**

class called momentum algorithm. If we replace $\nabla f(x_t)$ with $\sum_{i=1}^{n} \frac{\nabla f(x_t)}{n}$ we get the SGD version of the momentum algorithm with the batchsize n. For the sake of clarity, it is not included here and in the following algorithms. In practical use, however, the mini-batch component should be added to the algorithm.

These three properties represent a kind of toolbox for further algorithms. All these can be represented by a combination of these properties and the GD.

4.1 The Beginnings of GD Methods in Machine Learning

A major drawback of the GD is the direct influence of the size of the gradient. If we consider for example a very steep part of a function, we calculate a big gradient and therefore our step will be quite big. With so big steps we probably jump over a minimum. The opposite happens in a flat area and optimization can take extremely long. To avoid such scenarios Duchi et. al. invented in [DHS11] the AdaGrad (see Algorithm 4). Real

Algorithm 4 AdaGrad

1: **while** w not converged **do**
2: $v_{t+1} = v_t + \nabla_w f(w_t)^2$
3: $w_{t+1} = w_t - \alpha \frac{\nabla_w f(w_t)}{\sqrt{v_{t+1}+\epsilon}}$
4: **end while**

new in this algorithm is the first line (in Algorithm 4 line 2), where we calculate the squares of the gradient point wise. This means we lose the information about the sign of the gradient. Theses squares will be summed up over time and so we involve a kind of momentum to this algorithm. The second line is quite the same as in the original GD with the main difference by dividing with the square root of v and a new hyper parameter ϵ. So we can see that we are dividing the gradient with the square root of the momentum of the squared gradient. Without the momentum we would only separate the sign of the gradient. So the size of the gradient does not matter any more and we

only go steps in the size of the learning rate α. With the momentum and the fact that we only add positive values, we increase v every step where we are not in an extremum. Thus we are decreasing the step size with the time and thus refine the search. ϵ is only needed to avoid dividing by 0 and is typically in the range of $10^{-8} - 10^{-10}$.

All this improves the GD a lot but the momentum weights each gradient equally. So the first gradient will affect w_t on the same way as the current gradient. This fact seems a bit disproportionate, whereupon Hinton et. al. invented the algorithm RMSProp[9]. The RMSProp has only one difference to the AdaGrad namely another new hyper parameter β. We assume $0 < \beta < 1$ and it works as a decay rate for the momentum.

Algorithm 5 RMSProp

1: **while** w not converged **do**
2: $\quad v_{t+1} = \beta v + (1-\beta)\nabla_w f(w_t)^2$
3: $\quad w_{t+1} = w_t - \alpha \frac{\nabla_w f(w_t)}{\sqrt{v_{t+1}+\epsilon}}$
4: **end while**

Lets make a small example to illustrate the meaning of β. We assume $\beta = 0.9$ and so we only take 10% of $\nabla_w f(w_t)^2$ for our new v_t. This also means we take 90% of the old v_{t-1}. Now we look at the time $t+1$. We have again a new $\nabla_w f(w_{t+1})^2$ which only counts 10%. But the interesting thing is in v_t. We calculate v_t with 0.9 but in v_t we calculated already $\nabla_w f(w_t)^2$ with 0.1. So $\nabla_w f(w_t)^2$ only counts 9% to v_{t+1} and for each time step we go further it will be less. This mathematical trick ensures that the gradients are weighted as one would expect. Old gradients are not so important than new gradients for the current calculation.

Almost everywhere this optimizer performs much better than for example AdaGrad. Thus RMSProp became the most popular optimizer for many researchers until ADAM was invented. The ADAM optimizer is so important for this work that it has been given its own chapter.

4.2 Notation

Before introducing the ADAM, we give a brief recap of the notation used. The symbol $\perp$ denotes the transpose of a vector or matrix. We use the component-wise multiplication and division of vectors, as well as componentwise addition of vectors and scalar without any special notation. The reader should always know which calculation method is used due to the size of the arguments. For $f : \mathbb{R}^n \to \mathbb{R}$ the gradient and Hessian are written as ∇f and $\nabla^2 f$, provided they exist. Throughout this work we assume $f : \mathbb{R}^n \to \mathbb{R}^n$ at

[9] RMSProp is an unpublished optimizer, first mentioned by Geoffrey Hinton, Nitish Srivastava and Kevin Swersky in an online course [HSS12]

least continuously differentiable, twice continuously differentiable for some results. The open ball with radius r around $x \in \mathbb{R}^n$ is denoted by $B_r(x) = \{y \in \mathbb{R}^n : \|y - x\| < r\}$ and $\|x\|$ is any norm. We denote $\rho(A) = \max\{|\lambda| | \lambda \text{ eigenvalue of } A\}$ the spectral radius of a matrix A and $\text{diag}(v) \in \mathbb{R}^{n \times n}$ describes a matrix with components of $v \in \mathbb{R}^n$ on the diagonal.

4.3 Introduction to the ADAM Optimizer

The Adaptive Moment Estimation (ADAM) originally published by Kingma and Ba [KB15] is probably one of the most used training algorithm for neural networks. By combining the advantages of RMSProp [HSS12] and AdaGrad [DHS11], ADAM performed better than the previous optimizers. For example, Figure 22 shows the training costs for a multilayer neural network for the most commonly used optimizers. As one can see, ADAM performs best here. Implementations of the ADAM exist in all popular machine learning frameworks like Tensorflow or PyTorch. Even a couple of enhancements exist like Nadam [Doz16] or AMSGrad [RKK19] but each of them only performs better in special cases. To sum up, I would like to cite Ruder [Rud16] *¨Insofar, ADAM might be the best overall choice.¨*.

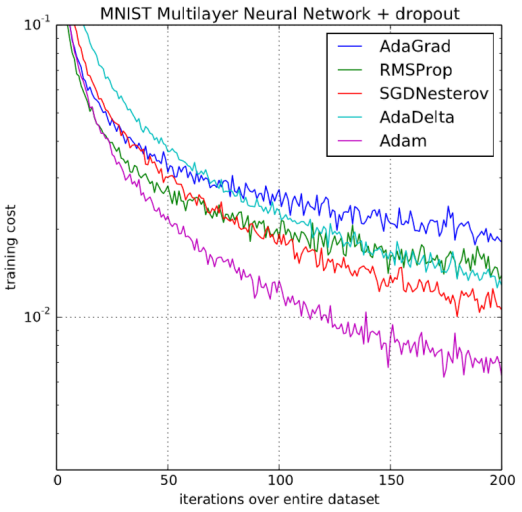


Figure 22: Comparison of optimizers. [KB15]

Unfortunately, there are several definitions of the ADAM optimizer in the literature. In the original publication from Kingma and Ba [KB15] they use $\sqrt{v} + \epsilon$ and the bias correction in $\hat{m}$ and $\hat{v}$. In [RKK19], [Che+19] and [Zou+19] they do not even use an ϵ. Only [Zou+19] initialize $v_0 = \epsilon$ to avoid dividing by zero at least in the first step of the algorithm. All three apply the bias correction in the learning rate α_t. In this publication we use the bias correction as mentioned in [KB15, Chapter 2] and $\sqrt{v + \epsilon^2}$ (see Algorithm 6). The differences between $\sqrt{v + \epsilon^2}$ and $\sqrt{v} + \epsilon$ are minimal (see i.e. Figure 23[10]) especially in the region around $v \approx 0$. The example in Figure 23 was intentionally kept simple (simple data set and classic CNN architecture), as only the behavior of the various ADAM methods is to be shown.

[10] The experiment is programmed with Keras 2.2.4, Tensorflow 1.13.1 and Python 3.6.8.

Algorithm 6 ADAM Optimizer [KB15]

Require: $\alpha \in \mathbb{R}^+, \epsilon \in \mathbb{R}, \beta_1, \beta_2 \in (0,1), w_0 \in \mathbb{R}^n$ and the function $f(w) \in C^2(\mathbb{R}^n, \mathbb{R})$

1: $m_0 = 0, v_0 = 0, t = 0$
2: **while** w not converged **do**
3: $\quad m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla_w f(w_t)$
4: $\quad v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla_w f(w_t)^2$
5: $\quad w_{t+1} = w_t - \alpha \frac{\sqrt{1-\beta_2^{t+1}}}{(1-\beta_1^{t+1})} \frac{m_{t+1}}{\sqrt{v_{t+1}+\epsilon^2}}$
6: $\quad t = t+1$
7: **end while**

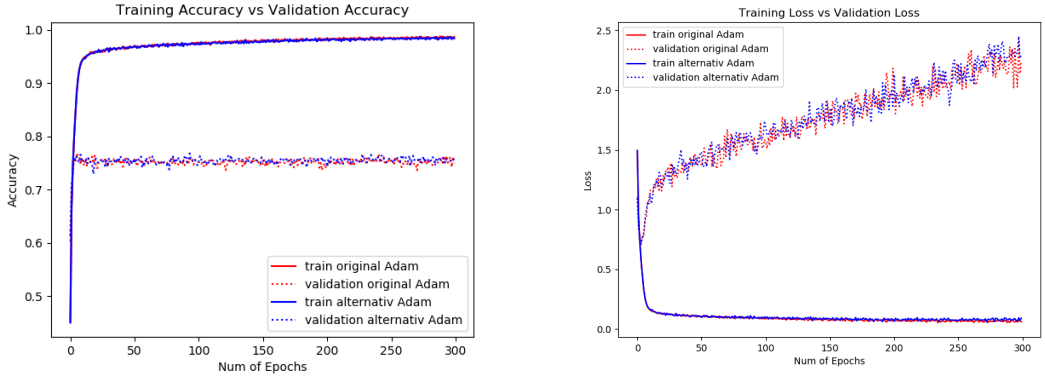


Figure 23: Training of the Fashion-MNIST dataset ([XRV17]) with two different ADAM methods.

The aim behind the use of ϵ, avoid dividing by zero, is fulfilled in both variants. With a fixed ϵ the used variant is even better than the original one. The basic idea of the ADAM is an usage without ϵ, but to avoid dividing by zero Kingma and Ba add a small disturbance ϵ. However, as can be seen in Figure 24, the deviation between $\sqrt{v+\epsilon^2}$ and $\sqrt{v}$ is much smaller than between $\sqrt{v}+\epsilon$ and $\sqrt{v}$ for a small value v.

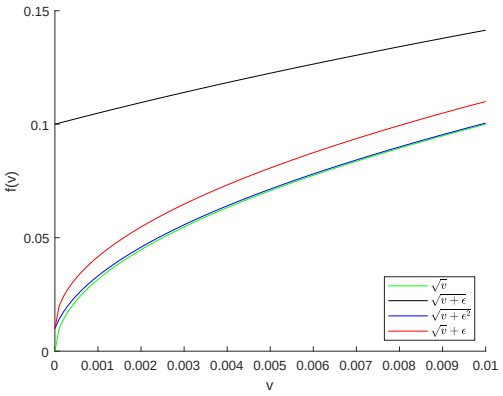


Figure 24: Comparison between ϵ possibilities.

In the following chapter we introduce

the hyperparameter bounding

$$\frac{\alpha}{\epsilon} \max_{i=1}^{n} (\mu_i) (1 - \beta_1) < 2\beta_1 + 2 \tag{4.3}$$

with μ_i the i-th eigenvalue of the Hessian $\nabla^2 f(w_\star)$ and all other parameters defined as in Algorithm 6. This inequality is derived in the proof of Theorem 4.5 and henceforth shows whether the ADAM converges.

4.3.1 Preliminaries

We recall some standard definitions and facts from the theory of difference equations and discrete time dynamical systems, see e.g. [ANT00, Definition 5.4.1] or [KP01, Chapter 4.2]. Consider $T : \mathbb{N}_0 \times M \to M$ with $M \subset \mathbb{R}^n$ which defines a non-autonomous dynamical system by the iteration

$$x_{t+1} = T(t, x_t), \quad t \in \mathbb{N}_0, x_0 \in M \tag{4.4}$$

with solutions $x : \mathbb{N}_0 \to M$, $t \mapsto x_t$ depending on the initial value x_0. We use the notations $x_t = x(t; x_0)$ and $x = x(\cdot; x_0)$ to emphasize the dependence of solutions on the initial value if necessary. We always use the initial time $t_0 = 0$.

Autonomous systems constitute the special case where T does not depend on t, so we can abbreviate to $\bar{T} : M \to M$ and write

$$x_{t+1} = \bar{T}(x_t), \quad t \in \mathbb{N}_0, x_0 \in M \tag{4.5}$$

A point $x_\star \in M$ is called *equilibrium* or *fixed point* if $T(t, x_\star) = x_\star$ for all $t \in \mathbb{N}_0$, so the constant function $x_t = x_\star$ for all $t \in \mathbb{N}_0$ is a solution of (4.4). In the following the asterisk will always denote equilibria or their components. Consider a solution $x = x(\cdot; x_0)$ of (4.4). x is called *stable*, if for each $\varepsilon > 0$ there exists $\delta = \delta(\varepsilon)$ such that any solution $\tilde{x} = \tilde{x}(\cdot; \tilde{x}_0)$ of (4.4) with $\|\tilde{x}_0 - x_0\| < \delta$ fulfills $\|\tilde{x}_t - x_t\| < \varepsilon$ for all $t \in \mathbb{N}_0$.

x is called *attractive* if there exists $\delta > 0$ such that any solution $\tilde{x}$ with $\|\tilde{x}_0 - x_0\| < \delta$ fulfills $\lim_{t\to\infty} \|\tilde{x}_t - x_t\| = 0$. x is called *asymptotically stable* if it is stable and attractive.

Recall that a *contraction* is a self-mapping on some set with Lipschitz constant $L < 1$, i.e. a mapping $\bar{T} : M \to M$, $M \subset \mathbb{R}^n$ with $\left\|\bar{T}(x) - \bar{T}(y)\right\| \leq L \|x - y\|$ for all $x, y \in M$. If M is complete, i.e. all Cauchy sequences converge, then a unique fixed point $x_\star \in M$ of $\bar{T}$ exists by the Banach fixed point theorem.

Theorem 4.1. ***Linearized asymptotic stability implies local nonlinear stability*** *Consider $\bar{T} : M \to M$ with a fixed point $x_\star$ and $\bar{T}$ continuously differentiable in an open $B_r(x_\star) \subset M$ of $x_\star$. Denote the Jacobian from the system $\bar{T}$ by $J_{\bar{T}}(x_\star)$, and assume $\|J_{\bar{T}}(x_\star)\| < 1$ for some norm on $\mathbb{R}^{n\times n}$. Then there exists $0 < \varepsilon \leq r$ and $0 \leq c < 1$ such that for all x_0 with $\|x_0 - x_\star\| < \varepsilon$*

$$\|x(t; x_0) - x_\star\| \leq c^t \|x_0 - x_\star\| \qquad \forall t \in \mathbb{N}_0.$$

i.e. $x_\star$ is locally exponentially and asymptotically stable.

The theorem is the core of the first method of Lyapunov for discrete time systems. For a proof see [Ela05, Corollary 4.35.].

In addition, we define the spectral radius $\rho(A)$ of $A \in \mathbb{R}^{n\times n}$ as the largest eigenvalue of A in terms of amount. The spectral radius is an important indicator of the stability of equilibrium points.

Theorem 4.2. *Let $x(t+1) = Tx(t)$ be a system with $T : M \to M$, $x_\star$ is an equilibrium point and T is continuous differentiable in an area around $x_\star$. Then follows:*

- *If $\rho(J_T(x_\star)) < 1$, than $x_\star$ is asymptotically stable*
- *If $|\lambda| > 1$ for all eigenvalues of $J_T(x_\star)$, so $x_\star$ is unstable.*

For a proof see [Kra15] or for the first point (the first point is the important case in this work) see [Ela05, Corollary 4.34].

4.4 Convergence Proof [11]

We consider the learning algorithm from the standpoint of dynamical systems and define a common state vector x consisting of the moments – like m and v for ADAM – and the weights, so we have $x = (m, v, w)$. Then the optimization can be written as an iteration $x_{t+1} = T(t, x_t)$ for some function $T : \mathbb{N}_0 \times X \to X$ with $X \subset \mathbb{R}^p$, which defines a non-autonomous dynamical system. The function f to be minimized in the learning process, or rather its gradient, becomes a part of T. If f is at least continuously differentiable a local minimum gives the necessary condition $\nabla f(w_\star) = 0$. We show that this condition leads to a fixed point $x_\star$ of T, where the moments are all zero. We analyze the stability properties of this fixed point and prove local asymptotic stability. This is done by considering a time-variant iteration T as the perturbation of a time-invariant iteration $\bar{T}$ where Banach-like fixed point arguments can be applied. We use the

[11] This chapter expands the results and presents proofs that are referenced in [BW19a].

second method of Lyapunov for stability analysis where the vanishing moments simplify the computation and estimates for the eigenvalues. Asymptotic stability however is equivalent to convergence of the iteration defined T to $x_\star$ for all x_0 sufficiently close to $x_\star$. The conditions needed for the fixed point analysis and stability results require the learning rate to be sufficiently small. Note that these results cannot be obtained directly from standard fixed point theorems for autonomous systems, because the iteration index t enters the dynamics. Therefore also estimates of the eigenvalues depend on the iteration t, and even a bound on the spectral radius uniform in t does not give the convergence results presented here: It is well known that $\rho(A) < 1$ implies the existence of a vector norm with induced matrix norm such that $\|A\| < 1$, but this norm depends on A. So $\rho(A_t) \leq c < 1$ for some c for all $t \in \mathbb{N}_0$ does not imply the existence of a *single* norm such that $\|A_t\| < 1$ for all t. We emphasize that the result is purely qualitative, giving no explicit guidance to the choice of the learning rates. The main advantage of our approach is the clearness of the proof, only computation of eigenvalues is needed once the iteration has been written in terms of T and $\bar{T}$. These calculations are much more simple than the lengthy estimates in [KB15], [RKK19] and [Che+19].

4.4.1 Theoretical Work

Let $w \in \mathbb{R}^n$ be the weights of the function $f(w) \in C^2(\mathbb{R}^n, \mathbb{R})$, which has to be minimized. We also define $g(w) := \nabla f(w) \in \mathbb{R}^n$ as the gradient of f and the state variable of our dynamical system $x = (m, v, w)$. With these definitions we can rewrite the ADAM-Optimizer as a system of the form (4.4).

$$\begin{aligned} m_{t+1} &:= \beta_1 m_t + (1-\beta_1)\, g(w_t) \in \mathbb{R}^n \\ v_{t+1} &:= \beta_2 v_t + (1-\beta_2)\, g(w_t)^2 \in \mathbb{R}^n \\ w_{t+1} &:= w_t - \alpha \frac{\sqrt{1-\beta_2^{t+1}}}{\left(1-\beta_1^{t+1}\right)} \frac{m_{t+1}}{\sqrt{v_{t+1}+\epsilon^2}} \in \mathbb{R}^n \end{aligned} \tag{4.6}$$

So the ADAM optimizer can be written as the iteration of a time-variant dynamical system $x_{t+1} = T(t, x_t; p) \in \mathbb{R}^{3n}$ with $T = T(t, x; p) = T(t, m, v, w; \alpha, \beta_1, \beta_2, \epsilon)$ to express the dependence of the iteration on the state $x = (m, v, w)$ and the hyperparameters $p = (\alpha, \beta_1, \beta_2, \epsilon)$. We split the system in an autonomous and a non-autonomous part

$$x_{t+1} = T(t, x_t; p) = \bar{T}(x_t; p) + \Theta(t, x_t; p) \tag{4.7}$$

with

$$\bar{T}\left(x_t; p\right) = \begin{bmatrix} \beta_1 m_t + (1-\beta_1)\, g\left(w_t\right) \\ \beta_2 v_t + (1-\beta_2)\, g\left(w_t\right)^2 \\ w_t - \alpha \frac{m_{t+1}}{\sqrt{v_{t+1}+\epsilon^2}} \end{bmatrix} \tag{4.8}$$

and

$$\Theta\left(t, x_t; p\right) = \begin{bmatrix} 0 \\ 0 \\ -\alpha\left(\frac{\sqrt{1-\beta_2^{t+1}}}{1-\beta_1^{t+1}} - 1\right) \frac{m_{t+1}}{\sqrt{v_{t+1}+\epsilon^2}} \end{bmatrix} \tag{4.9}$$

To avoid lengthy expressions we use m_{t+1} and v_{t+1} as an abbreviation for the updated terms instead of the filters depending on m_t, $g(w_t)$ and v_t. The autonomous system is ADAM without bias correction, the disturbance term Θ adds bias correction which leads to a non-autonomous system. The Jacobian matrix of the autonomous system (4.8) is

$$J_{\bar{T}}\left(m_t, v_t, w_t\right) = \begin{bmatrix} \beta_1 I & 0 & (1-\beta_1)\, \nabla_w g\left(w_t\right) \\ 0 & \beta_2 I & \frac{\partial v_{t+1}}{\partial w_t} \\ \frac{\partial w_{t+1}}{\partial m_t} & \frac{\partial w_{t+1}}{\partial v_t} & \frac{\partial w_{t+1}}{\partial w_t} \end{bmatrix} \tag{4.10}$$

with

$$\begin{aligned} \frac{\partial v_{t+1}}{\partial w_t} =& 2\,(1-\beta_2)\, \operatorname{diag}(g\left(w_t\right)) \nabla_w g\left(w_t\right) \\ \frac{\partial w_{t+1}}{\partial m_t} =& -\alpha \operatorname{diag}\left(\frac{\beta_1}{\sqrt{v_{t+1}+\epsilon^2}}\right) \\ \frac{\partial w_{t+1}}{\partial v_t} =& \frac{\alpha\beta_2}{2} \operatorname{diag}\left(\frac{m_{t+1}}{\left(v_{t+1}+\epsilon^2\right)^{\frac{3}{2}}}\right) \\ \frac{\partial w_{t+1}}{\partial w_t} =& I - \alpha\left((1-\beta_1) \operatorname{diag}(v_{t+1}+\epsilon^2)^{-\frac{1}{2}} - \operatorname{diag}\left(\frac{m_{t+1}}{\left(v_{t+1}+\epsilon^2\right)^{\frac{3}{2}}} g\left(w_t\right)\right)\right) \nabla_w g\left(w_t\right) \end{aligned}$$

We have the following simple observation:

Lemma 4.3. *Consider a critical point $w_\star$ for f, $\nabla f(w_\star) = 0$. Then $x_\star = (0, 0, w_\star)^\perp$ is a fixed point for (4.8) and (4.6).*

Proof. We start the iteration with $w_0 = w_\star$, $v_0 = 0$ and $m_0 = 0$, i.e. $x_0 = (0, 0, w_\star)$. Then (4.8) gives $x_1 = T(x_0) = x_0$, and inductively $x_t = x_0$ for all t. The same holds for (4.6). □

Now we investigate the stability of this equilibrium with the goal of asymptotic stability for local minima $w_\star$. The analysis is simplified because the m and v components of $x_\star$ are 0. So we reach the following Jacobian:

$$J_{\bar{T}}(0,0,w_\star) = \begin{bmatrix} \beta_1 I & 0 & (1-\beta_1)\nabla_w g(w_\star) \\ 0 & \beta_2 I & 0 \\ \frac{-\alpha\beta_1}{\epsilon} I & 0 & I - \frac{\alpha(1-\beta_1)}{\epsilon}\nabla_w g(w_\star) \end{bmatrix} \tag{4.11}$$

Theorem 4.4. *Let $J_{\bar{T}}(m,v,w) \in \mathbb{R}^{3n\times 3n}$ be the Jacobian of system (4.8) and $w_\star \in \mathbb{R}^n$ a minimum of f with positive definite Hessian $\nabla_w^2 f(w_\star) = \nabla_w g(w_\star)$. Denote $\mu_i \in \mathbb{R}$ with $i = 1,\dots,n$ the i-th eigenvalue of $\nabla_w g(w_\star)$, $\varphi_i = \frac{\alpha\mu_i}{\epsilon}(1-\beta_1)$ and all other parameters are defined as in Algorithm 6. Then $J_{\bar{T}}(0,0,w_\star)$ has the eigenvalues, for $i = 1,\dots,n$:*

$$\begin{aligned} \lambda_{1,i} &= \beta_2 \\ \lambda_{2,i} &= \frac{(\beta_1 + 1 - \varphi_i) + \sqrt{(\beta_1+1-\varphi_i)^2 - 4\beta_1}}{2} \\ \lambda_{3,i} &= \frac{(\beta_1 + 1 - \varphi_i) - \sqrt{(\beta_1+1-\varphi_i)^2 - 4\beta_1}}{2} \end{aligned}$$

Proof. We see that $J_{\bar{T}}(0,0,w_\star)$ has the n-fold eigenvalue β_2. So we can drop second block row and column of $J_{\bar{T}}$ and investigate the eigenvalues of

$$\begin{bmatrix} \beta_1 I & (1-\beta_1)\nabla_w g(w_\star) \\ -s\beta_1 I & I - s(1-\beta_1)\nabla_w g(w_\star) \end{bmatrix} =: \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

where we use the abbreviation $s := \frac{\alpha}{\epsilon}$. B and D are symmetric since $\nabla_w g(w_\star)$ is the Hessian of f. By the spectral theorem we can diagonalize B as $B = Q\Lambda Q^\perp$ with an orthogonal matrix Q and a diagonal matrix of eigenvalues Λ. Analogously holds $D = I - Q\Lambda Q^\perp = Q(I-\Lambda)Q^\perp$. We make a similarity transformation with $\tilde{Q} := \begin{bmatrix} Q & 0 \\ 0 & Q \end{bmatrix} \in \mathbb{R}^{2n\times 2n}$. This leaves the eigenvalues unchanged and gives

$$\tilde{Q}^\perp \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tilde{Q} = \begin{bmatrix} \beta_1 I & (1-\beta_1)\mu_i I \\ -s\beta_1 I & I - s(1-\beta_1)\mu_i \end{bmatrix}$$

with μ_i the i-th eigenvalue of the Hessian. Eigenvalues does not change in similarity transformations, so we can also calculate the eigenvalues of our new block matrix with

four diagonal sub matrices.

$$\begin{aligned}&\det\begin{bmatrix}(\beta_1-\lambda)\,I & (1-\beta_1)\,\mu_i I\\ -s\beta_1 I & (1-s\,(1-\beta_1)\,\mu_i-\lambda)\,I\end{bmatrix}\\ =&\det\left((\beta_1-\lambda)\,(1-s\,(1-\beta_1)\,\mu_i-\lambda)\,I+(1-\beta_1)\,s\beta_1\mu_i I\right)\overset{!}{=}0\end{aligned}$$

Therefore the matrix is a diagonal matrix, we can conclude:

$$\det\left(J_{\bar{T}}\,(0,0,w_\star)-\lambda I\right)=\prod_{i=1}^{n}(\beta_1-\lambda)\,(1-s\,(1-\beta_1)\,\mu_i-\lambda)+(1-\beta_1)\,s\beta_1\mu_i$$

Each factor can be written as

$$\lambda^2-(1-s\,(1-\beta_1)\,\mu_i+\beta_1)\,\lambda+\beta_1\overset{!}{=}0$$

and following the statement

$$\lambda_{23,i}=0.5\left(1-s\,(1-\beta_1)\,\mu_i+\beta_1\pm\sqrt{(1-s\,(1-\beta_1)\,\mu_i+\beta_1)^2-4\beta_1}\right)$$

is true. □

Combining Theorem 4.4 and 4.1, we still have to show that $|\lambda_{j,i}|<1$ holds, then the spectral radius for the Jacobian is smaller than 1 and we prove the local convergence.

Theorem 4.5. *Let the parameters be defined as in Theorem 4.4 and inequality (4.3) holds, then* $\rho\left(J_{\bar{T}}\,(0,0,w_\star)\right)<1$.

Proof. We already have calculate the eigenvalues of the Jacobian in Theorem 4.4. With these we can easily see that $|\lambda_1|=|\beta_2|<1$ is satisfied per the requirements of Algorithm 6. Therefore we only have to look at:

$$\lambda_{23,i}=\frac{1+\beta_1-\varphi_i\pm\sqrt{(1+\beta_1-\varphi_i)^2-4\beta_1}}{2}$$

We define $\varphi_i:=\frac{\alpha\mu_i}{\epsilon}\,(1-\beta_1)$ and simplify the eigenvalue.

$$|\lambda_{23,i}|=\frac{1}{2}\left|(1+\beta_1-\varphi_i)\pm\underbrace{\sqrt{(1+\beta_1-\varphi_i)^2-4\beta_1}}_{①}\right|$$

First we look at upper bound of the eigenvalues. For this we take term ① combined with

the regrets for φ_i. It should be noted that $\varphi_i > 0$ holds since $0 < \alpha, \epsilon, \mu_i$ and $0 < \beta < 1$. Recall that μ_i is the ith-eigenvalue of the Hessian and we are in the minimum $x_\star$. Therefore the Hessian has to be positive definite and the estimation $0 < \mu_i$ holds true for all $i = 1, \ldots, n$. In addition we have inequality (4.3) which corresponds with our abbreviations to $\max_{i=1,\ldots,n} \varphi_i \leq 2\beta_1 + 2$. Thus φ_i is bounded by $0 < \varphi < 2\beta_1 + 2$ and we can estimate the eigenvalues of the Jacobian.

$$\sqrt{(1+\beta_1-\varphi_i)^2 - 4\beta_1} < \sqrt{(1+\beta_1)^2 - 4\beta_1} = \pm(1-\beta_1) \tag{4.12}$$

So if we put this in $\lambda_{23,i}$ we have the inequality $|\lambda_{23,i}| < \frac{1}{2}|1+\beta_1-\varphi_i \pm (1-\beta_1)|$. Easy to see are the two cases:

$$\begin{aligned} &|\lambda_{23,i}| < 1 && \text{with } + \\ &|\lambda_{23,i}| < \beta_1 < 1 && \text{with -} \end{aligned}$$

In both cases we see that the eigenvalues are smaller than 1 in absolute value. To show the lower bound $\lambda_{23,i} > -1$, we look again at term ①.

$$\underbrace{\sqrt{(1+\beta_1-\varphi_i)^2 - 4\beta_1}}_{\in \mathbb{C}\setminus\mathbb{R}} = i\sqrt{4\beta_1 - (1+\beta_1-\varphi_i)^2}$$

Then we can write:

$$\begin{aligned} |\lambda_{23,i}| =& \frac{1}{2}\sqrt{(1+\beta_1-\varphi_i)^2 + 4\beta_1 - (1+\beta_1-\varphi_i)^2} \\ =& \sqrt{\beta_1} < 1 \end{aligned}$$

The last inequality is given by the requirements of Theorem 4.5 and so we proved the whole Theorem. □

To be able to apply the results to the ADAM, we require some statements about the convergence of fixed point with perturbation.

Theorem 4.6. ***Convergence to fixed point with perturbation*** *Let $M \subset \mathbb{R}^n$ be a complete set, $\bar{T} : M \to M$ Lipschitz continuous with $L < 1$, $x_\star \in M$ the unique fixed point of $\bar{T}$. Assume $B_r(x_\star) \subset M$ for some $r > 0$. Recall that the non-autonomous system (4.7) is defined by*

$$\tilde{x}_{t+1} = T(\tilde{x}_t) := \bar{T}(\tilde{x}_t) + \Theta(t, \tilde{x}_t)$$

for $\Theta : \mathbb{N}_0 \times M \to \mathbb{R}^n$ *with the bound* $\|\Theta(t, \tilde{x})\| \leq C\beta^t \|\tilde{x} - x_\star\|$ *for all* $\tilde{x}_t \in M$, $t \in \mathbb{N}_0$ *for some* $C \geq 0$ *and* $0 < \beta < 1$. *Then there exists* $\varepsilon > 0$ *such that for all* $\tilde{x}_0 \in M$ *with* $\|\tilde{x}_0 - x_\star\| < \varepsilon$ *the iteration defined by (4.7) is well-defined, i.e. stays in* M, *and converges to* $x_\star$.

Proof. Let $x = x(\cdot, \tilde{x}_0)$ be the solution of the undisturbed iteration $x_{t+1} = \bar{T}(x_t)$ with initial condition $\tilde{x}_0$, $\tilde{x} = \tilde{x}(\cdot, \tilde{x}_0)$ the corresponding solution of (4.7). We define $e_t := \|\tilde{x}_t - x_\star\|$, and estimate using the assumptions

$$\begin{aligned} e_{t+1} =& \left\|\bar{T}(\tilde{x}_t) + \Theta(t, \tilde{x}_t) - x_\star\right\| \\ =& \left\|\bar{T}(\tilde{x}_t) - \bar{T}(x_\star) + \Theta(t, \tilde{x}_t)\right\| \\ \leq& \left\|\bar{T}(\tilde{x}_t) - \bar{T}(x_\star)\right\| + \|\Theta(t, \tilde{x}_t)\| \\ \leq& L \|\tilde{x}_t - x_\star\| + C\beta^t \|\tilde{x}_t - x_\star\| \\ =& (L + C\beta^t) e_t \end{aligned}$$

Choosing t large enough, we get $0 < L + C\beta^t \leq \tilde{L} < 1$ for all $t \geq K$ because $\beta, L < 1$. Then

$$e_t \leq \left(\prod_{k=1}^{K} (L + C\beta^k)\right) \tilde{L}^{t-K} e_0 =: \tilde{C}\tilde{L}^{t-K} e_0$$

with $\tilde{C}$ independent of $\tilde{x}_0$. So e_t converges to 0 exponentially.

The arguments so far have only been valid if $\tilde{x}_t \in M$, i.e. the iteration is well defined. But choosing $\tilde{x}_0$ such that $e_0 = \|\tilde{x}_0 - x_\star\| < \frac{r}{\tilde{C}}$ small enough that we can achieve $e_t \leq \tilde{C} e_0 < r$. □

Corollary 4.7. *Let the parameters be defined as in Theorem 4.4 and such that* $\frac{\alpha \max_{i=1}^n(\mu_i)}{\epsilon} \cdot (1 - \beta_1) < 2\beta_1 + 2$ *holds for* $i = 1, \ldots, n$, *then Algorithm 6 converges locally with exponential rate of convergence.*

Proof. Consider the non-autonomous system (4.7) with $\bar{T}(x_t; p)$ and $\Theta(t, x_t; p)$ as defined in equations (4.8) and (4.9). The Hessian of f is continuous, so the gradient of f is locally Lipschitz with some constant $L > 0$, $\|g(w_1) - g(w_2)\| \leq L \|w_1 - w_2\|$ for all w_1, w_2 in some neighborhood of $w_\star$. Let all other parameters be defined as in Theorem 4.4,

especially $\frac{\alpha \max_{i=1}^{n}(\mu_i)}{\epsilon}(1-\beta_1) < 2\beta_1 + 2$. Using $m_\star = 0$ and $g(w_\star) = 0$ we estimate

$$\begin{aligned}
\|\Theta(t,x;p)\| =& \alpha \left| \frac{\sqrt{1-\beta_2^{t+1}}}{1-\beta_1^{t+1}} - 1 \right| \frac{\|\beta_1 m + (1-\beta_1) g(w)\|}{\sqrt{\beta_2 v + (1-\beta_2) g(w)^2 + \epsilon^2}} \\
\leq& \frac{\alpha}{\epsilon} \left| \frac{\sqrt{1-\beta_2^{t+1}} - \left(1-\beta_1^{t+1}\right)}{\left(1-\beta_1^{t+1}\right)} \right| \|\beta_1 m + (1-\beta_1) g(w)\| \\
\leq& \frac{\alpha}{\epsilon(1-\beta_1)} \left| \frac{\left(1-\beta_2^{t+1}\right) - \left(1-\beta_1^{t+1}\right)^2}{\sqrt{1-\beta_2^{t+1}} + \left(1-\beta_1^{t+1}\right)} \right| (\beta_1 \|m\| + (1-\beta_1) \|g(w)\|) \\
\leq& \frac{C}{4} \left| \left(1-\beta_2^{t+1}\right) - \left(1-\beta_1^{t+1}\right)^2 \right| (\beta_1 \|m - m_\star\| + (1-\beta_1) \|g(w) - g(w_\star)\|) \\
\leq& \frac{C}{4} \left| -\beta_2^{t+1} - 2\beta_1^{t+1} + \beta_1^{2(t+1)} \right| (\beta_1 \|m - m_\star\| + (1-\beta_1) L \|w - w_\star\|) \\
\leq& C\beta^{t+1} (\beta_1 \|m - m_\star\| + (1-\beta_1) L \|w - w_\star\|)
\end{aligned}$$

where we have used the Lipschitz continuity of g, and set $\beta = \max\{\beta_1, \beta_2, \beta_1^2\}$, $C := \frac{4\alpha}{\epsilon(1-\beta_1)\left(\sqrt{1-\beta_2}+(1-\beta_1)\right)}$. The term $\beta_1 \|m - m_\star\| + (1-\beta_1) L \|w - w_\star\|$ corresponds to a norm

$$\|(\tilde{m}, \tilde{w})\|_* := \beta_1 \|\tilde{m}\| + (1-\beta_1) L \|\tilde{w}\|, \qquad \tilde{m}, \tilde{w} \in \mathbb{R}^n$$

on $\mathbb{R}^{2n}$ (which does not depend on $w_\star$). By the equivalence of norms in finite dimensional spaces we can estimate $\|(\tilde{m}, \tilde{w})\|_* \leq \tilde{C} \|(\tilde{m}, \tilde{w})\|$ for some $\tilde{C} > 0$. We continue the estimate:

$$\begin{aligned}
\leq& C\beta^{t+1} \tilde{C} \|(m - m_\star, w - w_\star)\| \\
\leq& (C\beta\tilde{C})\beta^t \|x - x_\star\| =: \bar{C}\beta^t \|x - x_\star\|
\end{aligned}$$

for some $\bar{C} > 0$. With this estimate and Theorem 4.6, it is sufficient to prove exponential stability of a fixed point of $\bar{T}$. By Theorem 4.5 we get $\rho\left(J_{\bar{T}}(0,0,w_\star)\right) < 1$. Thus with Theorem 4.1 the fixed point $(0,0,w_\star)$ corresponding to the minimum $w_\star$ is locally exponentially stable, and Theorem 4.6 gives local exponential convergence of the non-autonomous system $T(t,x)$, i.e. the ADAM algorithm. □

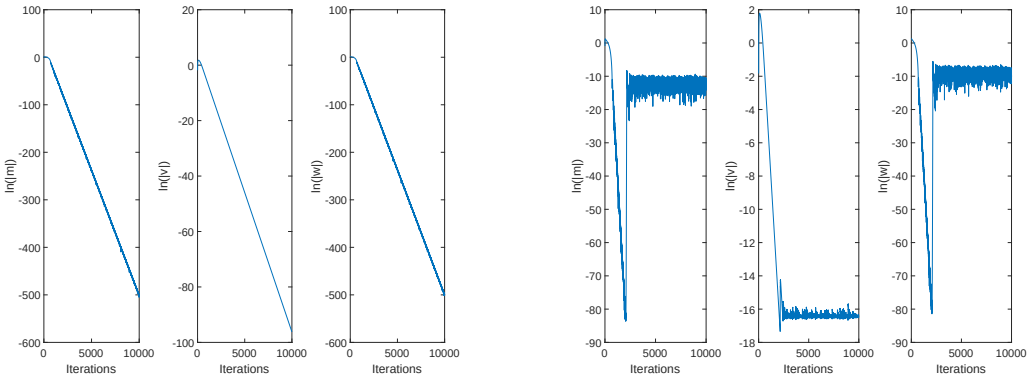
4.4.2 Numerical Experiments

The analysis made in the previous subsection will now be empirically substantiated. For this purpose, we first examine the behavior of the solution w (or v and m) close

to the tipping point of inequality (4.3). Then we consider the convergence behavior with respect to the hyperparameters of the ADAM, inequality (4.3) and the original convergence condition from [KB15]. The functions used in the experiments are chosen randomly and intentionally simple. The hyperparameters are largely based on those proposed by [KB15].

4.4.2.1 Numerical Convergence

The convergence proof in the subsection before only shows the local convergence under the hyperparameter bound (4.3). Whether the boundary is strict or whether there are elements outside this boundary that also converge was not answered. To observe the numerical behavior of ADAM, we choose $f(x) = \frac{x^2}{2} + 10, \mathbb{R} \to \mathbb{R}$ with the hyperparameters $\beta_1 = 0.9, \beta_2 = 0.99, \alpha = 0.01, m_0 = 0, v_0 = 0$ and $w_0 = 4$ similar to those of [KB15]. With $\epsilon = 10^{-2}$ inequality (4.3) holds and ADAM shows exponential convergence behavior (see Figure 25a). By choosing $\epsilon = 10^{-8}$ which clearly breaks inequality (4.3), we see



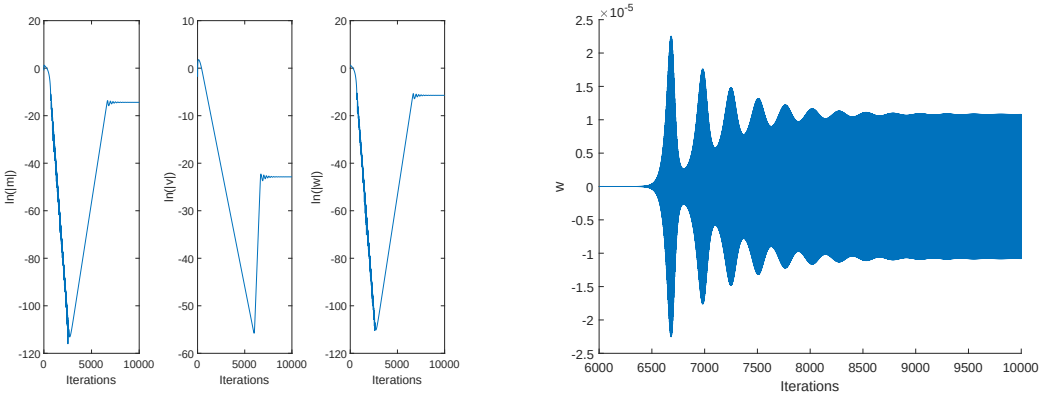
(a) Exponential convergence behavior

(b) Chaotic behavior

Figure 25: Behavior near the equality of inequality (4.3)

oscillating behavior as mentioned in [BW19b] (see Figure 25b). If we solve inequality (4.3) with the assumed hyperparameters to ϵ, we reach the bound $\epsilon < 2.63158 \cdot 10^{-4}$. When slowly reducing ϵ, one can recognize the evolution of chaotic behavior. Already at $\epsilon = 2.62936 \cdot 10^{-4}$ the exponential convergence is disturbed (see Figure 26a) and w starts to jump around $w_\star$ (see Figure 26b).

We could not find an ϵ with which our variant converges and the original variant of [KB15] does not converge or vice versa.



(a) Chaotic behavior near boundary

(b) Chaotic behavior of w near boundary

Figure 26: Different behavior of the ADAM

4.4.2.2 Solution Behavior

To compare our requirements for convergence to the requirements taken by [RKK19] or [KB15], we make some empirical experiments. First, we look at the different requirements to the hyperparameter.

$$\beta_1 < \sqrt{\beta_2} \tag{4.13}$$

$$\beta_1^2 < \sqrt{\beta_2} \tag{4.14}$$

Inequality (4.3) describes the needed requirement. Problematically in this estimation is that we need the maximum eigenvalue of $(1-\beta_1)\nabla_w g(w_\star)$ and consequently $w_\star$. Therefore our estimation is an a posteriori estimation. But with (4.3) we learn something about the relationship between the hyperparameters. $\frac{\alpha}{\sqrt{\epsilon}}$ has to be very small to fulfill inequality (4.3). With a small α or a big ϵ we always make the weight change smaller and so we do not jump over $w_\star$. Inequality (4.13) was presented in [RKK19] and inequality (4.14) was originally presented in [KB15]. Both are a priori estimations for the hyperparameters.

To show the behavior of all estimations we set up the following experiments. In Experiment 4.8 and 4.9 we want to minimize $f(w_t) := w_t^4 + w_t^3$ with the minimum $w_\star = -\frac{3}{4}$. In Experiment 4.10 we minimize the multidimensional function $f(w_{t,1}, w_{t,2}) := (w_{t,1}+2)^2 \cdot (w_{t,2}+1)^2 + (w_{t,1}+2)^2 + 0.1\,(w_{t,2}+1)^2$ with the minimum $w_\star = (-2,-1)$. We run the ADAM optimizer 10000 times in every hyperparameter setting and if the last five iterations $w_{end} \in \mathbb{R}^5$ are near enough to the known solution $w_\star$ the attempt is declared as convergent. Near enough in this setting means that all components of w_{end}

Table 10: Color description for the convergence investigations

Color	Inequality (4.14) satisfied	Inequality (4.3) satisfied	ADAM finds solution
green	yes	yes	yes
blue	no	yes	yes
yellow	yes	no	yes
white	no	no	yes
black	yes	yes	no
cyan	no	yes	no
magenta	yes	no	no
red	no	no	no

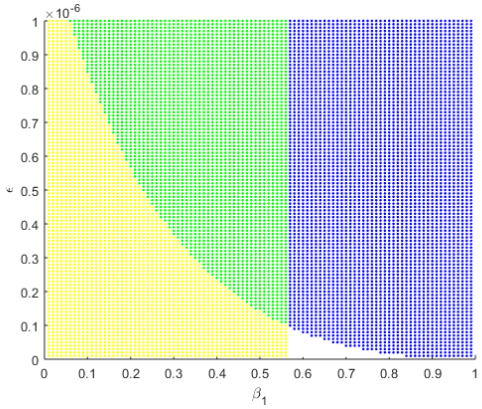
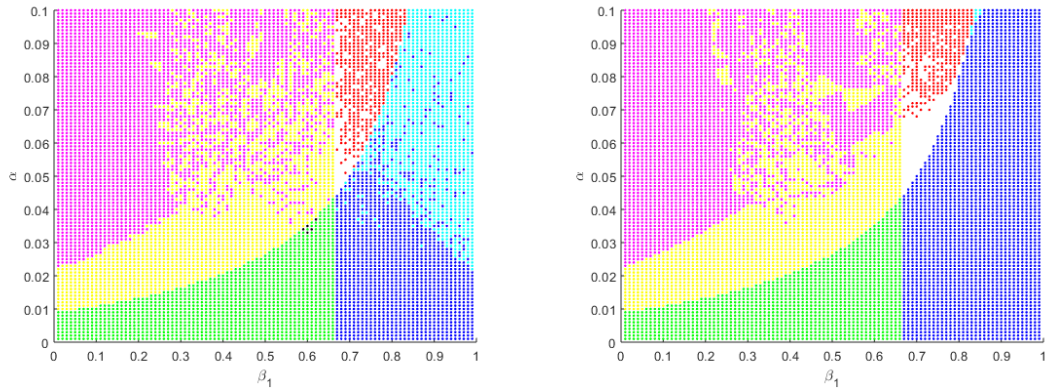


Figure 27: Iterating over ϵ and β_1

are contained in the interval $[w_\star - 10^{-2}, w_\star + 10^{-2}]$. The color coding of our experiments can be found in Table 10. To keep the clarity of our results we only compare the original ADAM inequality with our inequality. With inequality (4.13) we obtain similar figures.

Experiment 4.8. (Iterating over ϵ and momentum factor β_1)
First, we iterate over $\epsilon \in \{10^{-8}, \dots, 10^{-6}\}$ and $\beta_1 \in \{0.01, \dots, 0.99\}$. The other hyperparamteres are fixed $\alpha = 0.001$, $\beta_2 = 0.1$. This setting leads us to Figure 27. The white area – ADAM converge but no inequality is satisfied – is formed because we only talk about estimation and not clear boundaries. The blue and yellow area can be made larger or smaller by changing β_2 or α.

Experiment 4.9. (Iterating over learning rate α and momentum factor β_1)
In the second experiment we iterate over $\alpha \in \{0.001, \dots, 0.1\}$ and $\beta_1 \in \{0.01, \dots, 0.99\}$. $\beta_2 = 0.2$ and $\epsilon = 10^{-4}$ are fixed. With the starting point $x_0 = -2$ we reach Figure



(a) Iterating over α and β_1 (b) Iterating over α and β_1 with $x_0 = -\frac{3}{4}+10^{-9}$

Figure 28: Effect of local convergence

28a. In the magenta and the cyan area the ADAM method is not reaching the solution, although inequality (4.14) or (4.3) is satisfied. The ADAM is oscillating around the solution but do not reach them. The big difference is that the non-convergence in the cyan area is attributable to the fact that our proof only shows local convergence. By starting in $x_0 = -\frac{3}{4} + 10^{-9}$ the cyan area is almost complete blue (see Figure 28b). In contrary the magenta area does not change that much.

Experiment 4.10. (Experiment 4.10 with a two dimensional function $f(w_{t,1}, w_{t,2})$)
In the last experiment we use the same hyperparameters as in Experiment 4.8. Therefore we reach a similar looking Figure 29 by iterating over the parameters. The reason for

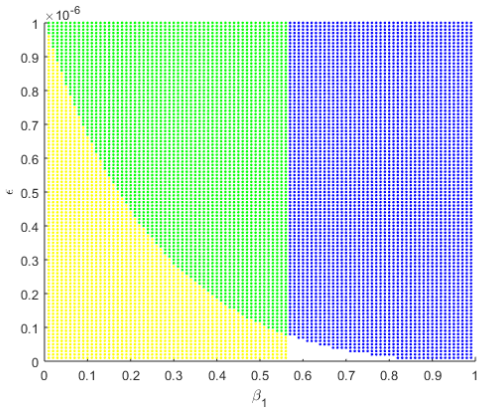


Figure 29: Experiment 4.10 :Iterating over ϵ and β_1.

the enlargement of the blue and green area is the different function $f(x)$, thus different eigenvalues in inequality (4.3). By observing the convergence behavior from each of the four differently colored areas in Figure 29, we can not spot big differences.

4.5 Non-Convergence Proof [12]

We have shown in the previous section that we can achieve local convergence for the ADAM. Now we want to consider global convergence. For this we show that for the objective function $f(w) = \frac{1}{2}cw^2$ with $c > 0$ 2-cycles occur for a wide range of hyperparameters in the ADAM iteration without bias correction and that iterations of the bias corrected algorithm converge to this limit cycle if it is stable. We proceed in several steps, analyzing simplified variants of ADAM first, then adding complexity in each step. The analysis uses Maple as a computer algebra system and some continuity and disturbance arguments because the naive approach of applying the `solve` command to find 2-cycles fails – the equations are too complicated.

1. We start with the scalar case $f(w) = \frac{1}{2}cw^2$, $c > 0$. We obtain analytical expressions for 2-limit-cycles $\bar{T}^2(x; \alpha, \beta_1, \beta_2, \varepsilon) = x$ of the autonomous system with $\varepsilon = 0$.

2. Calculating the eigenvalues of $\bar{T}^2$ we find that these do not depend on the learning rate α and the factor c in the objective function. For some typical values of the hyperparameters we give evidence that these limit cycles are often attractive. We have not managed to give analytical estimates for stable eigenvalues using Computer Algebra Software (CAS) so far.

3. Using the implicit function theorem we show that for a neighborhood of $\varepsilon_\star = 0$ there exists a unique limit cycle of the autonomous system with $\varepsilon > 0$. By continuity of the eigenvalues, these limit cycles are also attractive for ε small enough.

4. We apply a disturbance estimate to show that locally solutions of $T(t, x; p)$ converge to the limit cycles of $\bar{T}(x; p)$. The proof is essentially the same as in [BW19a, Theorem V.1.] and holds for cycles of any integer length.

4.5.1 Existence of 2-Limit-Cycles in ADAM

Step 1: We show that limit cycles of period 2 exist for ADAM without bias correction, i.e. the autonomous system $\bar{T}$. A 2-cycle corresponds to a non-constant solution of $\bar{T}(\bar{T}(x; p); p)) = x$, so we try to solve this system of equations with Maple. This fails, so we do not use arbitrary parameters but fix $\varepsilon = 0$. Now Maple succeeds and returns

$$\tilde{m} = \frac{1}{2}\frac{c\,(\beta_1 - 1)^2\,\alpha}{(\beta_1 + 1)^2}, \quad \tilde{v} = \frac{1}{4}\frac{\alpha^2\left(\beta_1^2 - 2\,\beta_1 + 1\right)c^2}{(\beta_1 + 1)^2}, \quad \tilde{w} = \frac{1}{2}\frac{\alpha\,(\beta_1 - 1)}{\beta_1 + 1} \tag{4.15}$$

[12] This chapter expands the results and presents proofs that are referenced in [BW19b].

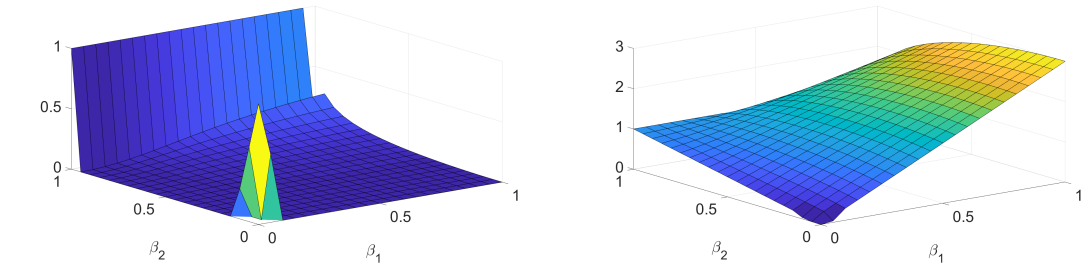


Figure 30: Absolute magnitude of real and complex eigenvalues

with $(-\tilde{m}, \tilde{v}, -\tilde{w})$ the other point on the 2-cycle. Note that $\tilde{m} \neq 0$, so we have a 2-cycle indeed. We abbreviate these points as $\tilde{x}_1$ and $\tilde{x}_2$. The v components of $\tilde{x}_1$ and $\tilde{x}_2$ are identical. This limit 2-cycle exists for all $\beta_1 \neq \pm 1$ that is for all reasonable ADAM hyperparameters. Maple also returns more 2-cycles depending on the roots of $2\beta_1\beta_2 - 2\beta_2^2 - 2\beta_1 + 2\beta_2$, we have not analyzed these. We could not determine cycles of greater period $q \in \mathbb{N}$ by solving $\bar{T}^q(x) = x$.

Step 2: But we can determine the stability of the limit cycle using the Eigenvalues of the Jacobian of $\bar{T}^2(\tilde{x}_1)$, which are the same as those of $\bar{T}^2(\tilde{x}_2)$. The Jacobian, with no easy interpretation, is computed by Maple as:

$$\begin{bmatrix} -(\beta_1+2)\beta_1 & 2\frac{(\beta_1+1)\beta_2}{\alpha c} & c(\beta_1-1)(\beta_1+2\beta_2-1) \\ -2(\beta_2-1)\alpha\beta_1 c & 3{\beta_2}^2 - 2\beta_2 & 2\frac{c^2(\beta_1-1)(\beta_1+3/2\beta_2-1/2)(\beta_2-1)\alpha}{\beta_1+1} \\ 2\frac{\beta_1(\beta_1+1)(1-\beta_1-2\beta_2)}{c(\beta_1-1)} & 2\frac{(2\beta_1+3\beta_2-1)(\beta_1+1)\beta_2}{c^2\alpha(\beta_1-1)} & 2{\beta_1}^2 + (8\beta_2-6)\beta_1 + 6{\beta_2}^2 - 6\beta_2 + 1 \end{bmatrix}$$

Using Maple we obtain a very lengthy expression for the eigenvalues which does *not* depend on α or c, but on β_1 and β_2 only. This is surprising as most algorithms show dependence on the learning rate, and one might assume that the behavior at a limit cycle is different at least for $\alpha \to 0$ and $\alpha \to \infty$.

Plotting the absolute values of the eigenvalues over β_1 and β_2 we can see that these limit cycles are often attractive, see Figure 30. Local attractivity holds if the absolute values are less than 1: This is the case for the real eigenvalue, see left plot, but the magnitude approaches 1 as β_2 approaches 1 – consider the standard value $\beta_2 = 0$ suggested in [KB15]. The pair of complex conjugate eigenvalues can be stable as well as unstable – unstable again for β_1 and β_2 near 1. This is good news: For typical β_1, β_2 the limit cycle will not turn up in numerical simulations as it is unstable. We have not managed to give analytical estimates for stable eigenvalues using CAS so far.

Step 3: Now we show that the limit cycle also exists for $\varepsilon > 0$ sufficiently small. We

fix the hyperparameters α, β_1, β_2 and consider the function

$$F(x, \varepsilon) = \bar{T}(\bar{T}(x; \alpha, \beta_1, \beta_2, \varepsilon); \alpha, \beta_1, \beta_2, \varepsilon) - x$$

Consider a state $\tilde{x}$ on a 2-limit-cycle for $\tilde{\varepsilon} = 0$ as in step 1, then $F(\tilde{x}, 0) = 0$ that is we have a zero of F. If $\frac{\partial F}{\partial x}(\tilde{x}, 0)$ is invertible, then the Implicit Function Theorem shows that in a neighborhood of $\tilde{\varepsilon} = 0$ there exists a unique $x(\varepsilon)$ with $F(x(\varepsilon), \varepsilon) = 0$. This zero of F corresponds to a 2-cycle of $\bar{T}$ with hyperparameter ε. (We always have $\varepsilon > 0$, but on a non-trivial 2-cycle we have shown that $v > 0$ by the explicit term for $\tilde{x}_1$ and $\tilde{x}_2$ in (4.15), so even a small reduction of ε would be allowed.) Calculating the determinant with Maple we get

$$\det\left(\frac{\partial F}{\partial x}(\tilde{x}, 0)\right) = 8(\beta_1 + 1)(\beta_2 - 1)(\beta_1 + \beta_2)$$

which is non-zero for all relevant $0 < \beta_1, \beta_2 < 1$. So the limit cycle exists for small $\varepsilon < \hat{\varepsilon}$, let us call this upper bound

$$\hat{\varepsilon} = \hat{\varepsilon}(\alpha, \beta_1, \beta_2) \tag{4.16}$$

for reference and by continuous dependence of the eigenvalues on the matrix, these limit cycles are also attractive.

Step 4: We apply a disturbance estimate to show that $\bar{T}(x; \alpha, \beta_1, \beta_2, \varepsilon)$ and $T(t, x; \alpha, \beta_1, \beta_2, \varepsilon)$ have asymptotically the same limit cycles. The following theorem is a variation of [BW19a, Theorem V.1.] and holds for cycles of any integer length. The proof is very similar and omitted for brevity. The difference between the variants is that here we do not use an estimate of the type $\|\Theta(t, \tilde{x})\| \leq C\beta^t \|\tilde{x} - x_\star\|$ where the fixed point $x_\star$ appears but rather an exponentially decaying term $C\beta^t$. Consequently we cannot show exponential stability of the 2-limit-cycle but only exponential convergence of trajectories nearby, with the constants depending on the initial value.

Theorem 4.11. *Let $X \subset \mathbb{R}^n$ be a closed set, $\|\cdot\|$ a norm on $\mathbb{R}^n$. Let $T : \mathbb{N}_0 \times X \to X$ be a mapping which is a contraction w.r.t. the second variable uniform in $t \in \mathbb{N}_0$, i.e. there exists $L < 1$ with*

$$\|T(t, x) - T(t, y)\| \leq L \|x - y\| \quad \forall x, y \in X, t \in \mathbb{N}_0$$

Furthermore assume that the difference between $T(t + 1, \cdot)$ and $T(t, \cdot)$ is exponentially bounded: There exist $C \geq 0$, $0 < \beta < 1$ such that

$$\|T(t + 1, x) - T(t, x)\| \leq C\beta^t \quad \forall x \in X, k \in \mathbb{N}_0$$

Then T has a unique fixed-point $x_\star$ in X: $T(t, x_\star) = x_\star$ for all $t \in \mathbb{N}_0$. For all $x_0 \in X$, the sequence defined by $x_{t+1} = T(t, x_t)$, $t \in \mathbb{N}_0$, converges to $x_\star$ exponentially.

To apply this theorem to the 2-limit-cycle we have to estimate the difference between two iterations of ADAM with and without bias correction:

$$\left\|T(t+1, T(t,x)) - \bar{T}(\bar{T}(x))\right\|$$

We use the fact from the proof of Corollary 4.7 that

$$\|\Theta(t,x)\| = \left\|T(t,x) - \bar{T}(x)\right\| \leq C\beta^t \|x - x_\star\|$$

as well as

$$\|\Theta(t,x)\| = \left\|T(t,x) - \bar{T}(x)\right\| \leq C\beta^t \|x\|$$

Using this we estimate

$$\begin{aligned}\left\|T(t+1, T(t,x)) - \bar{T}(\bar{T}(x))\right\| \leq& \left\|T(t+1, T(t,x)) - \bar{T}(T(t,x))\right\| + \left\|\bar{T}(T(t,x)) - \bar{T}(\bar{T}(x))\right\| \\ \leq& C\beta^{t+1} \|T(t,x)\| + L\left\|T(t,x) - \bar{T}(x)\right\|\end{aligned}$$

where L is a local Lipschitz constant near the limit cycle. The Lipschitz continuity exists because $\bar{T}$ is continuously differentiable. We continue the estimate

$$\begin{aligned}&\leq C\beta^{t+1} \|T(t,x)\| + LC\beta^t \|x\| \\ &= C\beta^{t+1} \left\|\bar{T}(x) + \Theta(t,x)\right\| + LC\beta^t \|x\| \\ &\leq C\beta^{t+1} \left\|\bar{T}(x)\right\| + C^2\beta^{t+1}\beta^t \|x\| + LC\beta^t \|x\| \\ &\leq \tilde{C}\beta^t \max\{\left\|\bar{T}(x)\right\|, \|x\|\}\end{aligned}$$

with $\tilde{C} \geq \max\{C^2, LC\}$. As we consider only states x near the limit cycle $\tilde{x}_1$, $\tilde{x}_2$ we can locally bound the term $\left\|\bar{T}(x)\right\|$ by continuity of $\bar{T}$.

We summarize our findings in the following theorem.

Theorem 4.12. *Consider the ADAM-Optimizer as defined in Algorithm 6 and objective function $f(w) = \frac{1}{2}cw^2$, $c > 0$. Then the algorithm is locally convergent under the assumptions stated in [BW19a]. However there exist solutions that converge to the 2-limit-cycles of the algorithm without bias correction; so the algorithm does not converge globally to the minimum $w_\star = 0$.*

Table 11: Parameters of the different limit cycles

Parameter	Experiment 4.13	Experiment 4.15	Experiment 4.16
c	10	1	1
α	0.001	0.5	0.8
β_1	0.9	0.2	0.5
β_2	0.999	0.5	0.6
ε	10^{-8}	10^{-6}	0.01
m_0	$-1.281144718 \cdot 10^{-5}$	0	0
v_0	$5.925207756 \cdot 10^{-8}$	0	0
w_0	$2.434174964 \cdot 10^{-5}$	ϵ_{mach}	ϵ_{mach}

4.5.2 Numerical Simulations: Discrete Limit Cycles

In [dG18, Proposition 3.3] the authors show the existence of a discrete limit cycle for the ADAM. This discrete limit cycle depends on the learning rate α and $\beta_1 = 0$. Therefore we demand $0 < \beta_1 < 1$ this limit cycle does not affect the local convergence proof of [BW19a]. But we found in some numerical experiments few limit cycles which alter the convergence of ADAM. Therefore, we will recall the hyperparameter bound (4.3)

$$\frac{\alpha \max_{i=1}^{n}(\mu_i)}{\epsilon}(1-\beta_1) < 2\beta_1 + 2$$

This bound is marked in the experiments with a red cross and depicts in every of our experiments the bifurcation position. The $\epsilon_{mach} = 2^{-52} \approx 2.2204 \cdot 10^{-16}$ is the machine accuracy of IEEE floating point arithmetic with double precision.

Experiment 4.13. (Non convergence with hyper parameters suggested by [KB15])
In the first experiment, we found a 2-limit-cycle close the hyper parameters suggested by [KB15]. A plot between the three main values m_t, v_t, w_t looks like a fountain and we can see that the values of m_t and w_t become more diffuse with increasing v_t (see Figure 31 left). By looking closely to w_t, it attracts attention that w_t is reaching the solution 0 but leaving it again (see Figure 31 right). One of the eigenvalues of the corresponding Jacobian is greater than 1, so the solution is not a stable 2-limit-cycle. The other real valued 2-limit-cycles also are not stable. Therefore we reach in Experiment 4.13 a limit-cycle with a higher order than 2 (see Figure 32). In the following we will only consider w and α. By the fact that m_t and v_t are only auxiliary variables depending on the history of w, they are less important than w and α. In order to clarify this aspect, reference is made at this point to page 3 in [KB15]. There is a definition for v_t without v_{t-1} and thus a definition of w without m and v is possible. In addition, the following remark gives an insight into the dependency from m_t to the history of w in 2-limit-cycles.

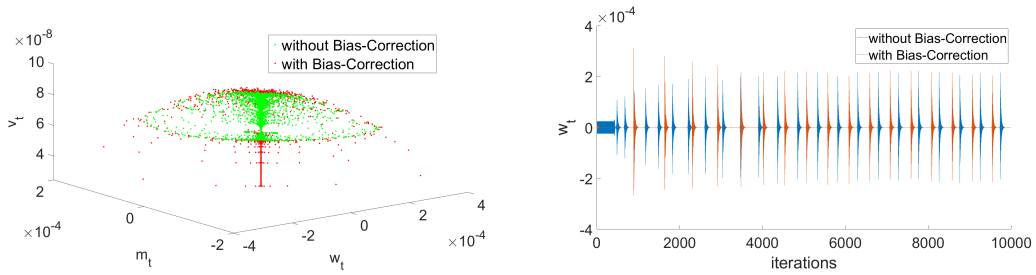


Figure 31: Discrete limit cycle with parameters suggested by [KB15] (Experiment 4.13)

Remark 4.14. We assume a 2-limit-cycle and therefore we can write $m_t = m_{t+2}$. With this knowledge, we can rewrite the m_t-update rule.

$$\begin{bmatrix} -\beta_1 & 1 \\ 1 & -\beta_1 \end{bmatrix} \begin{bmatrix} m_t \\ m_{t+1} \end{bmatrix} = (1-\beta_1) \begin{bmatrix} g(w_t) \\ g(w_{t+1}) \end{bmatrix}$$

Defining $\beta_1 \in (0,1)$ the system is uniquely solvable and thus m_t does not have more information for the system than w. The same applies to v_t.

If we are iterating over α from 10^{-4} to 0.001 we reach Figure 32 and see a evolution which looks like a Hopf bifurcation. With inequality (4.3) we can calculate exactly the coordinate of the bifurcation (see the red cross in Figure 32).

$$\alpha_{\text{Bifurcation}} = \frac{(2\beta_1 + 2)\sqrt{\epsilon}}{(1-\beta_1)\max_{i=1}^{n}(\mu_i)} = 0.00038$$

Experiment 4.15. (Non convergence with a good start point)
In the second experiment we can see that it is possible that even if we are starting closely to the solution $w_0 = \epsilon_{mach}$ we are ending in a stable cycle far away. By iterating over α from 10^{-4} to 0.01 we can see a pitchfork bifurcation of the ADAM (see Figure 33 right). Here, too, inequality (4.3) can be used to calculate the start of the two cycles (see red cross in Figure 33). With $\alpha = 0.5$ stable, the eigenvalues of the corresponding Jacobian are $\lambda_1 \approx 0.0113983$ and $\lambda_{2,3} \approx -0.7606667 \pm 0.5465392\,\mathrm{i}$. In absolute value all three eigenvalues are smaller than 1 and so we reach a stable 2-limit-cycle between $w_{2n-1} = 1/6$ and $w_{2n} = -1/6$ with $n \in \mathbb{N}$ (see Figure 33 left).

Experiment 4.16. (Non convergence with a big ϵ)
In contrast to the implicit function argument, Experiment 4.16 uses $\varepsilon = 0.01 \gg \epsilon_{mach}$. In Figure 34 on the left side one can see that starting at $\alpha = 0.6$ ADAM converges to a 2-limit-cycle. At around $\alpha = 0.7$ ADAM shows a chaotic behavior. On the right side

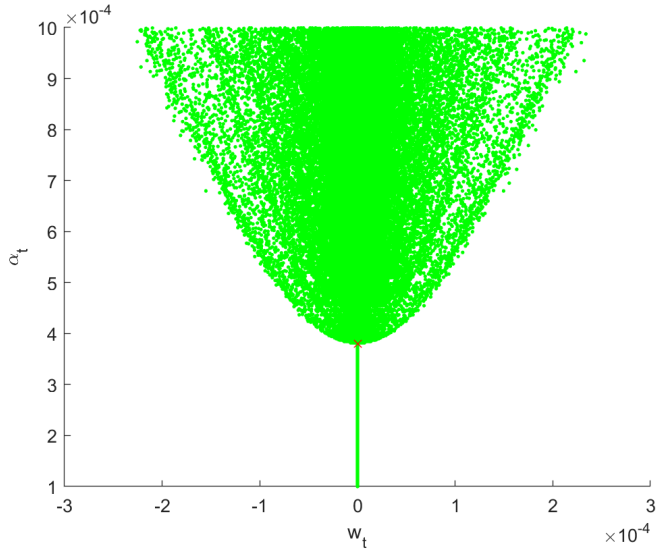


Figure 32: Hopf bifurcation (Experiment 4.13)

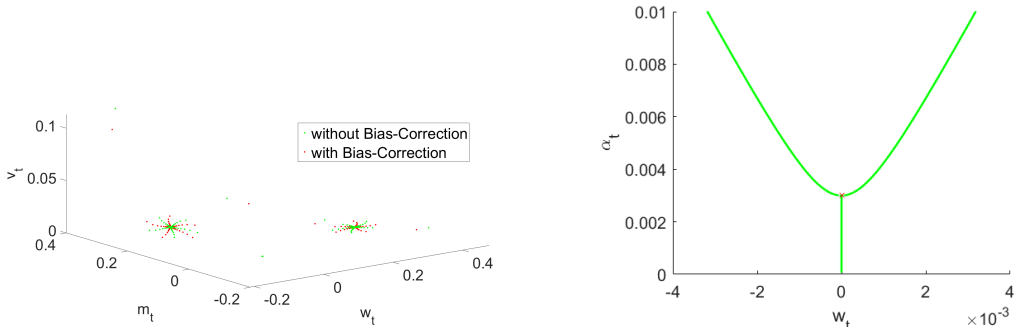


Figure 33: Discrete limit cycle starting near the solution (Experiment 4.15)

one can see the behavior of the parameters m, v and w at $\alpha = 0.8$. It visualizes the chaotic behavior and reminds of the shape of a Lorenz system.

Even if we want to minimize a multidimensional problem we can detect such a bifurcation. For example, with $f(w) : \mathbb{R}^2 \to \mathbb{R}, f(w) = w^{\perp} C w$ and

$$C = \begin{bmatrix} 1.1184 & 0.5841 \\ 0.5841 & 3.8816 \end{bmatrix} = Q^{\perp} \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix} Q$$

we obtain Figure 35. In every of our experiments the first bifurcation is exactly on the solved inequality (4.3) (see the red cross in Figure 32 - 35). This result is only an empirical proved assumption and an analytical proof would be desirable.

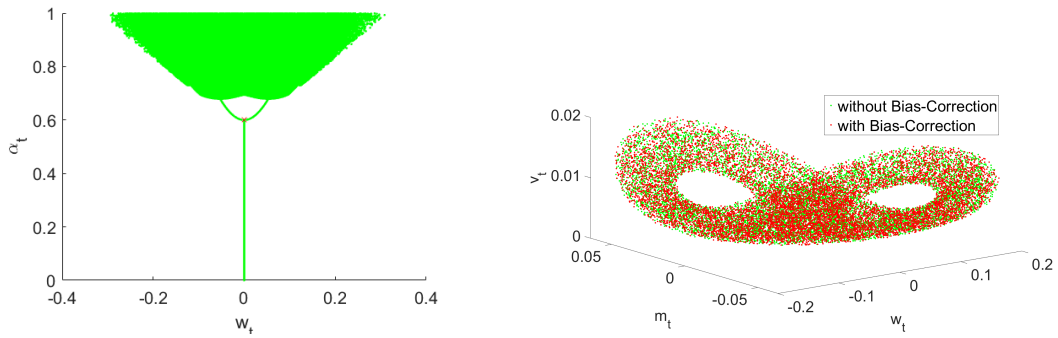


Figure 34: Discrete limit cycle with a large ε (Experiment 4.16)

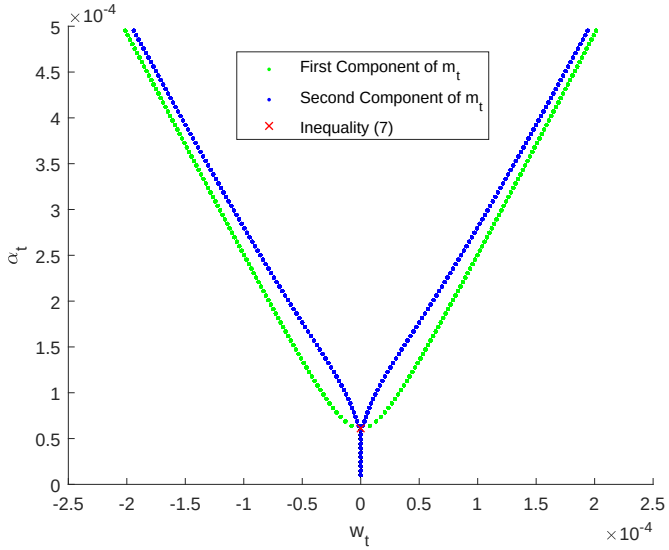


Figure 35: Bifurcation in a multidimensional optimization

4.6 Analysis by Lyapunov Exponent

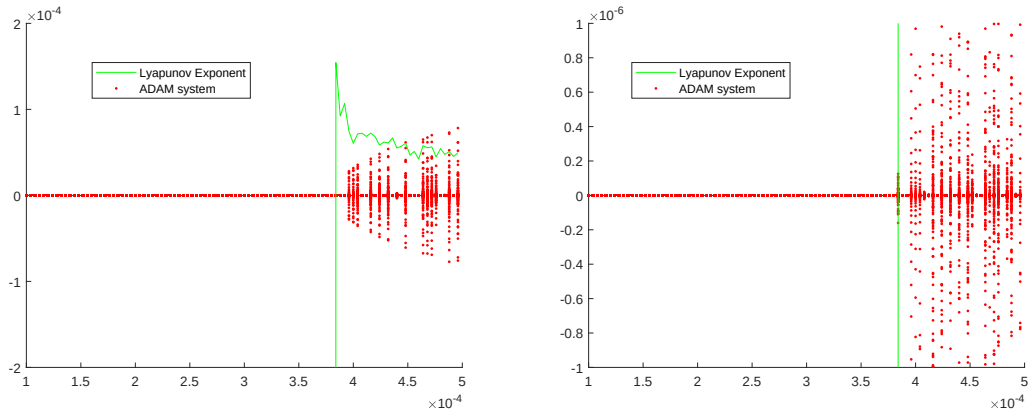
Lyapunov exponents are very useful in the study of chaotic behavior in dynamical systems. These measure the rates of convergence or divergence of nearby trajectories ([MT11]). Hence we now consider the Lyapunov exponents from the $w - \alpha$-graph, like in the experiments before. We fix α in each step, than we run the ADAM optimizer and have a look at the last 300 iterations. For each fixed α we calculate the largest Lyapunov exponent.[13] Therefore we are in the one-dimensional case. Wolf in [Wol+85] explains Table 12 in words. For this analysis we use the parameters and hyperparameters of Experiment 4.13 and 4.15 from Table 11. We recall that we found in Experiment 4.13

[13]For the numerical calculations we use Matlab with the Predictive Maintenance Toolbox and the included function *lyapunovExponent*.

Table 12: Description of Lyapunov Exponents

Sign of Lyapunov exponent	+	0	-
Dynamic of the system	chaos	marginally stable orbit	periodic orbit

a chaotic behavior after α reaches the equality of inequality (4.3). With this knowledge and Table 12 we have to assume that the largest Lyapunov exponent is negative by solving inequality (4.3) and positive afterwards. This assumption became true by the numerical calculation as we can see in Figure 36.

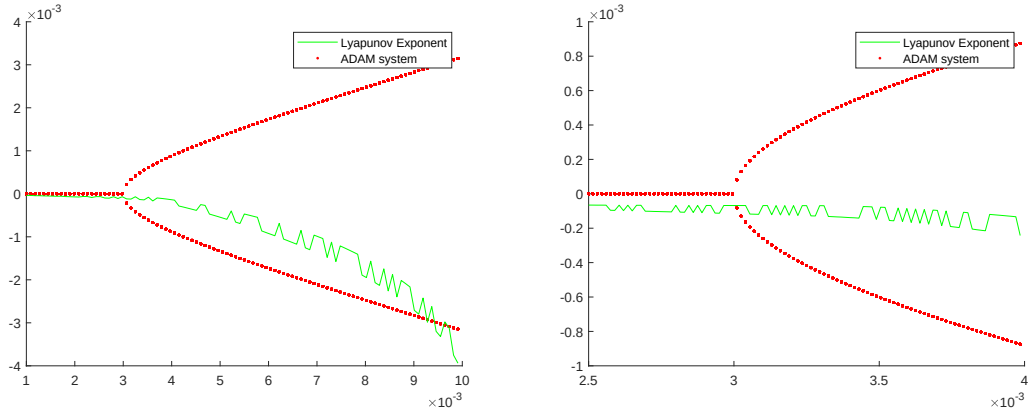


(a) ADAM with Lyapunov exponent

(b) Graphic left zoomed in

Figure 36: Experiment 4.13 with the Lyapunov exponents

In Experiment 4.15 the ADAM optimizer ends in a 2-limit-cycle. Therefore the cycle is stable we have to expect a completely negative Lyapunov exponent. The numerical calculations displayed in Figure 37 proves this assumption.



(a) ADAM with Lyapunov exponent

(b) Graphic left zoomed in

Figure 37: Experiment 4.15 with Lyapunov exponents

4.7 Original ADAM Formulation

We recall that we use the ADAM optimizer with $\frac{1}{\sqrt{v+\epsilon^2}}$ instead of $\frac{1}{\sqrt{v}+\epsilon}$ as suggested in [KB15]. As explained in Section 4.3, this makes only a mathematical difference. Algorithmically, one just wants to avoid dividing by zero, which happens in both variants. So that we can also make statements about the original version, we change line 5 from Algorithm 6.

Algorithm 7 ADAM Optimizer by Kingma and Ba[KB15]

Require: $\alpha \in \mathbb{R}^+, \epsilon \in \mathbb{R}, \beta_1, \beta_2 \in (0,1), w_0 \in \mathbb{R}^n$ and the function $f(w) \in C^2(\mathbb{R}^n, \mathbb{R})$

1: $m_0 = 0, v_0 = 0, t = 0$
2: **while** w not converged **do**
3: $m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla_w f(w_t)$
4: $v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla_w f(w_t)^2$
5: $w_{t+1} = w_t - \alpha \frac{\sqrt{1-\beta_2^{t+1}}}{(1-\beta_1^{t+1})} \frac{m_{t+1}}{\sqrt{v_{t+1}}+\epsilon}$
6: $t = t+1$
7: **end while**

4.7.1 Local Convergence[14]

In order to prove the convergence of the original version, reference is made at this point to [BCS18, Theorem 2.7]. Since [BCS18] is only available as a non-refereed preprint, the proof can be found in the appendix.

[14]This chapter expands the results and presents proofs that are referenced in [BW21].

Theorem 4.17. ***Exponential stability implies existence of a Lyapunov function, arbitrary fixed point***
Let $x_\star$ be an equilibrium point for the nonlinear system the autonomous system $x(t+1) = f(x(t))$ where $f : D \to \mathbb{R}^n$ is continuously differentiable and $D = \{x \in \mathbb{R}^n | \, \|x - x_\star\| < r\}$. Let k, λ and r_0 be positive constants with $r_0 < r/k$. Let $D_0 = \{x \in \mathbb{R}^n | \, \|x - x_\star\| < r_0\}$. Assume that the solution of the system satisfy

$$\|x(t, x_0)\| \leq k \|x_0 - x_\star\| e^{-\lambda t}, \forall x_0 \in D_0, \forall t \geq 0 \tag{4.17}$$

Then there there is a function $V : D_0 \to \mathbb{R}$ with

$$\begin{aligned} c_1 \|x - x_\star\|^2 \leq V(x) \leq c_2 \|x - x_\star\|^2 \\ V(f(x)) - V(x) \leq -c_3 \|x - x_\star\|^2 \\ |V(x) - V(y)| \leq c_4 \|x - y\| (\|x - x_\star\| + \|y - x_\star\|) \end{aligned}$$

Consider the ADAM optimizer without the bias correction as in system (4.5) that is with the term $\frac{1}{\sqrt{v+\epsilon^2}}$ in the w equation of $x_{t+1} = \bar{T}(x_t)$ with $x = [m, v, w]^\perp$. From Corollary 4.7 we know that $x_\star$ is locally exponentially stable. With small changes we can modify this system to the original ADAM formulation of [KB15] without bias correction.

$$\tilde{x}_{t+1} = \bar{T}(x_t) + h(x_t), \quad t \in \mathbb{N}_0, x_0 \in M \tag{4.18}$$

with the autonomous system $\bar{T}(x_t)$ as defined in equation (4.8) and

$$h(x_t) := \left[0, 0, \alpha m_{t+1} \left(\frac{1}{\sqrt{v_{t+1}} + \epsilon} - \frac{1}{\sqrt{v_{t+1} + \epsilon^2}} \right)^{-1} \right]^\perp \tag{4.19}$$

Note that we define the original ADAM without bias correction in dependence of the system (4.5) studied so far and as an autonomous system. By means of this consideration, we can prove the local convergence of system (4.18).

Corollary 4.18. *Assume the original ADAM without bias correction $\tilde{x}_{t+1} = \bar{T}(x_t) + h(x_t)$ as defined in equation (4.18) and $f : \mathbb{R}^n \to \mathbb{R}$ strictly convex with minimum $w_\star \in \mathbb{R}^n$. Assume $f \in C^2$ and $\nabla^2 f(w_\star)$ positive definite. Assume inequality (4.3) holds for the hyperparameters. Then the original ADAM without bias correction converges locally with exponential rate of convergence.*

Proof. We start the proof by considering the disturbance $h(x_t)$ from equation (4.19).

We can estimate the difference in $h(x_t)$ with

$$\begin{aligned}
\left|\frac{1}{\sqrt{v}+\epsilon}-\frac{1}{\sqrt{v+\epsilon^2}}\right| &= \left|\frac{\sqrt{v+\epsilon^2}-(\sqrt{v}+\epsilon)}{\sqrt{v+\epsilon^2}\,(\sqrt{v}+\epsilon)}\right| \\
&= \left|\frac{\left(\sqrt{v+\epsilon^2}-\sqrt{v}+\epsilon\right)\left(\sqrt{v+\epsilon^2}+\sqrt{v}+\epsilon\right)}{\sqrt{v+\epsilon^2}\,(\sqrt{v}+\epsilon)\left(\sqrt{v+\epsilon^2}+\sqrt{v}+\epsilon\right)}\right| \\
&= \left|\frac{2\sqrt{v}\epsilon}{\sqrt{v+\epsilon^2}\,(\sqrt{v}+\epsilon)\left(\sqrt{v+\epsilon^2}+\sqrt{v}+\epsilon\right)}\right| \\
&\leq \left|\frac{2\sqrt{v}\epsilon}{\sqrt{v}\epsilon\,(2\epsilon)}\right| = \frac{1}{\epsilon}
\end{aligned}$$

Therefore we can estimate the whole disturbance $\|h\,(x_t)\|$

$$\begin{aligned}
\|h\,(x_t)\| \leq & \frac{\alpha}{\epsilon}\,\|m_{t+1}-m_\star\| \\
= & \frac{\alpha}{\epsilon}\,\|\beta_1 m_t+(1-\beta_1)\,g\,(w_t)-m_\star\| \\
\leq & \frac{\alpha}{\epsilon}\,(\|\beta_1 m_t-m_\star\|+(1-\beta_1)\,\|g\,(w_t)\|) \\
\leq & \frac{\alpha}{\epsilon}\,(\beta_1\,\|m_t-m_\star\|+(1-\beta_1)\,\|g\,(w_t)-g\,(w_\star)\|) \\
\leq & \frac{\alpha}{\epsilon}\,(\beta_1\,\|m_t-m_\star\|+(1-\beta_1)\,L\,\|w_t-w_\star\|) \\
\leq & \frac{\alpha}{\epsilon}\beta_1\,\|m_t-m_\star\| \\
\leq & C\,(\epsilon)\,\|x_t-x_\star\| \qquad (4.20)
\end{aligned}$$

with $C\,(\epsilon)=\frac{\alpha}{\epsilon}$. For the next steps we need the following Theorem.

Theorem 4.19. ***Exponential convergence under disturbances*** *Suppose $f:\mathbb{R}^n\to\mathbb{R}^n$ Lipschitz continuous with L_f and the discrete time system $x_{t+1}=f(x_t),\quad x(0)=x_0$ Assume $f(x_\star)=x_\star$ and that $x_\star$ is a global exponentially stable fixed point. Assume $g:\mathbb{R}^n\to\mathbb{R}^n$ Lipschitz continuous with L_g, with $g(x_\star)=0$ and $\|g(x)\|\leq L_g\,\|x-x_\star\|$ for all $x\in\mathbb{R}^n$ and the discrete time system*

$$\tilde{x}_{t+1}=f(\tilde{x}_t)+g(\tilde{x}_t),\quad \tilde{x}(0)=\tilde{x}_0 \qquad (4.21)$$

Then, if L_g is small enough, $\tilde{x}_\star=x_\star$ is a global exponentially stable fixed point for (4.21) as well.

Proof. The converse Lyapunov theorem ensures the existence of a function V with the

following properties

$$\begin{aligned} c_1 \|x - x_\star\|^2 \leq V(x) \leq c_2 \|x - x_\star\|^2 \\ \Delta V(x) = V(x_{t+1}) - V(t) \leq -c_3 \|x - x_\star\|^2 \\ |V(x) - V(y)| \leq c_4 \|x - y\| (\|x - x_\star\| + \|y - x_\star\|) \end{aligned}$$

We show that for L_g small enough, $\tilde{V}(\tilde{x}) := V(\tilde{x})$ is also a Lyapunov function for (4.21). We use the tilde symbol to denote time derivatives along (4.21):

$$\begin{aligned} \tilde{\Delta}\tilde{V}(\tilde{x}) =& \tilde{V}(f(\tilde{x}) + g(\tilde{x})) - \tilde{V}(\tilde{x}) \\ =& V(f(\tilde{x}) + g(\tilde{x})) - V(\tilde{x}) \\ =& V(f(\tilde{x})) - V(\tilde{x}) + V(f(\tilde{x}) + g(\tilde{x})) - V(f(\tilde{x})) \end{aligned}$$

The properties of V and g give

$$\begin{aligned} \tilde{\Delta}\tilde{V}(\tilde{x}) \leq& -c_3 \|\tilde{x} - x_\star\|^2 + c_4 \|g(\tilde{x})\| (\|f(\tilde{x}) + g(\tilde{x}) - x_\star\| + \|f(\tilde{x}) - x_\star\|) \\ =& -c_3 \|\tilde{x} - x_\star\|^2 + c_4 \|g(\tilde{x})\| (\|f(\tilde{x}) - x_\star + g(\tilde{x})\| + \|f(\tilde{x}) - x_\star\|) \\ \leq& -c_3 \|\tilde{x} - x_\star\|^2 + c_4 L_g \|\tilde{x} - x_\star\| (\|f(\tilde{x}) - f(x_\star)\| + \|g(\tilde{x})\| + \|f(\tilde{x}) - f(x_\star)\|) \\ \leq& -c_3 \|\tilde{x} - x_\star\|^2 + c_4 L_g \|\tilde{x} - x_\star\| (L_f \|\tilde{x} - x_\star\| + L_g \|\tilde{x} - x_\star\| + L_f \|\tilde{x} - x_\star\|) \\ =& -(c_3 - c_4 L_g (L_f + L_g + L_f)) \|\tilde{x} - x_\star\|^2 \end{aligned}$$

So for L_g small enough we get exponential convergence. □

Thus, the first requirement of Theorem 4.19 is fulfilled. We know with the Theorems 4.1 and 4.5 that the undisturbed system (4.8) converges locally and exponentially to $x_\star$. With this knowledge and inequality (4.20), we can apply Theorem 4.19. It follows that $\tilde{x}_\star = x_\star$ is a global exponentially stable fixed point for the system (4.18). Thus follows the assertion of the Corollary. □

Involving the bias correction, we have to adjust system (4.18) to

$$\tilde{x}_{t+1} = T(x_t) + h(x_t) + \tilde{\Theta}(t, x) \tag{4.22}$$

with

$$\tilde{\Theta}(t, x) := \left[0, 0, \alpha \left(1 - \frac{\sqrt{1 - \beta_2^{t+1}}}{1 - \beta_1^{t+1}}\right) \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon}\right]^\perp \tag{4.23}$$

Corollary 4.20. *Assume the original ADAM with bias correction as the non-autonomous system $\tilde{x}_{t+1} = \bar{T}(x_t) + h(x_t) + \tilde{\Theta}(t,x)$ like defined in equation (4.22). All other requirements are adopted by Corollary 4.18. Then system (4.22) converges locally with exponential rate of convergence.*

Proof. Assume the non-autonomous system (4.9) with $\bar{T}(x_t)$, $h(x_t)$ and $\tilde{\Theta}(t,x)$ as defined in equations (4.8), (4.19) and (4.23). Since the estimation of $\left\|\tilde{\Theta}(t,x)\right\|$ is analogous to the estimation in the proof of Corollary 4.7, reference is made here to this proof and the estimation of $\left\|\tilde{\Theta}(t,x)\right\|$ is given by

$$\left\|\tilde{\Theta}(t,x)\right\| \leq C\beta^{t+1}\left(\beta_1 \|m - m_\star\| + (1-\beta_1) L \|w - w_\star\|\right)$$

Where $L > 0$ is the Lipschitz constant of f, $\beta = \max\{\beta_1, \beta_2, \beta_1^2\}$ and $C := \frac{4\alpha}{\epsilon(1-\beta_1)\left(\sqrt{1-\beta_2}+(1-\beta_1)\right)}$. Analogous to the proof of Corollary 4.7 we can shorten the inequality by the equivalence of norms in finite dimensional spaces. Thus follows

$$\left\|\tilde{\Theta}(t,x)\right\| \leq \bar{C}\beta^t \|x - x_\star\|$$

for some $\bar{C} > 0$. With Corollary 4.18 we prove the exponential stability of a fixed point of $\tilde{x}_{t+1} = \bar{T}(x_t) + h(x_t)$. Combining this result with the upper estimation and Theorem 4.6, we get local exponential convergence of the non-autonomous system (4.22), i.e. the ADAM algorithm with bias correction defined as in [KB15]. □

4.7.2 Global Non-Convergence

Analogous to Subsection 4.7.1, we can also transfer the global non-convergence to the ADAM variant of [KB15]. Denote the autonomous system $\bar{T}_{OV}(x_t)$ as described in equation (4.18) and $\tilde{\Theta}(t,x)$ as defined in equation (4.23). Therefore the summation Recap:

$$\bar{T}_{OV}(x_t) = \begin{bmatrix} \beta_1 m_t + (1-\beta_1) g(w_t) \\ \beta_2 v_t + (1-\beta_2) g(w_t)^2 \\ w_t - \alpha \frac{m_{t+1}}{\sqrt{v_{t+1}}+\epsilon} \end{bmatrix}$$

$$x_{t+1} = \bar{T}_{OV}(x_t) + \tilde{\Theta}(t,x)$$

describes the ADAM optimizer of [KB15]. Recall that we divide the proof into four steps:

1. We start with the scalar case $f(w) = \frac{1}{2}cw^2$, $c > 0$. We obtain analytical expressions for 2-limit-cycles $\bar{T}^2(x; \alpha, \beta_1, \beta_2, \varepsilon) = x$ of the autonomous system with $\varepsilon = 0$.
2. Calculating the eigenvalues of $\bar{T}^2$ we find that these do not depend on the learning rate α and the factor c in the objective function. For some typical values of the hyperparameters we give evidence that these limit cycles are often attractive. We have not managed to give analytical estimates for stable eigenvalues using CAS so far.
3. Using the implicit function theorem we show that for a neighborhood of $\varepsilon_\star = 0$ there exists a unique limit cycle of the autonomous system with $\varepsilon > 0$. By continuity of the eigenvalues, these limit cycles are also attractive for ε small enough.
4. We apply a disturbance estimate to show that locally solutions of $T(t, x; p)$ converge to the limit cycles of $\bar{T}(x; p)$. The proof is essentially the same as in [BW19a, Theorem V.1.] and holds for cycles of any integer length.

The only difference between Algorithm 6 and the version of [KB15] is the position of the ϵ. Since we only consider systems with $\epsilon = 0$ in steps one to three, these can be taken exactly from Section 4.5. Therefore we only have to take a closer look at step four. For this we need again Theorem 4.11. Again we have to estimate the difference between two iterations of ADAM with and without bias correction:

$$\left\| T\left(t+1, T\left(t, x\right)\right) - \bar{T}_{OV}\left(\bar{T}_{OV}\left(x\right)\right) \right\|$$

Here we use the estimation from the proof of Corollary 4.7, since the ϵ position does not matter in this estimation.

$$\left\| \tilde{\Theta}\left(t, x\right) \right\| = \left\| T\left(t, x\right) - \bar{T}_{OV}\left(x\right) \right\| \leq C\beta^t \left\| x - x_\star \right\|$$

as well as

$$\left\| \tilde{\Theta}\left(t, x\right) \right\| = \left\| T\left(t, x\right) - \bar{T}_{OV}\left(x\right) \right\| \leq C\beta^t \left\| x \right\|$$

Again we can estimate

$$||T\left(t+1, T\left(t, x\right)\right) - \bar{T}_{OV}\left(\bar{T}_{OV}\left(x\right)\right)|| \leq C\beta^{t+1}||T\left(t, x\right)|| + L||T\left(t, x\right) - \bar{T}_{OV}\left(x\right)||$$

with L as a local Lipschitz constant near the limit cycle. The Lipschitz continuity exists because $\bar{T}_{OV}$ is locally continuously differentiable in an area around $\tilde{v} \neq 0$. With this

knowledge the proof works simultaneously to the proof of Theorem 4.12 and we can summarize these results again in a theorem.

Theorem 4.21. *Consider the ADAM-Optimizer in batch mode as defined in [KB15] and objective function $f(w) = \frac{1}{2}cw^2$, $c > 0$. Then the algorithm is locally convergent under the assumptions stated in Corollary 4.20. However there exist solutions that converge to the 2-limit-cycles of the algorithm without bias correction; so the algorithm does not converge globally to the minimum $w_\star = 0$.*

4.8 Application

We now want to relate the theoretical results from the upper sections to practical applications. For this purpose, we first consider the quantitative effect of inequality (4.3) on an example application. Then, in another example application, we calculate all components of the ADAM individually and can thus classify the eigenvalues.

4.8.1 Neural Networks under Consideration of Inequality (4.3)

Due to its a priori character, the use of inequality (4.3) is limited for real problems. Nevertheless, we can indicate trends for the hyperparameters. In the ADAM optimizer we have four hyperparameters $(\alpha, \beta_1, \beta_2, \epsilon)$ to adjust the learning process. But accordingly to inequality (4.3), only α, β_1 and ϵ influence the convergence behavior. Therefore we only adjust these parameters with respect to inequality (4.3).

$$\frac{\alpha \max\limits_{i=1}^{n}(\mu_i) - 2\epsilon}{\alpha \max\limits_{i=1}^{n}(\mu_i) + 2\epsilon} < \beta_1 \tag{4.24}$$

$$\frac{2\epsilon\,(\beta_1 + 1)}{\max\limits_{i=1}^{n}(\mu_i)\,(1 - \beta_1)} > \alpha \tag{4.25}$$

$$\frac{\alpha \max\limits_{i=1}^{n}(\mu_i)\,(1 - \beta_1)}{2\,(\beta_1 + 1)} < \epsilon \tag{4.26}$$

Since $\max\limits_{i=1}^{n}(\mu_i)$ is quite difficult to calculate, we only make quantitative statements in this subsection. Within the framework of the other restrictions, β_1 and ϵ must be as large as possible and α as small as possible. In the tests we use a standard CNN. However, this is not explained in detail, because in this subsection we only pay attention to the differences in the hyperparameters and not to the general result. The architecture and the data set are therefore randomly chosen and could also be changed. With this knowledge, we test some hyperparameters on the dataset Fashion-MNIST [XRV17].

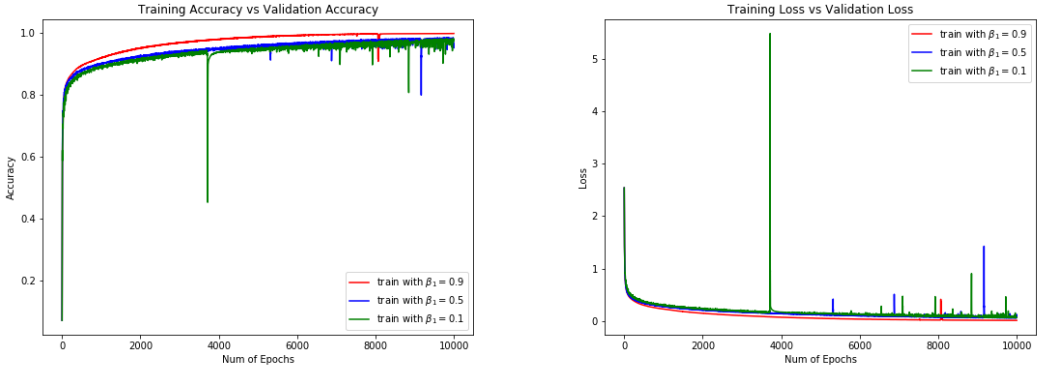


Figure 38: Accuracy and loss diagram with different values for β_1.

By choosing a big β_1, e.g. $\beta_1 = 0.9$ like suggested in [KB15], we can detect a much faster increasing accuracy and decreasing loss. Thus the minimum $x_\star$ is found faster. In addition, the graphs of $\beta_1 = 0.9$ appear smoothed. This is due to the fact that the history m_t is weighted more strongly by a big β_1. In the same manner, the new gradient is weighted weaker than by a small β_1. Therefore, outliers, as near epoch 4000 in Figure 38, are weighted with a large β_1 weaker and thus smoothed more strongly.

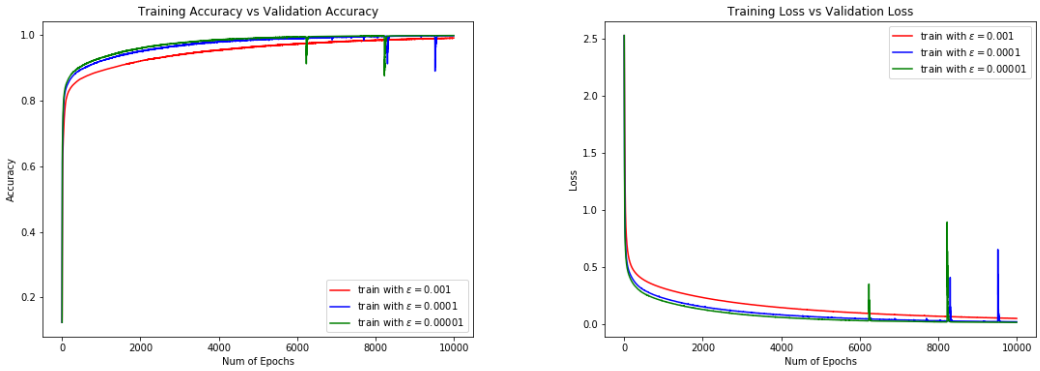


Figure 39: Accuracy and loss diagram with different values for ϵ.

However, if we increase ϵ or decrease α, our experiments show a slower convergence speed (see Figure 40 or 39). This is due to the resulting smaller step sizes in the ADAM optimizer. For $\alpha = 0.01$ there are many outliers in training. This follows from the high weighting of the new step. The tests show that $\alpha = 0.001$ (as suggested by Kingma and Ba [KB15]) gives the best results.

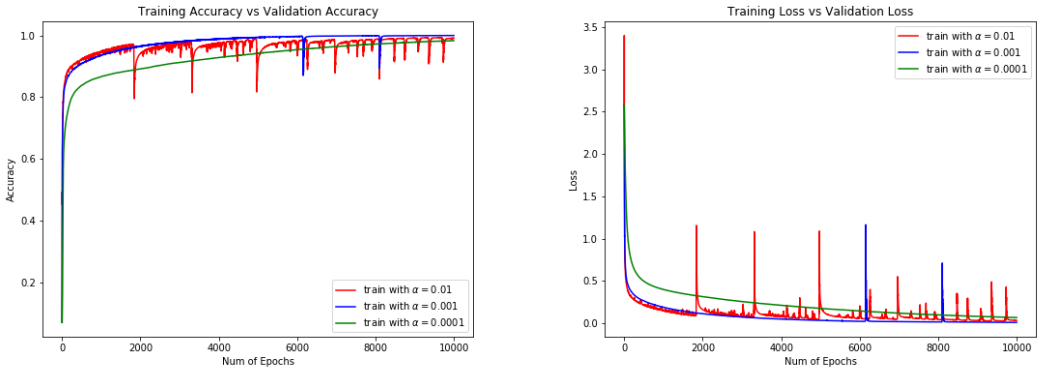


Figure 40: Accuracy and loss diagram with different values for α.

Since besides convergence also convergence speed is an important topic for ADAM users, we recommend a big $\beta_1 < 1$. Kingma and Ba use $\beta_1 = 0.9$ in [KB15] and thus fulfill very good starting possibilities for the inequality (4.3) and thus also for the ADAM optimizer. In summary, it can be said that the hyper parameters were already very well chosen by Kingma and Ba, even in view of the new convergence proof. Therefore, these should preferably be used. Only if the convergence speed is not important, ϵ or α should be changed (ϵ bigger or α smaller).

4.8.2 Eigenvalue Calculation on Neural Networks

Basically the ADAM optimizer is a gradient descent algorithm and thus the main task is to minimize differentiable functions in every area of advanced mathematics. However, the optimizer is mainly known in the field of neural networks. Thus we want to confirm the convergence statement with an example in the field of neural networks. Therefore we use the Modified NIST (MNIST) database used in [LeC+89]. Here we have to identify the handwritten numbers from zero to nine. To simplify the calculation of the eigenvalues, we use such a simple data set. For the same reason, we also use a simple two-layer architecture. Since we work in complete batch mode, we shrink the images to 14×14 pixels in order to relieve the RAM of our train computer.

To reduce complexity, we use a fully-connected neural network without any hidden layer. We have $196 = 14 * 14$ input neurons and ten output neurons for each possible number. These two layers are fully connected and so we reach 1960 weights with additionally ten bias weights. Thus we have to train at least 1970 independent train images to reach a minimum without any degrees of freedom. In Figure 41 we can see the structure of the experiment.

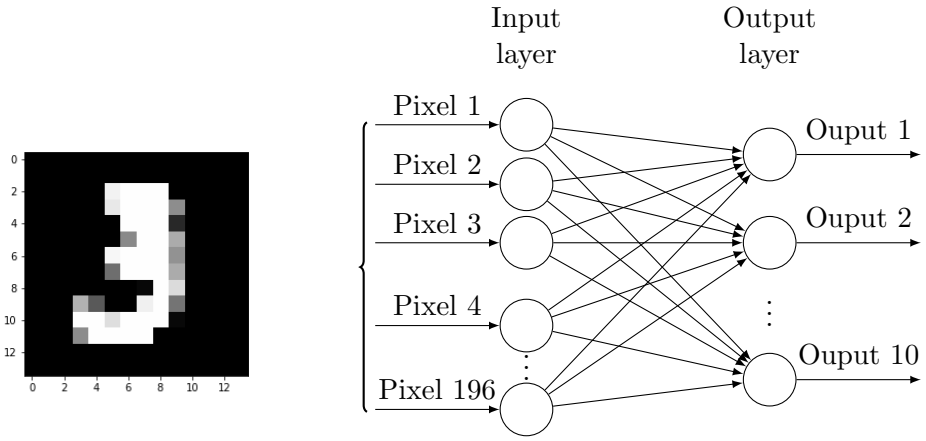


Figure 41: Experiment setup

The numbers in the data set are mostly centered in the image. This results in redundant pixels. See e.g. the pixel in the upper left corner in Figure 41, which is always black or zero no matter which number is shown. These redundant pixels cause a plateau in the loss function and thus the eigenvalue zero in the Hessian. This eigenvalue violates inequality (4.12) thus, our proof of convergence and the convex optimization setting is no longer valid. To avoid this problem, we use the following loss function with regularization term.

$$E = -\sum y_{true} \log\left(y_{pred}\right) + c \sum w^2 \tag{4.27}$$

Indices have been omitted from equation (4.27) for reasons of clarity. The first sum in this loss function the categorical cross entropy, is typically used on the MNIST data set and minimizes errors in wrong classifications. This loss function is standard and is therefore not explained further here. The second sum describes a regularization on the weights of the neural network. Therefore connections to redundant pixels will be minimized to zero and this avoids indefiniteness of the Hessian. More on this topic later. As activation we use the softmax function as common. This allows us to guarantee that the sum of the outputs is equal to one, giving us probabilities for each output.

At this point we would like to mention a few special programming features.[15] Untypical for neural networks, we used a double-precision floating-point format (*float64*). Float32

[15]The code is written in Python and Tensorflow 2.4.1

is usually used in neural networks due to avoiding OOM errors (*Out of memory errors*). Others, such as [He+16], even work with *float16* to reduce training time. This feature is slightly hidden and can be set in the Keras backend with *tf.keras.backend.set_floatx('float64')*. Experiments in *float32* often showed small rounding errors. This was reflected in eigenvalues alternating around one or eigenvalues with the exact value one. Especially problematic for experiments without regression ($c = 0$ in the loss function (4.27)).
To calculate the Hessian of $f(w)$, we use the function *tf.GradientTape* from Tensorflow. This allows us to save the gradients in form of a tensor. Unfortunately *tf.GradientTape* does not support calculating the Hessian in eager execution mode. Thus we calculate the Hessian with the Jacobian from the Gradient. But to reduce the train time, we only calculate the Hessian in the last step.
With the function *optimizer.get_slot*, it is possible to receive the internal parameters m_t and v_t from the ADAM optimizer. With these parameters we calculate $J_{\bar{T}}(m_t, v_t, w_t)$ (see equation (4.10)) and $J_{\bar{T}}(0, 0, w_\star)$ (see equation (4.11)), which should converge against each other for a vanishing gradient.

To avoid scenarios with indefinite Hessians, we use a regularization as described in equation (4.27) with $c = 10^{-8}$. So even if we have redundant pixels, the corresponding weights have a strict minimum in $w_{i,j} = 0$. With hyper parameters $\alpha = 10^{-5}, \beta_1 = 0.9, \beta_2 = 0.8, \epsilon = 10^{-2}$ and a training set of size 4000 we reach a train accuracy of 1.0, a test accuracy of 0.8278 and a gradient in the euclidean norm smaller than 10^{-3}. The generalization error is still high due to no dropout layer or random picking inputs like in SGD. The euclidean norm of the gradient also acts as an abort criterion and we not train a fixed number of epochs (to reach this abort criterion we train approximately 10^6 epochs). The corresponding maximal eigenvalue of the Jacobian is

$$\max_{i=1}^{3n}(|\lambda_i|) = 0.9999999200024413$$

With eigenvalues of Theorem (4.4) and the minimal[16] eigenvalue (of the Hessian) $\mu = 7.999 \cdot 10^{-5}$, the theoretically maximum eigenvalue is

$$\max_{i=1}^{3n}(|\lambda_i|) = 0.999999920000185$$

and meets the calculated eigenvalue very well with a difference of $2 \cdot 10^{-12}$. Also the

[16]Note that equation (4.3) bound the maximal eigenvalue of the Hessian, but both the minimum and the maximum of the Hessian eigenvalue can maximize the eigenvalue of the Jacobian.

eigenvalue $\lambda = \beta_2$ could be found with a small tolerance of $\pm 10^{-6}$ 1970 times. This is exactly one third of the maximum rank of the Jacobian and thus agrees with the theory. Thus, both the maximum eigenvalue and the 1970 eigenvalues $\lambda = \beta_2$ provide empirical support for the theoretical convergence analysis in Section 4.4.

4.8.2.1 Experiments without Regularization

In early experiments we use the loss function (4.27) with $c = 0$ so without regularization. However, this always led to eigenvalues slightly higher than one. This phenomenon could be attributed to redundant pixels in the image. For example, each image in the MNIST dataset has a black pixel in the upper left corner (see Figure 41). I.e. no matter which number is shown, this pixel carries the same information and is therefore redundant for decision finding. If we think in the context of neural networks, this means that for some pixels we cannot find an strict minimum in the categorical crossentropy and we get eigenvalues equal to zero in the Hessian as described above.

These eigenvalues equal to zero arise from the fact that there are infinitely many pairs of values $w_{i,j}$ and Θ_j, which set redundant pixels to zero (see Figure 43). Thus, the redundant pixels do not necessarily have to have a grey value of zero.

Lemma 4.22. *Let $i_s \in \mathbb{N}$ be an arbitrary but fixed value and a redundant pixel for the decision finding of a neural network with an architecture as in Figure 41, then exist infinitely many pairs of $w_{s,j}$ and Θ_j to minimize a loss function without regularization or any other comparable method. Thus no strict minimum can be found.*

Proof. For a proof see the sum in Figure 43 before the softmax function. Let $s \in \mathbb{N}^+$ be an index of a redundant pixel. Then we can rewrite it to

$$\sum_{k=1}^{196} w_{k,j} i_k + \Theta_j = w_{s,j} i_s + \Theta_s + \sum_{\substack{k=1 \\ k \neq s}}^{196} w_{k,j} i_k + \tilde{\Theta}_j$$

and see easily that $w_{s,j} i_s + \Theta_s$ has infinite solutions to be equal zero, since only i_s is fixed. □

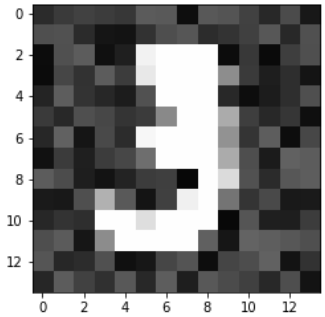


Figure 42: MNIST image with noisy redundant pixel

Even if we use images with randomly chosen grey values for each grey value equal to zero (these are the grey values which are mostly redundant), we can identify a similar behavior (for an example see Figure 42). Here we assume that sometimes the redundant and random pixel is helpful for the classification and sometimes detrimental for it. This results in $w_{s,j}i_s + \Theta_s = c$ in the minimum with an constant $c \in \mathbb{R}$ not necessarily equal to zero. Thus there are again infinitely many solutions $w_{s,j}$ and Θ_s to fulfill the equation.

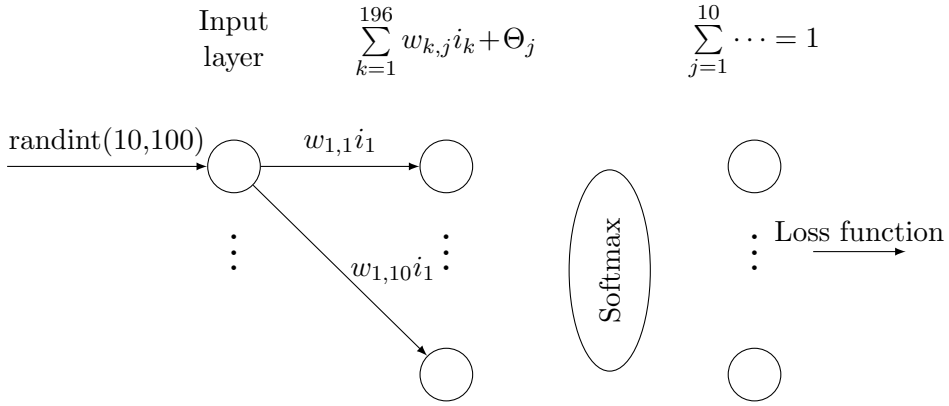
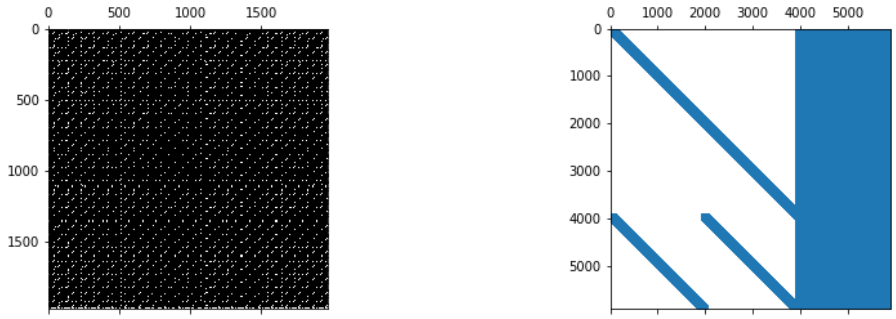


Figure 43: Extract from the network architecture used

Experiments with random grey values for redundant pixels confirm the theoretical considerations. For this we chose random grey values at each pixel equal to zero and for each image. The new grey values were determined using the discrete uniform distribution between 10 and 100.[17] By randomly changing pixels with grey value zero, we also change non-redundant pixels. However, this is equivalent to a noisy image and therefore should not seriously change the eigenvalues of the Hessian matrix. In the experiment, we

[17] In Numpy this is the function *randint(10,100)*. Hence the abbreviation in Figure 43.

stopped after six million epochs because hardly any changes were left and the difference between the losses of two consecutive epochs was approximately 10^{-16}. In addition, the network was error-free with 4000 training images and a training accuracy of 1.0. The test accuracy is 0.8123 and thus indicates an overfit network. As soon as one randomly re-transformed the training data, the loss exploded. This also speaks for an overfitting. If we look at this with $w_{s,j}i_s + \Theta_s = c$, the new noise changes c. Thus, there are new pairs $w_{s,j}$ and Θ_s, which must first be trained again. Nevertheless, there are infinitely many pairs which minimize the loss function for a redundant pixel. This can also be seen in the Hessian and the Jacobian (see Figure 44).



(a) Hessian Matrix
(spy function with precision 10^{-5})

(b) Jacobian Matrix
(spy function with precision 10^{-10})

Figure 44: Hessian and Jacobian for noisy images and $c = 0$

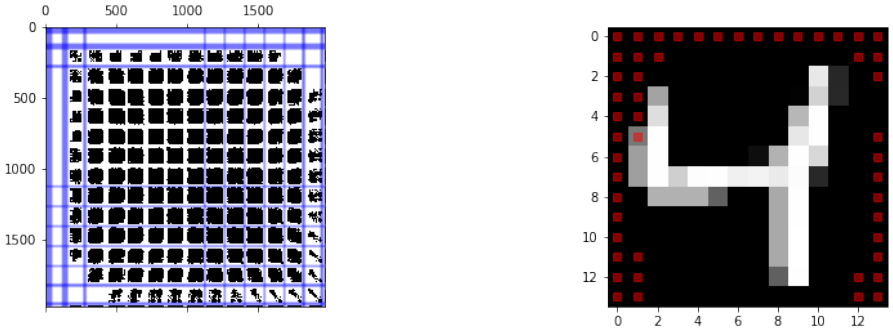
In principle, the spy function[18] of the Hessian matrix looks good, since there are no large or systematic white areas. However, the eigenvalues of the Hessian matrix are in the minimum negative and complex $-9.33 \cdot 10^{-11} - i1.54 \cdot 10^{-26}$ and in the maximum positive 3.14. Recall that we observe the eigenvalues of an symmetric matrix, so theoretically we only get real eigenvalues. However, since the complex fraction is in the range of 10^{-26}, calculation inaccuracy is assumed here. Nevertheless, the Hessian matrix is indefinite and we cannot make any conclusions about extrema. Also for the gradient, no value in absolute value is smaller than 10^{-6}. The fact that the gradient is not completely equal to zero also results in the Jacobian matrix with the derivatives $\frac{\partial v_{t+1}}{\partial w_t}$ and $\frac{\partial w_{t+1}}{\partial v_t}$ not equal to zero. The calculated eigenvalue of the Jacobian matrix is $1.0 + 2.431 \cdot 10^{-13}$. Thus, nothing can be said about the convergence of the algorithm, which is, however, logical due to the above considerations.

[18] Spy function calculates $|a| < \textit{precision}$ for each value a in a given matrix. Then, each value that fulfills the function is drawn white and all others are drawn black.

A missing strict minimum, however, should be considered when selecting the loss function. Especially when using momentum methods, a progress could be faked, which is only based on old momentum. In addition, regularization promotes generalization[19] of the network [KH92]. In a nutshell, we can conclude:

Any model that is trained with partially redundant inputs should have regularization in the loss function so that a strict minimum is possible at all.

Based on this knowledge, we were also able to turn the tables and extract redundant pixels using the Hessian matrix. For this we used the original images again (without any artificial noise) and focused first on the pixels that are zero in each image. These pixels were located exclusively in the corner and edge areas of the images.



(a) Hessian with always zero pixel marked blue (b) All redundant pixel with sample image

Figure 45: Redundant pixel observation

We now consider all the connections of these pixels. Recall that we are using a neural network without any hidden layer (see Figure 41), so each pixel has ten connections to the output layer. These ten connections to each pixel with always grey value zero where located and visualized in blue in Figure 45a. Consider that Tensorflow works with "channels last". Thus, each image is vectorized row wise in the Flatten layer. In black we can observe a spy function of the Hessian matrix. This spy function is without precision so each value unequal to zero is drawn in black. We can identify that each blue area is over a white spot of the Hessian. This is conform with the theory, because by these zero lines the rank of the Hessian matrix is reduced and the eigenvalues equal zero arise. This is what we conclude earlier for redundant pixel. Thus, we can confirm the theory experimentally at this point. However, we also notice that there are other

[19] "generalization bound is the expected difference between the error a model incurs on a training set versus the error incurred on a new data point, sampled from the same distribution that generated the training data" [HRS16]

white spots in the Hessian matrix. So there are also redundant pixels, which do not always have grey value zero. Therefore, we searched for all pixels where all ten associated connections could be traced back to a zero row in the Hessian matrix. Recall that it is enough to search for a zero row, because of the symmetric of the Hessian. The columns also drawn in Figure 45a are only for aesthetics and do not contain any other information. Using this method, we were able to find 50 pixels, all of which are irrelevant to the neural network (see the red squares in Figure 45b). In Figure 45b we also see that these redundant pixel are not always black (see the pixel with coordinates $(1, 5)$). In addition, it should be noted that we only marked pixels where each connection was due to a zero row in the Hessian matrix. So it may well be that there are additional pixels which are redundant for certain outputs.

With the analysis of the Hessian matrix, the complexity of a neural network could be reduced. For an optimization directly in the training process, however, the calculation of the Hessian matrix will be too computationally expensive. We could also detect the redundant pixels in the eigenvectors of the Hessian matrix (see Figure 46). However, only the first 170 connections had an exact unit vector as eigenvector (see Figure 46 upper left corner). The other redundant pixels can only be guessed by the zero lines that appear. This unsatisfactory statement about the redundant pixels and the increased computational effort due to the eigenvectors, led us to not investigate the eigenvectors further.

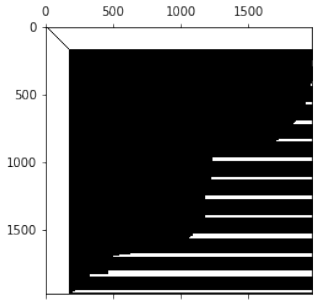


Figure 46: Eigenvectors of the Hessian matrix

4.9 ADAM Optimizer in incremental mode[20]

In contrast to Section 4.4 we are also able to prove the convergence of the ADAM Optimizer in incremental mode. Therefore we had to reformulate the ADAM as in Algorithm 8 (The modifications to Algorithm 6 are marked in green). Note that in contrast to Section 4.4 we work with a decreasing step size α_t and a decreasing momentum in m_t (see variable $\beta_{1,t}$ in the definition of m_t). Therefore the function f was generalized to f_t and so the ADAM is in incremental mode. In addition, we also include the technical parameter ϵ, which was neglected in [BWG18] and [AT21].

[20]This chapter expands the results and presents proofs that are referenced in [BWG18].

Algorithm 8 Incremental ADAM Optimizer [KB15]

Require: $\alpha_t = \frac{\alpha}{\sqrt{t}}$, with $\alpha \in \mathbb{R}^+$, $g_{t+1} = \nabla_w f_{t+1}(w_t)$, $\epsilon \in \mathbb{R}^+, \beta_1, \beta_2 \in (0,1)$, $\beta_{1,t} := \beta_1 \lambda^t, \lambda \in (0,1)$, $w_0 \in \mathbb{R}^n$ and the function $f(w) \in C^2(\mathbb{R}^n, \mathbb{R})$

1: $m_0 = 0, v_0 = 0, t = 0$
2: **while** w_t not converged **do**
3: $m_{t+1} = \beta_{1,t} m_t + (1 - \beta_1) g_{t+1}$
4: $v_{t+1} = \beta_2 v_t + (1 - \beta_2) g_{t+1}^2$
5: $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$
6: $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$
7: $w_{t+1} = w_t - \alpha_t \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon}$
8: $t = t + 1$
9: **end while**

In [BWG18] we used a conjecture for which Akrout in [AT21] found a counterexample, with the atypically small variables $\beta_1 = 0.1 = \beta_2$. We were able to confirm this counterexample. However, the example for β near one i.e. $\beta_1 = 0.9$ and $\beta_2 = 0.999$ fulfills the assumption from [BWG18]. This is another indication for the very good selection of the standard parameters in [KB15].

Akrout in [AT21] improved and proved this conjecture, which we reproduce in the following theorem with our notation.

Theorem 4.23. *Let* $\beta_2 < 2\beta_1^2, \beta_2 \geq 2\beta_1 - \beta_1^2$ *and* $\tau = \lfloor \frac{-\log(2)}{\log(\beta_1)} \rfloor$ *hold, then follows* $\forall T \in [1, \infty)$

$$\sum_{t=1}^{T} \frac{\hat{m}_t^2}{\sqrt{t\hat{v}_t}} < \left(2 + \sqrt{\tau}\right) \sqrt{1 + \frac{\beta_2 (1 - \beta_1)^2}{\left(\beta_2 - \beta_1^2\right)(1 - \beta_2)} \log(T)} ||g_{1:T}||_2$$

Proof. See [AT21]. □

With this theorem, we can complete the convergence proof of the ADAM in incremental mode. Therefore we have to define the error sum first.

Definition 4.24. ***(Error sum)***
Let $w_\star := \arg\min_{w \in \chi} \sum_{t=1}^{T} f_t(w)$ *with* χ *as the set of* w*, which will arise in the ADAM Optimizer. The error sum is then defined as:*

$$R(T) := \sum_{t=1}^{T} \left(f_{t+1}(w_t) - f_{t+1}(w_\star)\right)$$

In the next step we will show that $R(T)$ has an upper bound. If this converges to zero as T increases, the algorithm converges.

Theorem 4.25. *Let $\nabla_w f_t(w_{t-1}) = g_t$ be bounded with $||g_t||_2 \leq G$ and $||g_t||_\infty \leq G_\infty, \forall t \in \{1, ..., T\}$. Furthermore, suppose that the difference between w_t in each time step is bounded by $||w_n - w_m||_2 \leq D$ and $||w_n - w_m||_\infty \leq D_\infty$ with $n, m \in \{1, ..., T\}$. Also let $\beta_1, \beta_2 \in (0,1), \beta_2 < 2\beta_1^2, \beta_2 \geq 2\beta_1 - \beta_1^2, \tau := \lfloor \frac{-\log(2)}{\log(\beta_1)} \rfloor$ and $\lambda \in (0,1)$ hold. Then the regret bound of the ADAM Optimizer can be estimated with:*

$$\begin{aligned} R(T) \leq & \frac{D_\infty^2}{2\alpha\left(1-\beta_1\right)} \sum_{i=1}^{d} \left(\sqrt{T\hat{v}_{T,i}} + \epsilon \right) + \frac{dD_\infty^2 G_\infty}{2\alpha\left(1-\beta_1\right)\left(1-\lambda\right)^2} \\ & + \frac{\alpha\left(1+\beta_1\right)\left(2+\sqrt{\tau}\right)\sqrt{1 + \frac{\beta_2(1-\beta_1)^2}{(\beta_2-\beta_1^2)(1-\beta_2)} \log\left(T\right)}}{2\left(1-\beta_1\right)} \sum_{i=1}^{d} ||g_{1:T,i}||_2 \end{aligned}$$

Proof. We consider the difference between the current weights w_{t+1} and the solution $w_\star$. Therefore we focus on the ith component of $w_t \in \mathbb{R}^d$.

$$\begin{aligned} w_{t+1,i} - w_{\star,i} =& w_{t,i} - w_{\star,i} - \alpha_t \frac{\hat{m}_{t+1,i}}{\sqrt{\hat{v}_{t+1,i}} + \epsilon} \\ \left(w_{t+1,i} - w_{\star,i}\right)^2 =& \left(w_{t,i} - w_{\star,i}\right)^2 - \frac{2\alpha_t \hat{m}_{t+1,i}}{\sqrt{\hat{v}_{t+1,i}} + \epsilon} \left(w_{t,i} - w_{\star,i}\right) + \alpha_t^2 \left(\frac{\hat{m}_{t+1,i}}{\sqrt{\hat{v}_{t+1,i}} + \epsilon} \right)^2 \\ \frac{2\alpha_t \hat{m}_{t+1,i}}{\sqrt{\hat{v}_{t+1,i}} + \epsilon} \left(w_{t,i} - w_{\star,i}\right) =& \left(w_{t,i} - w_{\star,i}\right)^2 - \left(w_{t+1,i} - w_{\star,i}\right)^2 + \alpha_t^2 \left(\frac{\hat{m}_{t+1,i}}{\sqrt{\hat{v}_{t+1,i}} + \epsilon} \right)^2 \\ \frac{1-\beta_1}{\sqrt{\hat{v}_{t+1,i}} + \epsilon} g_{t+1,i} \left(w_{t,i} - w_{\star,i}\right) =& \frac{1-\beta_1^t}{2\alpha_t} \left(\left(w_{t,i} - w_{\star,i}\right)^2 - \left(w_{t+1,i} - w_{\star,i}\right)^2 \right) \\ & + \frac{\alpha_t \left(1-\beta_1^t\right)}{2} \left(\frac{\hat{m}_{t+1,i}}{\sqrt{\hat{v}_{t+1,i}} + \epsilon} \right)^2 \\ & - \frac{\beta_{1,t}}{\sqrt{\hat{v}_{t+1,i}} + \epsilon} m_{t,i} \left(w_{t,i} - w_{\star,i}\right) \\ g_{t+1,i} \left(w_{t,i} - w_{\star,i}\right) =& \frac{\left(1-\beta_1^t\right)\left(\sqrt{\hat{v}_{t+1,i}} + \epsilon\right)}{2\alpha_t\left(1-\beta_1\right)} \left(\left(w_{t,i} - w_{\star,i}\right)^2 - \left(w_{t+1,i} - w_{\star,i}\right)^2 \right) \\ & + \frac{\alpha_t \left(1-\beta_1^t\right)\left(\sqrt{\hat{v}_{t+1,i}} + \epsilon\right)}{2\left(1-\beta_1\right)} \left(\frac{\hat{m}_{t+1,i}}{\sqrt{\hat{v}_{t+1,i}} + \epsilon} \right)^2 \\ & + \underbrace{\frac{\beta_{1,t}}{\left(1-\beta_1\right)} m_{t,i} \left(w_{\star,i} - w_{t,i}\right)}_{(*)} \end{aligned}$$

We multiply $(*)$ with $1 = \frac{\hat{v}_{t,i}^{\frac{1}{4}}\sqrt{\alpha_t}}{\hat{v}_{t,i}^{\frac{1}{4}}\sqrt{\alpha_t}}$ and use the binomial equation to simplify.

$$\begin{aligned}
&\frac{\beta_{1,t}}{1-\beta_1} m_{t,i}\left(w_{\star,i}-w_{t,i}\right)\frac{\hat{v}_t^{\frac{1}{4}}\sqrt{\alpha_t}}{\hat{v}_t^{\frac{1}{4}}\sqrt{\alpha_t}} \\
=&\frac{\beta_{1,t}}{1-\beta_1}\left(\frac{\hat{v}_t^{\frac{1}{4}}}{\sqrt{\alpha_t}}\left(w_{\star,i}-w_{t,i}\right)\sqrt{\alpha_t}\frac{m_{t,i}}{\hat{v}_{t,i}^{\frac{1}{4}}}\right) \\
\leq&\frac{\beta_{1,t}}{1-\beta_1}\left(\frac{\sqrt{\hat{v}_{t,i}}\left(w_{\star,i}-w_{t,i}\right)^2}{2\alpha_t}+\frac{\alpha_t m_{t,i}^2}{2\sqrt{\hat{v}_{t,i}}}\right)
\end{aligned}$$

If we put all these together we reach the following inequality. We separate it in five terms and handle each of them on their own. In addition, some terms were estimated using $1-\beta_1 < 1$ and $1-\beta_{1,t} > 1-\beta_1$.

$$\begin{aligned}
\underbrace{g_{t+1,i}\left(w_{t,i}-w_{\star,i}\right)}_{(4.28.1)} \leq& \underbrace{\frac{\left(\sqrt{\hat{v}_{t+1,i}}+\epsilon\right)}{2\alpha_t\left(1-\beta_1\right)}\left(\left(w_{t,i}-w_{\star,i}\right)^2-\left(w_{t+1,i}-w_{\star,i}\right)^2\right)}_{(4.28.2)} \\
&+\underbrace{\frac{\beta_{1,t}}{2\alpha_t\left(1-\beta_1\right)}\left(w_{\star,i}-w_{t,i}\right)^2\sqrt{\hat{v}_{t,i}}}_{(4.28.3)} \\
&+\underbrace{\frac{\beta_1\alpha_t m_{t,i}^2}{2\left(1-\beta_1\right)\sqrt{\hat{v}_{t,i}}}}_{(4.28.4)}+\underbrace{\frac{\alpha_t}{2\left(1-\beta_1\right)}\frac{\hat{m}_{t+1,i}^2}{\sqrt{\hat{v}_{t+1,i}}+\epsilon}}_{(4.28.5)}
\end{aligned} \tag{4.28}$$

In order to continue with Term (4.28.1), we must first be clear that we can write for a convex differentiable function $f_t(w)$:

$$f_{t+1}\left(w_t\right)-f_{t+1}\left(w_\star\right) \leq g_{t+1}^{\perp}\left(w_t-w_\star\right)$$

To get the link to the error sum, we sum over the elements of the gradient $i \in 1,...,d$ and the time steps $t \in 1,...,T$. Then we can estimate term (4.28.1):

$$\sum_{t=1}^{T}\sum_{i=1}^{d} g_{t+1,i}\left(w_{t,i}-w_{\star,i}\right)=\sum_{t=1}^{T} g_{t+1}^{\perp}\left(w_t-w_\star\right) \geq \sum_{t=1}^{T}\left(f_{t+1}\left(w_t\right)-f_{t+1}\left(w_\star\right)\right)=R\left(T\right)$$

Next we look at term (4.28.2).

$$
\begin{aligned}
&\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{\left(\left(w_{t,i}-w_{\star,i}\right)^2-\left(w_{t+1,i}-w_{\star,i}\right)^2\right)}{2\alpha_t\left(1-\beta_1\right)}\left(\sqrt{\hat{v}_{t,i}}+\epsilon\right)\\
=&\sum_{i=1}^{d}\frac{1}{2\alpha_1\left(1-\beta_1\right)}\left(w_{1,i}-w_{\star,i}\right)^2\left(\sqrt{\hat{v}_{1,i}}+\epsilon\right)\\
&+\sum_{i=1}^{d}\sum_{t=2}^{T}\frac{1}{2\alpha_t\left(1-\beta_1\right)}\left(w_{t,i}-w_{\star,i}\right)^2\left(\sqrt{\hat{v}_{t,i}}+\epsilon\right)\\
&-\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{1}{2\alpha_t\left(1-\beta_1\right)}\left(w_{t+1,i}-w_{\star,i}\right)^2\left(\sqrt{\hat{v}_{t,i}}+\epsilon\right)
\end{aligned}
$$

In the last term, we can split off the last summand and change the bounds of the summation to $t=2$ and T.

$$
\begin{aligned}
=&\sum_{i=1}^{d}\frac{1}{2\alpha_1\left(1-\beta_1\right)}\left(w_{1,i}-w_{\star,i}\right)^2\left(\sqrt{\hat{v}_{1,i}}+\epsilon\right)\\
&+\sum_{i=1}^{d}\sum_{t=2}^{T}\frac{1}{2\alpha_t\left(1-\beta_1\right)}\left(w_{t,i}-w_{\star,i}\right)^2\left(\sqrt{\hat{v}_{t,i}}+\epsilon\right)\\
&-\sum_{i=1}^{d}\sum_{t=2}^{T}\frac{1}{2\alpha_{t-1}\left(1-\beta_1\right)}\left(w_{t,i}-w_{\star,i}\right)^2\left(\sqrt{\hat{v}_{t-1}}+\epsilon\right)\\
&-\sum_{i=1}^{d}\frac{1}{2\alpha_T\left(1-\beta_1\right)}\left(w_{T+1,i}-w_{\star,i}\right)^2\left(\sqrt{\hat{v}_{T}}+\epsilon\right)\\
=&\sum_{i=1}^{d}\frac{1}{2\alpha_1\left(1-\beta_1\right)}\underbrace{\left(w_{1,i}-w_{\star,i}\right)^2}_{\leq D_\infty^2}\left(\sqrt{\hat{v}_{1,i}}+\epsilon\right)\\
&+\sum_{i=1}^{d}\sum_{t=2}^{T}\frac{1}{2\left(1-\beta_1\right)}\underbrace{\left(w_{t,i}-w_{\star,i}\right)^2}_{\leq D_\infty^2}\left(\frac{\sqrt{\hat{v}_{t,i}}}{\alpha_t}-\frac{\sqrt{\hat{v}_{t-1,i}}}{\alpha_{t-1}}\right)\\
&\underbrace{-\sum_{i=1}^{d}\frac{1}{2\alpha_T\left(1-\beta_1\right)}\left(w_{T+1,i}-w_{\star,i}\right)^2\left(\sqrt{\hat{v}_{T}}+\epsilon\right)}_{\leq 0}
\end{aligned}
$$

By the fact that the last term is negative and we have limited the distance between the

weight steps by D_∞, we can estimate the equation.

$$\begin{aligned}
&\leq \frac{D_\infty^2}{2\alpha\left(1-\beta_1\right)}\left(\sum_{i=1}^{d}\left(\sqrt{\hat{v}_{1,i}}+\epsilon\right)+\sum_{i=1}^{d}\sum_{t=1}^{T}\left(\sqrt{t\hat{v}_{t,i}}-\sqrt{\left(t-1\right)\hat{v}_{t-1,i}}\right)\right)\\
&= \frac{D_\infty^2}{2\alpha\left(1-\beta_1\right)}\sum_{i=1}^{d}\sqrt{T\hat{v}_{T,i}}+\epsilon
\end{aligned}$$

In the next step we will reformulate term (4.28.3), so that it will fit in our regret bound.

$$\begin{aligned}
&\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{\beta_{1,t}}{2\alpha_t\left(1-\beta_1\right)}\underbrace{\left(w_{\star,i}-w_{t,i}\right)^2}_{\leq D_\infty^2}\sqrt{\hat{v}_{t,i}}\\
\leq &\frac{D_\infty^2}{2\alpha}\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{\beta_{1,t}}{\left(1-\beta_1\right)}\sqrt{t\hat{v}_{t,i}}
\end{aligned}$$

With the definition of $\hat{v}_{t,i}$ only depending on the different gradients at each time step, we can estimate $\hat{v}_{t,i}$.

$$\begin{aligned}
\sqrt{\hat{v}_{t,i}} &= \sqrt{1-\beta_2}\sqrt{\frac{\sum_{j=1}^{t}g_{j,i}^2\beta_2^{t-j}}{1-\beta_2^t}} \leq \sqrt{1-\beta_2}G_\infty\sqrt{\frac{\sum_{j=1}^{t}\beta_2^{t-j}}{1-\beta_2^t}}\\
&\leq \sqrt{1-\beta_2}G_\infty\sqrt{\frac{\sum_{j=1}^{t}\beta_2^{j}}{1-\beta_2^t}} \leq \sqrt{1-\beta_2}G_\infty\sqrt{\frac{1-\beta_2^t}{\left(1-\beta_2^t\right)\left(1-\beta_2\right)}}\\
&\leq G_\infty
\end{aligned}$$

Finally, we need to estimate $\sum_{t=1}^{T}\frac{\beta_{1,t}}{1-\beta_{1,t}}\sqrt{t}$ for the estimation of term (4.28.3).

$$\begin{aligned}
\sum_{t=1}^{T}\frac{\beta_{1,t}}{1-\beta_1}\sqrt{t} &= \sum_{t=1}^{T}\frac{\beta_1\lambda^{t-1}}{1-\beta_1}\sqrt{t} \leq \frac{1}{1-\beta_1}\sum_{t=0}^{T-1}\lambda^t\left(t+1\right)\\
&\leq \frac{\frac{(T-1)\lambda^{T+1}-T\lambda^T+\lambda}{\left(\lambda-1\right)^2}+\frac{1-\lambda^T}{1-\lambda}}{\left(1-\beta_1\right)} = \frac{1-T\left(\lambda^T-\lambda^{T+1}\right)-\lambda T}{\left(1-\beta_1\right)\left(\lambda-1\right)^2}\\
&\leq \frac{1}{\left(1-\beta_1\right)\left(\lambda-1\right)^2}
\end{aligned}$$

Then term (4.28.3) results in:

$$\frac{\beta_{1,t}}{2\alpha_{t-1}\left(1-\beta_1\right)}\left(w_{\star,i}-w_{t,i}\right)^2\sqrt{\hat{v}_{t,i}} \leq \frac{dD_\infty^2 G_\infty}{2\alpha\left(1-\beta_1\right)\left(1-\lambda\right)^2}$$

We estimate term (4.28.4) with Theorem 4.23.

$$\begin{aligned}
&\frac{\beta_1\alpha}{2\left(1-\beta_1\right)}\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{\left(1-\beta_{1,t}\right)^2\hat{m}_{t,i}^2}{\sqrt{t\hat{v}_{t,i}}}\\
\leq&\frac{\beta_1\alpha}{2\left(1-\beta_1\right)}\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{\hat{m}_{t,i}^2}{\sqrt{t\hat{v}_{t,i}}}\\
<&\frac{\beta_1\alpha\left(2+\sqrt{\tau}\right)\sqrt{1+\frac{\beta_2\left(1-\beta_1\right)^2}{\left(\beta_2-\beta_1^2\right)\left(1-\beta_2\right)}\log\left(T\right)}}{2\left(1-\beta_1\right)}\sum_{i=1}^{d}||g_{1:T,i}||_2
\end{aligned}$$

Analogously to term (4.28.4) we handle term (4.28.5).

$$\begin{aligned}
\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{\alpha_t}{2\left(1-\beta_1\right)}\frac{\hat{m}_{t+1,i}^2}{\sqrt{\hat{v}_{t+1,i}}+\epsilon} =&\frac{\alpha}{2\left(1-\beta_1\right)}\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{\hat{m}_{t+1,i}^2}{\sqrt{t\hat{v}_{t+1,i}}+\sqrt{t}\epsilon}\\
\leq&\frac{\alpha}{2\left(1-\beta_1\right)}\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{\hat{m}_{t+1,i}^2}{\sqrt{t\hat{v}_{t+1,i}}}\\
<&\frac{\alpha\left(2+\sqrt{\tau}\right)\sqrt{1+\frac{\beta_2\left(1-\beta_1\right)^2}{\left(\beta_2-\beta_1^2\right)\left(1-\beta_2\right)}\log\left(T\right)}}{2\left(1-\beta_1\right)}\sum_{i=1}^{d}||g_{1:T,i}||_2
\end{aligned}$$

If we combine all these terms, we get our assertion and the proof is finished.

$$\begin{aligned}
R\left(T\right)\leq&\frac{D_\infty^2}{2\alpha\left(1-\beta_1\right)}\sum_{i=1}^{d}\left(\sqrt{T\hat{v}_{T,i}}+\epsilon\right)+\frac{dD_\infty^2 G_\infty}{2\alpha\left(1-\beta_1\right)\left(1-\lambda\right)^2}\\
&+\frac{\alpha\left(1+\beta_1\right)\left(2+\sqrt{\tau}\right)\sqrt{1+\frac{\beta_2\left(1-\beta_1\right)^2}{\left(\beta_2-\beta_1^2\right)\left(1-\beta_2\right)}\log\left(T\right)}}{2\left(1-\beta_1\right)}\sum_{i=1}^{d}||g_{1:T,i}||_2
\end{aligned}$$

□

Using Theorem (4.25) we can prove Corollary 4.26 about the convergence rate.

Corollary 4.26. *Let f_t with $t=1,...,T$ be convex with a bounded gradient $||\nabla f_t\left(w\right)||_2 \leq G$, $||\nabla f_t\left(w\right)||_2 \leq G_\infty, \forall w \in \mathbb{R}^d$. Furthermore, suppose the difference between w_t is bounded by $||w_n - w_m||_2 \leq D$, $||w_n - w_m||_\infty \leq D_\infty, \forall m,n \in 1,...,T$. In addition, all*

prerequisites from Theorem 4.23 shall apply. Then the ADAM Optimizer (see Algorithm 8) has the following convergence rate $\forall T \geq 1$*:*

$$\frac{R(T)}{T} = \mathcal{O}\left(\sqrt{\frac{\log(T)}{T}}\right)$$

Proof. We apply Theorem 4.25 and divide by $T > 0$.

$$\begin{aligned}\frac{R(T)}{T} \leq & \frac{D_\infty^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} \frac{\sqrt{\hat{v}_{T,i}}}{\sqrt{T}} + \frac{d\epsilon D_\infty^2}{2T\alpha} + \frac{dD_\infty^2 G_\infty}{2T\alpha(1-\beta_1)(1-\lambda)^2} \\ & + \frac{\alpha(1+\beta_1)(2+\sqrt{\tau})\sqrt{1+\frac{\beta_2(1-\beta_1)^2}{(\beta_2-\beta_1^2)(1-\beta_2)}\log(T)}}{2T(1-\beta_1)} \sum_{i=1}^{d} ||g_{1:T,i}||_2\end{aligned}$$

Two terms still need to be estimated before we can state the convergence rate. We start with the sum over the norm of the gradients.

$$\begin{aligned}\sum_{i=1}^{d} ||g_{1:t,i}||_2 &= \sum_{i=1}^{d} \sqrt{g_{1,i}^2 + g_{2,i}^2 + ... + g_{T,i}^2} \\ &\leq \sum_{i=1}^{d} \sqrt{G_\infty^2 + G_\infty^2 ... + G_\infty^2} \\ &= \sum_{i=1}^{d} \sqrt{T} G_\infty = dG_\infty\sqrt{T}\end{aligned}$$

Finally, we use the estimation of $\sqrt{v_{t,i}}$ from the proof of term (4.28.3).

$$\sum_{i=1}^{d} \sqrt{\frac{\hat{v}_{T,i}}{T}} \leq \sum_{i=1}^{d} \frac{G_\infty}{\sqrt{T}} \leq \frac{dG_\infty}{\sqrt{T}}$$

With this we can estimate $\frac{R(T)}{T}$ and show that it converges to zero.

$$\lim_{T\to\infty} \frac{R(T)}{T} \leq \lim_{T\to\infty} \left(\frac{1}{\sqrt{T}} + \frac{1}{T} + \frac{1}{T} + \sqrt{\frac{\log(T)}{T}}\right) = 0$$

This proves the convergence speed of $\mathcal{O}\left(\sqrt{\frac{\log(T)}{T}}\right)$ of the ADAM Optimizer. □

With Corollary 4.26 we can prove the convergence of the incremental ADAM. However, this is only possible under three restrictions, which are difficult to check in practice

or are not implemented.

- *Decreasing step size α_t:* Without $\frac{1}{\sqrt{t}}$, the estimation of terms (4.28.4) and (4.28.5) is not possible. An estimation of $\frac{m_t}{\sqrt{v_t}}$ would be desirable here and would make this condition unnecessary.

- *Decreasing momentum in m_t with $\beta_{1,t}$:* $\beta_{1,t}$ is only needed in the estimation of (4.28.3). Here we obtain a $\sqrt{t}$ in the denominator through $\frac{1}{\alpha_t}$. The sum would result in an estimation with $T\sqrt{T}$ for $\lambda = 1$. This would not converge to zero. So if we could do without $\frac{\alpha}{\sqrt{t}}$, a generalization of $\lambda \in (0, 1]$ would be possible.

- *Bounded gradients and weights with D_∞/D_2 and G_∞/G_2:* The main drawback in the present proof is the condition of bounded gradients and weights. This prerequisite cannot be checked in advance in practice, but is nevertheless essential in the estimates of terms (4.28.2)-(4.28.5). Reddi et. al. [RKK19] have already shown that there are unbounded functions for which ADAM does not converge. In addition, they would be violating $\beta_2 < 2\beta_1^2$ with $\beta_1 = 0 \neq \beta_2$.

5 Convergence of other Adaptive Gradient Descent Methods

In the category of adaptive gradient descent algorithms are several popular ones, where ADAM is only one of them. Some of them can also be seen as momentum methods, and they are all available in a stochastic or mini-batch way. We will also apply our convergence proofs to some of these algorithms in the complete batch mode. This shows how general the methodology of each proof is and shows that other algorithms can also benefit from it. In selecting the algorithms, we have focused exclusively on those that enjoy great popularity in science and application (see SGD, AdaGrad, ADAM, AMSGrad named in [RKK19], SGD, AdaGrad, AdaDelta, RMSProp, ADAM, AdaMax, Nadam named in [Rud16] and SGD, AdaGrad, RMSProp, ADAM named in [KB15]). Only the extensions to ADAM have been omitted in this chapter (i.e. Nadam with Nesterov Momentum from [Doz16], AdaMax from [KB15] or AdamW with a decoupled weight decay from [LH19]).

To keep this work clear, we note these algorithms in the elegant way from [RKK19]. We assume the Generic Adaptive Method Setup from Algorithm 9 and set only v_t and m_t different for each optimizer (see Table 13). Note that we implemented a function $d(m_{t+1})$, which is only interesting for AdaDelta. This is necessary due to the use of m_t in the weight update of w_{t+1}. It will become clear that the essential requirement of our method are stable eigenvalues, which are present in most, but not all, algorithms. Furthermore, ADAM is specified as a version without bias-correction and all algorithms are without minibatch version. That means SGD is to be understood without stochastic.

Algorithm 9 Generic Adaptive Method Setup

Require: $\alpha \in \mathbb{R}^+$, $\epsilon \in \mathbb{R}$, $w_0 \in \mathbb{R}^n$ and the function $f(w) \in C^2(\mathbb{R}^n, \mathbb{R})$

1: $m_0 = 0$, $v_0 = 0$, $t = 0$
2: **while** w not converged **do**
3: $\quad v_{t+1} = \psi(v_t, w_t)$
4: $\quad m_{t+1} = \varphi(m_t, w_t)$
5: $\quad w_{t+1} = w_t - \alpha \frac{d(m_t, w_t)}{\sqrt{v_{t+1} + \epsilon^2}}$
6: $\quad t = t + 1$
7: **end while**

Table 13: Popular adaptive gradient descent optimizers

Algorithm	$\varphi(m_t, w_t)$	$\psi(v_t, w_t)$	Momentum	$\mathrm{d}(m_t, w_t)$
SGD [RM51]	$g(w_t)$	$1-\epsilon^2$	$\beta = 1$	$\varphi(m_t, w_t)$
RMSProp [Hin+12b]	$g(w_t)$	$\beta v_t + (1-\beta) g(w_t)^2$	$0 < \beta < 1$	$\varphi(m_t, w_t)$
AdaGrad [DHS11]	$g(w_t)$	$v_t + g(w_t)^2$	$\beta = 1$	$\varphi(m_t, w_t)$
AdaDelta [Zei12]	$\beta m_t + (1-\beta) \cdot g(w_t)^2 \frac{m_t+\epsilon^2}{v_{t+1}+\epsilon^2}$	$\beta v_t + (1-\beta) g(w_t)^2$	$0 < \beta < 1$	$g(w_t)\sqrt{m_t+\epsilon^2}$
ADAM [KB15]	$\beta_1 m_t + (1-\beta_1) g(w_t)$	$\beta_2 v_t + (1-\beta_2) g(w_t)^2$	$0 < \beta_1, \beta_2 < 1$	$\varphi(m_t, w_t)$

5.1 Local Convergence [21]

The convergence proof of the SGD is left to the reader due to its simplicity. The result can be found in Table 14.

5.1.1 RMSProp

First we observe the RMSProp algorithm by Hinton et. al. [Hin+12b]. We can define it with the following system

$$x_{t+1} = T(x_t) = \begin{bmatrix} \beta v_t + (1-\beta) g(w_t)^2 \\ w_t - \alpha \frac{g(w_t)}{\sqrt{v_{t+1}+\epsilon^2}} \end{bmatrix} \tag{5.1}$$

Herewith we can calculate the Jacobian analogous to the convergence proof of the ADAM Optimizer.

$$J_{\bar{T}}(v_t, w_t) = \begin{bmatrix} \beta_2 I & \frac{\partial v_{t+1}}{\partial w_t} \\ \frac{\partial w_{t+1}}{\partial v_t} & \frac{\partial w_{t+1}}{\partial w_t} \end{bmatrix}$$

[21] This chapter expands the results and presents proofs that are referenced in [BW21].

with

$$\begin{aligned}
\frac{\partial v_{t+1}}{\partial w_t} =& 2\left(1-\beta\right)\operatorname{diag}(g\left(w_t\right))\nabla_w g\left(w_t\right) \\
\frac{\partial w_{t+1}}{\partial v_t} =& \frac{\alpha\beta}{2}\operatorname{diag}\left(\frac{g\left(w_t\right)}{\left(v_{t+1}+\epsilon^2\right)^{\frac{3}{2}}}\right) \\
\frac{\partial w_{t+1}}{\partial w_t} =& I-\alpha\left(\operatorname{diag}(v_{t+1}+\epsilon^2)^{-\frac{1}{2}}-\operatorname{diag}\left(g\left(w_t\right)\left(v_{t+1}+\epsilon^2\right)^{-\frac{3}{2}}g\left(w_t\right)\right)\right)\nabla_w g\left(w_t\right)
\end{aligned}$$

This is simplified in the minimum $w_\star$ to

$$J_T\left(0,w_\star\right)=\begin{bmatrix}\beta I & 0\\ 0 & I-\frac{\alpha}{\epsilon}\nabla_w g\left(w_\star\right)\end{bmatrix}$$

As you can see we reach n times the eigenvalue $\lambda_1=\beta$, which is per definition smaller than 1. The submatrix $I-\frac{\alpha}{\epsilon}\nabla_w g\left(w_\star\right)$ is again symmetric since $\nabla_w g\left(w_\star\right)$ is the Hessian of f. Therefore we can diagonalize the matrix and get n times the eigenvalue $\lambda_2=1-\frac{\alpha}{\epsilon}\mu_i$. Next step is to analyze the absolute value of the eigenvalues.

$$\begin{aligned}
& & |\lambda_2| = |1-\frac{\alpha}{\epsilon}\mu_i| < 1 \\
\Leftrightarrow & & -2 < -\frac{\alpha}{\epsilon}\mu_i < 0 \\
\Leftrightarrow & & 0 < \mu_i < \frac{2\epsilon}{\alpha}
\end{aligned}$$

Due to the fact that we look at eigenvalues of a positive definite matrix, $0<\mu_i$ is clear. So for local convergence with exponential rate in the RMSProp algorithm we only have to fulfill:

$$\max_{i=1}^{n}(\mu_i) < \frac{2\epsilon}{\alpha}$$

5.1.2 AdaGrad

For AdaGrad, a proof in this form is not possible. The reason for this is $\beta=1$. With this missing parameter, the following Jacobian is created.

$$J_T\left(v_t,w_t\right)=\begin{bmatrix}I & 2g(w_t)\nabla_w g(w_t)\\ \frac{1}{2}\operatorname{diag}\left(\frac{g(w_t)}{(v_{t+1}+\epsilon^2)^{\frac{3}{2}}}\right) & I-\frac{\alpha}{\epsilon}\nabla_w g\left(w_t\right)\end{bmatrix}$$

In the minimum $(0, w_\star)$

$$J_T(0, w_\star) = \begin{bmatrix} I & 0 \\ 0 & I - \frac{\alpha}{\epsilon} \nabla_w g(w_\star) \end{bmatrix}$$

we see that $\lambda_1 = 1$ no matter which $\max_{i=1}^{n}(\mu_i)$ we calculate. Thus, no convergence statement is possible using our approach.

5.1.3 AdaDelta

Finally, we look at the AdaDelta algorithm from [Zei12]. This algorithm is somewhat more difficult to bring into the system structure. But with different time steps t and $t+1$ for v and m we can write:

$$x_{t+1} = T(x_t) = \begin{bmatrix} \beta v_t + (1-\beta) g(w_t)^2 \\ \beta m_t + (1-\beta) g(w_t)^2 \frac{m_t + \epsilon^2}{v_{t+1} + \epsilon^2} \\ w_t - \alpha \frac{\sqrt{m_t + \epsilon^2}}{\sqrt{v_{t+1} + \epsilon^2}} g(w_t) \end{bmatrix} \tag{5.2}$$

Note that we add the learning rate α to the optimizer different to the original paper. If we set $\alpha = 1$ we reach the original formulation from Zeiler [Zei12]. We will discuss some different learning rates in the experiments. For $T(x_t)$ we get the Jacobian

$$J_T(v_t, m_t, w_t) = \begin{bmatrix} \beta I & 0 & \frac{\partial v_{t+1}}{\partial w_t} \\ \frac{\partial m_{t+1}}{\partial v_t} & \frac{\partial m_{t+1}}{\partial m_t} & \frac{\partial m_{t+1}}{\partial w_t} \\ \frac{\partial w_{t+1}}{\partial v_t} & \frac{\partial w_{t+1}}{\partial m_t} & \frac{\partial w_{t+1}}{\partial w_t} \end{bmatrix}$$

with

$$
\begin{aligned}
\frac{\partial m_{t+1}}{\partial v_t} =& -\beta\left(1-\beta\right)\operatorname{diag}\left(\frac{g\left(w_t\right)^2\left(m_t+\epsilon^2\right)}{\left(v_{t+1}+\epsilon^2\right)^2}\right)\\
\frac{\partial m_{t+1}}{\partial m_t} =& \beta I+\left(1-\beta\right)\operatorname{diag}\left(\frac{g\left(w_t\right)^2}{v_{t+1}+\epsilon^2}\right)\\
\frac{\partial m_{t+1}}{\partial w_t} =& 2\left(1-\beta\right)\operatorname{diag}\left(\frac{\left(m_t+\epsilon^2\right)\left(\beta v_t+\epsilon^2\right)g\left(w_t\right)}{\left(v_{t+1}+\epsilon^2\right)^2}\right)\nabla g\left(w_t\right)\\
\frac{\partial v_{t+1}}{\partial w_t} =& 2\left(1-\beta\right)\operatorname{diag}(g\left(w_t\right))\nabla_w g\left(w_t\right)\\
\frac{\partial w_{t+1}}{\partial v_t} =& \frac{\beta\alpha}{2}\operatorname{diag}\left(\frac{\sqrt{m_t+\epsilon^2}g\left(w_t\right)}{\left(v_{t+1}+\epsilon^2\right)^{\frac{3}{2}}}\right)\\
\frac{\partial w_{t+1}}{\partial m_t} =& -\frac{\alpha}{2}\operatorname{diag}\left(\frac{g\left(w_t\right)}{\sqrt{v_{t+1}+\epsilon^2}}\frac{1}{\sqrt{m_t+\epsilon^2}}\right)\\
\frac{\partial w_{t+1}}{\partial w_t} =& I-\alpha\operatorname{diag}\left(\frac{\sqrt{m_t+\epsilon^2}}{\sqrt{v_{t+1}+\epsilon^2}}-\frac{\sqrt{m_t+\epsilon^2}\left(1-\beta\right)g\left(w_t\right)^2}{\left(v_{t+1}+\epsilon^2\right)^{\frac{3}{2}}}\right)\nabla_w g\left(w_t\right)
\end{aligned}
$$

Fortunately, by inserting $x_\star$, this Jacobian is greatly simplified to

$$
J_T\left(0,0,w_\star\right)=\begin{bmatrix}\beta I & 0 & 0\\ 0 & \beta I & 0\\ 0 & 0 & I-\alpha\nabla_w g\left(w_\star\right)\end{bmatrix}
$$

We can easily identify $2n$ times the eigenvalue $\lambda_1=\beta$ of the Jacobian. We diagonalize the Hessian of f and identify $\lambda_{2,i}=1-\alpha\mu_i$. By observing the spectral radius of the Jacobian we see that $\lambda_1=\beta$ is by definition between zero and one. For λ_2 we generate the following general inequality.

$$
0<\max_{i=1}^{n}\left(\mu_i\right)<\frac{2}{\alpha}
$$

Thus, when the inequality above is satisfied, AdaDelta is locally convergent with exponential rate of convergence.

5.1.4 Conclusion

These proofs of convergence, show the generality of this method. Only for AdaGrad we are not able to proof the convergence due to the missing decay rate β. The result can also be used to adjust the hyperparameters for future optimizations. Table 14 shows the

resulting hyperparameter bounding for each algorithm.

It is also quite astonishing that the convergence behavior of AdaDelta is not dependent on the parameter ϵ. The convergence behavior from the relatively similar algorithms ADAM and RMSProp depend heavily on the ratio between α and ϵ. Many algorithms reduce the learning rate α over time. For ADAM or RMSProp, however, it might also be useful to increase ϵ. At this point it should be noted that due to the addition with v it is not mathematically equivalent.

Table 14: Hyperparameter bounds for popular adaptive gradient descent optimizers

Algorithm	Hyperparameter bounding
SGD [RM51]	$\max_{i=1}^{n}(\mu_i) < \frac{2}{\alpha}$
RMSProp [Hin+12b]	$\max_{i=1}^{n}(\mu_i) < \frac{2\epsilon}{\alpha}$
AdaGrad [DHS11]	No convergence statement possible with this method
AdaDelta [Zei12]	$\max_{i=1}^{n}(\mu_i) < \frac{2}{\alpha}$
ADAM without bias-correction [KB15]	$\frac{\alpha}{\epsilon} \max_{i=1}^{n}(\mu_i)(1-\beta_1) < 2\beta_1 + 2$

5.1.5 Experiments

The same experiments as in Subsection 4.4.2 can be done with the optimizers of Table 14. Since we only have dependencies on ϵ and α, we only iterate over these hyperparameters with the same color coding (see Table 10) without any inequality beside our hyperparameter bounding. To avoid overloading this section, the experiments can be found in Appendix C.6.

However, AdaDelta plays a special role here, because it is quite astonishing that if you use the parameters suggested by Zeiler ($\alpha = 1$) the convergence behavior only depends on the eigenvalues of the Hessian of f. To be more clear, there are problems which converge or not converge unimpressed by the hyperparameters. To prove this theoretical result, we use the function $f(x) = \frac{1}{2}cw^2$ with $\max(\mu) = c$. For $c = 1.9$ – our inequality satisfied – we reach a convergent behavior for each hyperparameter setting we tested. In contrast, if we choose $c = 2.1$ – our inequality is not satisfied – we reach a non convergent behavior for each hyperparameter setting. At this point we do not use pictures, because they would only show green or red dots.

If we look closer to some fixed hyperparameters we can see that we are reaching the minimum in both cases. But with $c = 2.1$ we are leaving the minimum at approximately iteration 200 (compare Figure 47).

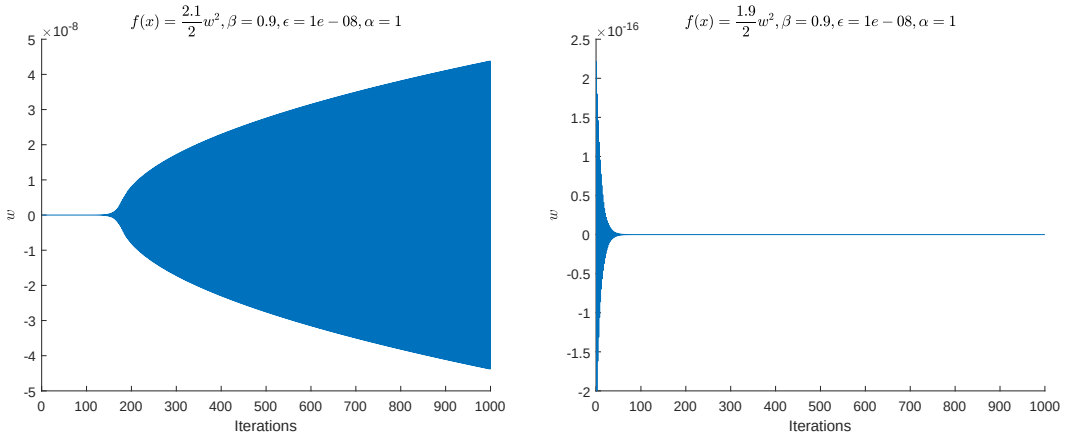
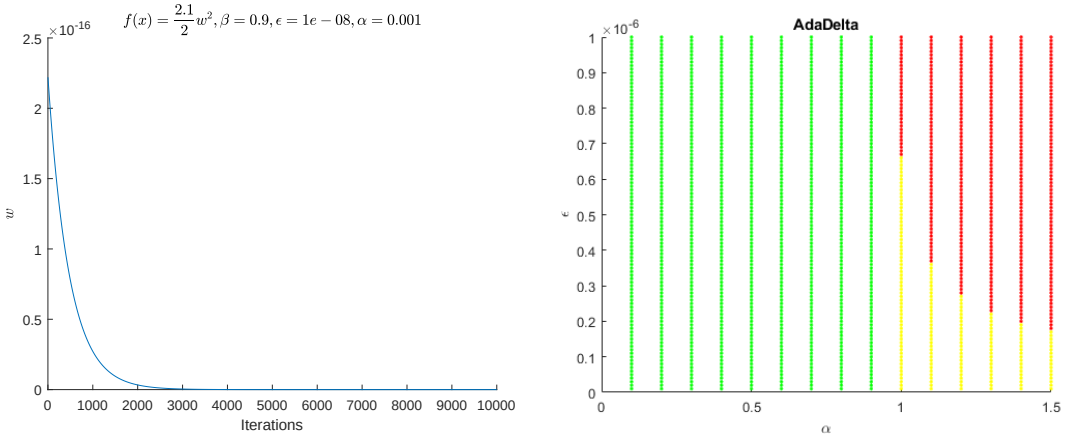


Figure 47: Convergence behavior with fixed hyperparameters

Different to the original paper from Zeiler [Zei12], commonly used frameworks for neural networks add a learning rate to the optimizer. Examples are the implementations in Tensorflow [AAB+16] or in PyTorch [Pas+19]. Tensorflow even goes one step further and mentions the original paper with $\alpha = 1.0$ but sets the default value to $\alpha = 0.001$. As previously shown, choosing a small learning rate can have a positive effect on convergence. See for example Figure 48a with $\alpha = 0.001$. Here we converge more slowly but we do not leave the minimum afterwards.



(a) Convergence behavior with $\alpha = 0.001$

(b) Convergence behavior by iterating over α and ϵ

Figure 48: Convergence behavior of AdaDelta

By iterating over the learning rate α and ϵ with $c = 2.1$, we can prove again our convergence inequality with the colors coded by Table 10 (see Figure 48b). Even if we increase the number of iterations, the red area expands at the cost of the yellow area.

5.2 Global Non-Convergence

We were able to compute some hyperparameter bounds for various ADG algorithms using the local convergence proof. Now we also want to investigate global non-convergence. For this, we look for 2-limit-cycles in the algorithms from Table 13 as in Section 4.5.

5.2.1 RMSProp

For the one dimensional function $f(w) = \frac{1}{2}cw^2$ with $c > 0$ and $\epsilon = 0$, we are able to calculate a 2-limit-cycle $T\left(T\left(x;p\right);p\right) = x$ with the CAS Maple:

$$\tilde{v} = \frac{\alpha^2 c^2}{4}, \tilde{w} = \pm\frac{\alpha}{2} \tag{5.3}$$

Note that $\tilde{v} \neq 0 \neq \tilde{w}$ and specially $\tilde{v} > 0$, thus we have a 2-limit-cycle. To prove the stability of this cycle, we observe the Jacobian of the twofold iteration at $(\tilde{v}, \tilde{w})$.

$$J\left(T\left(T\left(\tilde{v},\tilde{w}\right)\right)\right) = \begin{bmatrix} 3\beta^2 - 2\beta & -c^2\alpha\left(3\beta^2 - 4\beta + 1\right) \\ \frac{-6\beta^2+2\beta}{\alpha c^2} & 6\beta^2 - 6\beta + 1 \end{bmatrix}$$

The eigenvalues are

$$\lambda_{1,2} = \frac{1}{2}\left(1 + 9\beta^2 - 2\beta \pm sgn\left(\beta - \frac{1}{3}\right)(3\beta - 1)\sqrt{9\beta^2 - 10\beta + 1}\right)$$

and we can see that $\lambda_{1,2}$ depends only on the decay rate β.

Therefore, it is only interesting to consider the evolution of the absolute value of $\lambda_{1,2}$ with the parameter β in the range $0 < \beta < 1$ (see Figure 49). $|\lambda_{1,2}|$ looks quite similar except to the area $\beta < 0.1$. But the main point is that the absolute value of the eigenvalues $|\lambda_{1,2}|$ is strictly lower than one in the range $0 < \beta < 1$. Thus the 2-limit-cycle is asymptotically stable. Recall at this point that we observe RMSProp with $\epsilon = 0$. To make a link to the original version from [Hin+12b], we have to apply step three from Section 4.5. Hereby we define

$$F\left(x,\epsilon\right) = T\left(T\left(x;p\right);p\right) - x$$

with fixed hyperparameters α and β. Than we check if $\frac{\partial F}{\partial x}\left(\tilde{x},0\right)$ is invertible in the cycle.

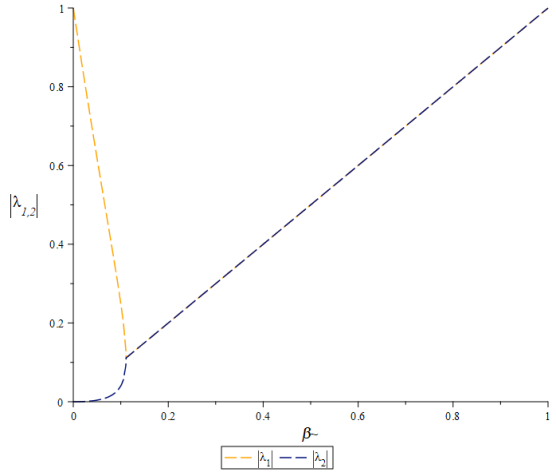


Figure 49: Absolute value of the eigenvalues of the Jacobian of the twofold iteration at $(\tilde{v}, \tilde{w})$.

Calculating the determinant with Maple we get

$$det\left(\frac{\partial F}{\partial x}\left(\tilde{x}, 0\right)\right) = 8\left(\beta - \beta^2\right)$$

which is non-zero for all relevant $0 < \beta < 1$. Now with the Implicit Function Theorem we can show that in the neighborhood of $\epsilon = 0$ exists a unique $x\left(\epsilon\right)$ with $F\left(x\left(\epsilon\right), \epsilon\right) = 0$. This zero of F corresponds to a 2-limit-cycle of T with hyperparameter ϵ. Due to the autonomous character of the RMSProp system we are already done at this point and do not have to apply a disturbance estimate as in step four of Section 4.5. Therefore the autonomous nonlinear system (5.1) has the asymptotically stable 2-limit-cycle (5.3). Consequently the adaptive gradient descent algorithm RMSProp is globally non-convergent, which we now summarize in the following theorem.

Theorem 5.1. *Consider the RMSProp-Optimizer as defined in Algorithm 9 with Table 13 and objective function $f(w) = \frac{1}{2}cw^2, c > 0$. Then the algorithm is locally convergent under the assumptions stated in Table 14. However, there exist solutions that converge to the 2-limit-cycle $\tilde{v} = \frac{\alpha^2 c^2}{4}, \tilde{w} = \pm\frac{\alpha}{2}$. So the algorithm does not converge globally to the minimum $w_\star = 0$.*

We are able to make equal experiments as in Subsection 4.5.2 for the ADAM. Again we set $f(w) = \frac{1}{2}cw^2, \alpha = 0.001, c = 2.1$ and $\beta = 0.8$. With the corresponding inequality from Table 14 we reach convergence with $\epsilon > \frac{c\alpha}{2} = 0.0011$. We plot $\epsilon = 0.0011$ with a red cross to signal the shift into convergence. Once again we see the 2-limit-cycle

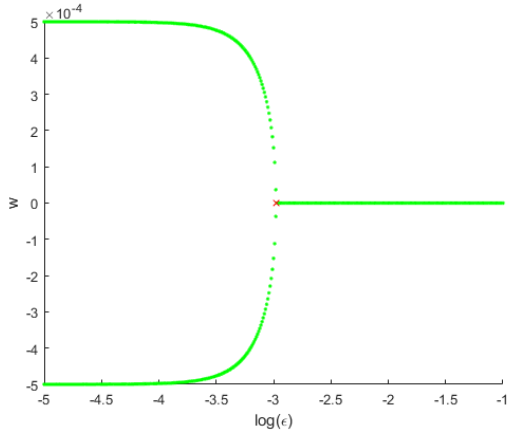


Figure 50: Iterating ϵ from 10^{-5} to 10^{-1}.

and therefore the non-convergence by not solving the hyper parameter bound and a convergent behavior by solving the inequality.

With Theorem 5.1 we have proved that there are asymptotically stable 2-limit-cycles in the RMSProp. But instead we could also ask when our fixed point $x_\star = (v_\star, w_\star) = (0, 0)$ becomes stable in setting above. Therefore we observe the fixed point $x_\star$ in the RMSProp system with $\epsilon > 0$ and the convex function $f(w) = \frac{1}{2}cw^2$. Here we can calculate the eigenvalues of the Jacobian in $x_\star$:

$$\lambda_1 = \beta^2 \qquad \lambda_2 = \frac{(\alpha c - \epsilon)^2}{\epsilon^2}$$

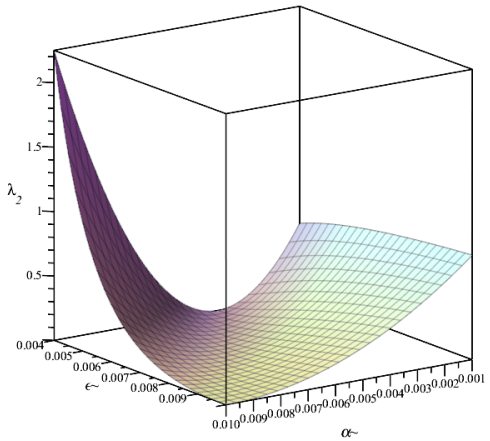


Figure 51: Second eigenvalue of the Jacobian and $f(w) = \frac{1}{2}w^2$.

β is always between 0 and 1 and thus always an indicator of stability, no matter what

hyperparameter setting is used. Therefore we only observe λ_2.

$$\begin{aligned} |\lambda_2| &< 1 \\ \frac{(\alpha c - \epsilon)^2}{\epsilon^2} &< 1 \\ (\alpha c - \epsilon)^2 &< \epsilon^2 \\ (\alpha c)^2 &< 2\alpha c\epsilon \\ c &< \frac{2\epsilon}{\alpha} \end{aligned} \tag{5.4}$$

In the second step we only observe the upper bound due to the greater 0 fraction and in the last step we set $c \neq 0$. Therefore is the fixed point $(0, 0)$ asymptotically stable if inequality (5.4) holds true. Note that we get exactly the inequality from Table 14. In Figure 51 we can see the behavior of λ_2 and with a bigger ϵ we reach a $\lambda_2 > 1$ and thus a break with the asymptotic stability.

An other way to check stability of fixed points is to find a Lyapunov function. These functions must correspond to the following definition.

Definition 5.2. ***(Lyapunov function)*** *Let $f : D \to \mathbb{R}^n$ be locally Lipschitz in $D \subset \mathbb{R}^n$ and the fixed point $x_\star = 0 \in D$. A function $V : D \to \mathbb{R}$ which is continuous and fulfill*

$$V(x_\star) = 0 \text{ and } V(x) > 0 \quad , \forall x \in D \setminus \{0\} \tag{5.5}$$

$$V(f(x)) - V(x) \leq 0 \quad , \forall x \in D \tag{5.6}$$

is called Lyapunov function.

For differential systems exists an analog definition. So with a Lyapunov function and Theorem 1.2 in [BCS18], we could also show asymptotic stability of a fixed point $x_\star$. Unfortunately we are not able to give a general Lyapunov function for RMSProp, but in the special case of $f(w) = \frac{1}{2}w^2$ we are able to give a Lyapunov function.

Theorem 5.3. *Set the RMSProp algorithm after Table 13 and Algorithm 9 and set the function $f(w) = \frac{1}{2}w^2$. Than*

$$V(x_t) = v_t + w_t^2$$

is a Lyapunov function. Thus the fixed point $x_\star = 0$ is asymptotically stable.

Proof. Recall system (5.1) of the RMSProp

$$T(v_t, w_t) = \begin{bmatrix} v_{t+1} \\ w_{t+1} \end{bmatrix} = \begin{bmatrix} \beta v_t + (1-\beta) g(w_t)^2 \\ w_t - \alpha \frac{g(w_t)}{\sqrt{v_{t+1}+\epsilon^2}} \end{bmatrix}$$

and set the function

$$V(x_t) = v_t + w_t^2$$

Firstly we prove equation (5.5) of Definition 5.2. Easily we can detect $V(x_\star) = 0$ and $V(x) > 0$ with x in an area around $x_\star$. Note that by definition $v_t > 0$ follows. So at least we have to proof $V(x_{t+1}) - V(x_t) \leq 0$. Therefore we separate the proof in the v_t and in the w_t parameter. Notice that we can rewrite $v_{t+1} = \beta v_t + (1-\beta) g(w_t)^2 = (1-\beta) \sum_{i=0}^{t} \beta^{t-i} g(w_i)^2$. Herewith we can write

$$\begin{aligned} v_{t+1} - v_t &= (1-\beta) \sum_{i=0}^{t} \beta^{t-i} g(w_i)^2 - (1-\beta) \sum_{i=0}^{t-1} \beta^{t-1-i} g(w_i)^2 \\ &= (1-\beta) \left(\sum_{i=0}^{t} \beta^{t-i} g(w_i)^2 - \frac{1}{\beta} \sum_{i=0}^{t-1} \beta^{t-i} g(w_i)^2 \right) \\ &= (1-\beta) \left(\beta g(w_t)^2 - \frac{1}{\beta} \right) \sum_{i=0}^{t-1} \beta^{t-i} g(w_i)^2 \end{aligned}$$

Therefore the first and the third term are always positive, we investigate our research to the term in the middle which has to be negative.

$$\begin{aligned} \beta g(w_t)^2 - \frac{1}{\beta} &< 0 \\ g(w_t)^2 &< 1 \end{aligned}$$

Therefore $g(w_t)^2 = \nabla f(w_t)^2$ is a descent direction and tends to zero, there exists a t so that $g(w_t)^2 < 1$ is true. Note that this proof works for all differentiable functions $f(x)$.

So lets have a look at the w case.

$$\begin{aligned} w_{t+1}^2 - w_t^2 &= w_t^2 - 2\alpha \frac{cw_t^2}{\sqrt{v_{t+1}+\epsilon^2}} + \alpha^2 \frac{c^2 w_t^2}{v_{t+1}+\epsilon^2} - w_t^2 \\ &= -2\alpha \frac{cw_t^2}{\sqrt{v_{t+1}+\epsilon^2}} + \alpha^2 \frac{c^2 w_t^2}{v_{t+1}+\epsilon^2} \overset{!}{\leq} 0 \end{aligned}$$

If the last inequality holds, we have proved that $V(x)$ is a Lyapunov function for RMSProp and $f(w) = \frac{1}{2}cw^2$.

$$\begin{aligned} -2\alpha \frac{cw_t^2}{\sqrt{v_{t+1}+\epsilon^2}} + \alpha^2 \frac{c^2 w_t^2}{v_{t+1}+\epsilon^2} &\leq 0 \\ \alpha^2 \frac{c^2 w_t^2}{v_{t+1}+\epsilon^2} &\leq 2\alpha \frac{cw_t^2}{\sqrt{v_{t+1}+\epsilon^2}} \\ \alpha \frac{c}{\sqrt{v_{t+1}+\epsilon^2}} &\leq 2 \\ c &\leq \frac{2}{\alpha}\sqrt{v_{t+1}+\epsilon^2} \end{aligned}$$

Note that $v_t \geq 0$ and we can generalize this equation to

$$c \leq \frac{2\epsilon}{\alpha}$$

Therefore c is the maximal eigenvalue of the Hessian of $f(w_t)$, we see the same equation as calculated in Table 14. We also see that equation (5.6) holds true and so $V(x)$ is a Lyapunov function for the RMSProp with $f(w) = \frac{1}{2}cw^2$. With Theorem 1.2 from [BCS18] we can also prove the asymptotic stability of $x_\star$. □

5.2.2 AdaGrad

We could not find any explicit 2-limit-cycle in AdaGrad. Larger cycles could not even be calculated with Maple. Thus the prove method cannot be applied to the AdaGrad.

5.2.3 AdaDelta

Recall system (5.2) with the special case $\alpha = 1$ equally to the original system [Zei12]. Again we are looking at the 2-limit cycle solving $T(T(x;p);p) = x$ with $\epsilon = 0$. With Maple we can identify

$$m_{t+1} = \frac{4v_t}{\alpha^2 c^2} \quad v_{t+1} = v_t \quad w_{t+1} = \pm\frac{\sqrt{v_t}}{c} \tag{5.7}$$

as a 2-limit cycle of AdaDelta. At first glance, the dependence of m_t is noticeable. Therefore the 2-limit cycle cannot be asymptotically stable in the v-parameter, since with a small disturbance on v_t there is always a 2-limit cycle with suitable parameters m_t, w_t as described above. This insight can also be found in the eigenvalues of the Jacobian of the twofold system at the 2-limit cycle.

$$\lambda_1 = 1 \quad \lambda_2 = \beta^2 \quad \lambda_3 = (2\beta - 1)^2$$

We see that λ_2 and λ_3 are less than one for the range $0 < \beta < 1$, signaling asymptotic stability. However λ_1 is equal to one for any parameter and therefore the 2-limit cycle has to be unattractive. This appears logically as soon as we look again on the cycle. Each disturbance in v_t generates another 2-limit cycle which is stable for the disturbed v_t. But with variable v_t we have a set of 2-limit cycles which can be arbitrarily close to zero by decreasing v_t. In contrary to [BW19b] we do not reach an asymptotically stable cycle but a set of stable cycles, which can be arbitrarily close to the solution.

The next step in [BW19b] is to apply the implicit function theorem to check the existence of the cycles even for $\epsilon \neq 0$. Since the Jacobian matrix is singular for $\epsilon = 0$, we cannot continue analogously. Therefore we define $F(m, w, \epsilon) = T(T(m, w; p); p) - x$ as a two dimensional system with the constant hyperparameter $v \in p$. Now we check the invertability of $\frac{\partial F}{\partial x}(\tilde{x}, 0)$ in a state $\tilde{x}$ on a 2-cycle for $\epsilon = 0$ as in equation (5.7). Therefore we calculate the determinant

$$det\left(\frac{\partial F}{\partial x}(\tilde{x}, 0)\right) = 4\left(\beta^4 - \beta^2\right)$$

which is non-zero for all relevant $0 < \beta < 1$. With the same arguments as for RMSProp or ADAM we can say that the two dimensional system is globally non-convergent. Since v_t is independent of ϵ and the 2-limit cycle exists for a constant v_t, we even prove the global non-convergence of the AdaDelta, which we summarize in the following theorem.

Theorem 5.4. *Consider the AdaDelta-Optimizer as defined in Algorithm 9 with Table 13 and objective function $f(w) = \frac{1}{2}cw^2, c > 0$. Then the algorithm is locally convergent under the assumptions stated in Table 14. However, there exist solutions that converge to the set of cycles $\tilde{m} = \frac{4v}{\alpha^2 c^2}, \tilde{v} = v, \tilde{w} = \pm\frac{\sqrt{v}}{c}$. So the algorithm does not converge globally to the minimum $w_\star = 0$.*

In Figure 52 (left side) we can see that the AdaDelta stays in the 2-limit-cycle (5.7) with the parameters $v_{start} = 1, \alpha = 0.1, \epsilon = 10^{(-4)}$ and $\beta = 0.9$. $\tilde{w} = \pm 1, \tilde{m} = 400$ and $\tilde{v} = 1$ are constant in the experiment. Even if we start in disturbed starting values like $w = \tilde{w} + 0.001$ and $m = \tilde{v} - 0.001$ AdaDelta converges to another cycle. For example

in Figure 52 right side it is converging against $v = 0.9999, m = 399.9698$ and $w = \pm 1$. Thereby m and w follows equation (5.7) with $v = 0.9999$.

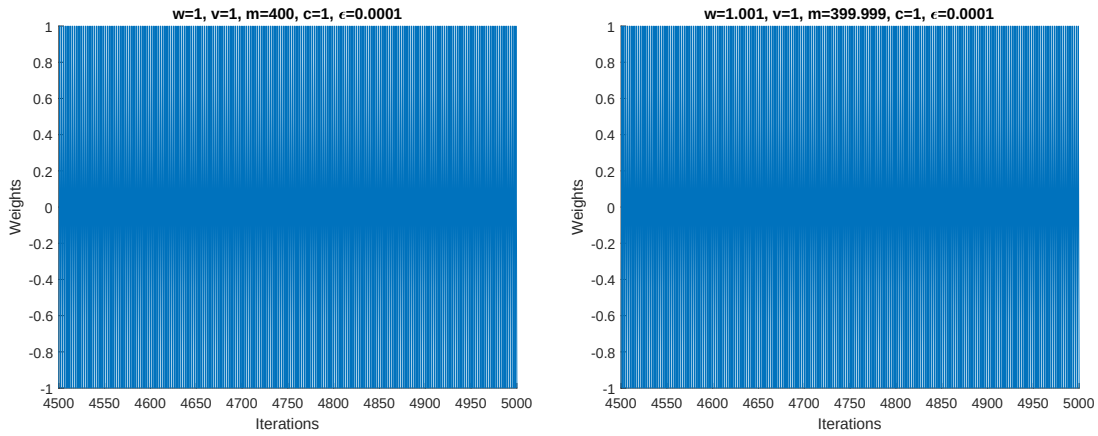


Figure 52: 2-limit-cycle with and without disturbance.

5.2.4 AMSGrad

Reddi et. al. [RKK19] introduce the AMSGrad as an improvement of the ADAM due to their non-converging counter example. They improve the ADAM with a maximums function in the second momentum v_t (see Algorithm 10). Thus v_t is a monotonic growing function and can not enlarge the weight change. In the original paper [RKK19] they do not use a bias correction and write that one has to use ϵ "problem-dependent". Therefore we also do not use a bias correction and set ϵ^2 as addition under the square root to be in common with our first ADAM definition (see Algorithm 6).

Algorithm 10 AMSGrad Optimizer by Reddi, Kale and Kumar[RKK19]

Require: $\alpha \in \mathbb{R}^+, \epsilon \in \mathbb{R}, \beta_1, \beta_2 \in (0,1), w_0 \in \mathbb{R}^n$ and the function $f(w) \in C^2(\mathbb{R}^n, \mathbb{R})$

1: $m_0 = 0, v_0 = 0, t = 0$
2: **while** w not converged **do**
3: $m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla_w f(w_t)$
4: $v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla_w f(w_t)^2$
5: $\hat{v}_{t+1} = \max(\hat{v}_t, v_{t+1})$
6: $w_{t+1} = w_t - \alpha \frac{m_{t+1}}{\sqrt{\hat{v}_{t+1}+\epsilon^2}}$
7: $t = t + 1$
8: **end while**

Due to the maximum function we can not derive the AMSGrad system and so we

can not apply our methods to this type of adaptive gradient descent algorithm. However by observing 2-limit-cycles in the AMSGrad, we found with the CAS Maple the same cycle as in ADAM (see equation (4.15)). Due to the maximum function we are not able to give a statement about the attractiveness of the cycle. But in experiments we see analogous behavior as in the ADAM. Even inequality (4.3) has always been valid (see red cross in Figure 53). Thus v_t is constant in the 2-limit-cycle the maximum function does not affect it and the behavior is similar to the ADAM. The solution to avoid this cycle would be the maximum function applied to m_t. However, this means that we lose the sign of the gradient and thus the direction of descent.

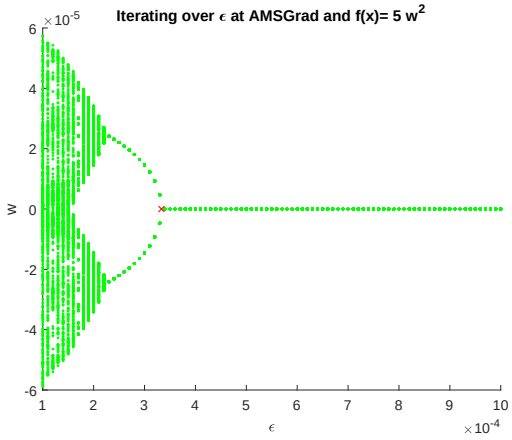


Figure 53: 2-limit-cycle in AMSGrad with inequality 4.3

Tests with deleting the momentum of m_t and v_t when we are in a 2-limit-cycle were unfortunately unsuccessful. The optimizer always moved back towards the 2-limit-cycle. The only way to avoid the 2-limit-cycles is to select the hyper parameters ϵ, β_1 and α "problem-dependent" with a help of inequality (4.15).

5.2.5 Conclusion

This section shows the global non-convergence of some of gradient descent algorithms described in Table 13. We conclude this results in Table 15. It is quite astonishing that we can only find 2-limit-cycles for optimizers with momentum. So the momentum causes in analytic research convergence issues due to 2-limit-cycles. Nevertheless, many of the most widely used optimizers (i.e. ADAM or AMSGrad) use momentum. We suppose that due to three points such cycles often do not occur or are not noticed in practice.

- *Small cycles:* Every 2-limit-cycle (except AdaDelta) was in a small area around the solution. If we can approximate this behavior to other functions it is possible

Table 15: Non-convergence results for famous GD optimizers

Algorithm	Hyperparameter bounding
SGD [RM51]	Cannot find a 2-limit-cycle with Maple
RMSProp [Hin+12b]	Found a 2-limit-cycle, even with $\epsilon \neq 0$
AdaGrad [DHS11]	Cannot find a 2-limit-cycle with Maple
AdaDelta [Zei12]	Found a set of 2-limit-cycles
ADAM without bias-correction [KB15]	Found a 2-limit-cycle, even with $\epsilon \neq 0$
AMSGrad [RKK19]	Found a 2-limit-cycle. Only empirical results for stability.

to confuse a cycle with a minimum. Especially for neural networks with an amount of weights a small cycle may not be detected.

- *Well suited Hyperparameters:* In Table 14 we saw conditions for local convergence. Default parameters in these optimizers are often fulfill these conditions very well. Furthermore time dependent learning rates α_t can also destroy 2-limit-cycles.

- *Stochastic incremental mode:* Especially in the area of neural networks these optimizers are used in stochastic incremental mode. Therefore each function f_i changes randomly every epoch. Even if there exists n-limit-cycles in incremental mode, this stochastic change probably destroy the cycle.

6 Conclusion and Outlook

First, in this work we studied the ability of neural networks to determine rotations of two-dimensional objects. We showed that naive methods (e.g. line search method) could not keep up with neural networks. With professional methods (e.g. PatMax), matching architectures behave in a roughly comparable way. Especially in applications with high interference, the neural networks performed well. Here, it must be assessed in the individual application case. Neural networks have the disadvantage that they require an enormous number of images for training. However, no image processing expert has to program the analysis afterwards.

In addition, this application was tested with real data. Average accuracies smaller than the pixel accuracy could be achieved. Depending on the application, neural networks can also be used in industrial applications.

In Section 2.5 we checked the pose estimation with neural networks of 3D objects with three degrees of freedom. For this, we used different camera perspectives and were able to find out that two single networks with a concatenating layer at the end predicts most accurately (called concatenate architecture in Section 2.5). Furthermore, we were able to show experimentally that more camera perspectives bring an accuracy advantage. However, redundant cameras do not produce an advantage. Therefore, the camera perspectives in real applications should always be shifted axis-wise to each other. Unless the objects have symmetries, in which case redundant cameras could be helpful. However, this has not been tested and should be tested in upcoming experiments.

We analyzed the U-shape – appeared in the experiments of Chapter 2 – in Chapter 3. We showed analytically and experimentally that clipping off the disturbances and an unbalanced pixel ratio between black and white pixels leads to this phenomenon. Comparable experiments with salt and pepper noise could be analytically refuted. We also examined different edge behavior in the pixel area. However, a similar U-shape phenomenon always occurred.

In the further course of the work, we then concerned ourselves with the convergence of the ADAM optimizer. The errors in the proof of convergence were clearly recognized in the previous chapters. With the ADAM in batch mode represented as a system, we could prove its local convergence. In doing so, we developed the a posteriori condition (inequality 4.3) for convergence. We backed up this theoretical proof with experiments.

By using CAS, we were able to find 2-limit-cycles for the ADAM in batch mode. These cycles proved the local non-convergence. Moreover, we also proved these cycles experimentally.

With the use of both papers [AT21] and [BWG18], we also proved local convergence for the ADAM in incremental mode.

We applied the methods developed in the previous chapters to other ADG methods. We determined hyperparameter boundaries for local convergence for SGD, RMSProp and AdaDelta. Thus, users can better select their hyperparameters using these boundaries to converge to a solution. We were also able to find 2-limit cycles for RMSProp, AdaDelta and AMSGrad. Therefore, we know that these algorithms are not globally convergent and users should therefore carefully select the hyperparameters.

Nevertheless further work is needed:

- Only rotations in an industrial environment were considered. For a pose estimation with six degrees of freedom, the translation in X, Y and Z direction still has to be predicted.
- Our tests in pose estimation are not completely comparable with industrial conditions. For this, tests would have to be carried out in companies in order to be able to correctly assess the influence of production fluctuations, employee changes and quality fluctuations. Due to the increased ability of neural networks to react to disturbances, we therefore assume an advantage in applications with neural networks.
- For individual algorithms (e.g., AdaGrad), we could not make any statement about convergence. These should be investigated in future work.
- In incremental mode, we could only show convergence for the ADAM. Similar proofs would be desirable for the other ADG algorithms.
- As already mentioned in the introduction, the field of AI and especially neural networks is growing rapidly due to increased computing power. This means that new discoveries and algorithms are being created every day. New optimizers can be tested for convergence using the method described in this thesis.

This work has shown that one does not have to neglect theory even in a work that is initially rather practical. We would like to conclude the work with a quotation from Eliezer Yudkowsky in order to point out the importance of theoretical foundations.

¨By far the greatest danger of Artificial Intelligence is that people conclude too early that they understand it.¨

Eliezer Yudkowsky [22]

[22] See [Yud08]

A Notation

α	Learning rate in the corresponding ADG
β_1, β_2	Momentum parameter in the corresponding ADG
$\mathbb{C}$	Set of all complex numbers
∇f	Gradient of the function f
$\nabla^2 f$	Hessian of the function f
$diag(v)$	Matrix with the vector v on the diagonal and otherwise zero as entries
ϵ_{mach}	Machine accuracy
$J_T(x_\star)$	Jacobian matrix of the System T at $x_\star$
m_t	First momentum at time step t in momentum methods
$\mathbb{N}$	Set of all natural numbers without zero
$\mathcal{N}(\mu, \sigma_0^2)$	Gaussian distribution with expectation μ and variance σ_0^2
$\mathcal{O}$	Landau notation
$\mathbb{R}$	Set of all real numbers
κ	Spectral condition number
$R(T)$	Regret bound of an algorithm depending on the time
$relu$	Rectifier linear unit
$\rho(A)$	$\max\{\|\lambda\| \mid \lambda \text{ eigenvalue of } A\}$ - Spectral radius
v_t	Second momentum at time step t in momentum methods
$x^\perp$	Transposed of a vector or matrix
$\mathcal{U}_{[a,b]}$	Continuous uniform distribution on the closed interval $[a, b]$

B Abbreviations

AdaDelta Adaptive Learning Rate Method

AdaGrad Adaptive Gradient Algorithm

ADAM Adaptive Moment Estimation

AdaMax ADAM based on the infinity norm

AdamW ADAM with decoupled weight decay

ADG Adaptive Gradient Decent

AI Artificial Intelligence

AMSGrad Combination of ADAM and RMSProp

CAS Computer Algebra Software

CNN Convolutional Neural Network

GD Gradient Descent

LASSO Least absolute shrinkage and selection operator

LEAP Learning Pose

MAE Mean Absolute Error

MNIST Modified NIST

MSE Mean Squared Error

Mobilenet Architecture for mobile and embedded vision applications

Nadam Nesterov-accelerated adaptive moment estimation

NIST National Institute of Standards and Technology

PLOSS PoseLoss

ResNet Residual Network

RMSProp Root Mean Square Propagation

SnP Salt and Pepper

SGD Stochastic Gradient Descent

SLOSS ShapeMatch-Loss

SVM Support Vector Machine

VGG Visual Geometry Group

C Technical Appendix

C.1 Experiments to Estimate the 1D Pose Estimation

Table 16: Test noise: Without noise / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
PatMax						0.015	Without noise
CNN	mse	0.443	9.139	0.443	9.139	0.210	Without noise
PoseCNN	ploss	0.470	2.774	0.436	2.376	0.244	Without noise
CNN	mse	0.260	0.240	0.260	0.240	0.260	Salt and pepper
PoseCNN	mse	0.314	0.271	0.313	0.271	0.313	Gaussian noise with random σ
PoseCNN	ploss	0.762	3.404	0.736	3.358	0.319	Gaussian noise with $\sigma = 0.01$
PoseCNN	mse	0.330	0.228	0.327	0.230	0.326	Without noise
CNN	ploss	0.908	5.501	0.849	5.028	0.343	Without noise
CNN	ploss	1.277	12.667	1.184	11.525	0.351	Gaussian noise with random σ
CNN	ploss	0.736	1.899	0.719	1.890	0.352	Gaussian noise with $\sigma = 0.01$
PoseCNN	ploss	0.876	7.134	0.795	5.525	0.428	Without noise
Mobilenet	mse	0.686	9.139	0.683	9.140	0.449	Salt and pepper
CNN	mse	0.454	0.283	0.453	0.283	0.453	Salt and pepper
VGG mod	mse	0.474	0.438	0.467	0.442	0.465	Salt and pepper
Mobilenet	mse	0.484	0.367	0.473	0.369	0.469	Without noise
PoseCNN	ploss	1.868	19.562	1.778	18.966	0.493	Gaussian noise with random σ
CNN	mse	3.731	33.915	3.730	33.915	0.498	Gaussian noise with $\sigma = 0.01$
VGG mod	ploss	5.363	7.149	5.246	7.167	0.498	Gaussian noise with $\sigma = 0.01$

Table 16: Test noise: Without noise / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
ResNet50	mse	0.770	9.145	0.758	9.146	0.523	Without noise
Mobilenet	mse	3.136	30.071	3.111	30.073	0.565	Salt and pepper
CNN	ploss	1.404	11.812	1.296	10.858	0.568	Gaussian noise with $\sigma = 0.01$
VGG mod	mse	0.591	0.526	0.577	0.530	0.571	Gaussian noise with random σ
PoseCNN	ploss	1.199	13.185	1.074	11.565	0.584	Gaussian noise with random σ
CNN	ploss	1.102	10.695	0.942	7.578	0.604	Salt and pepper
PoseCNN	mse	0.852	9.017	0.852	9.017	0.626	Without noise
PoseCNN	mse	0.856	8.926	0.855	8.926	0.634	Gaussian noise with $\sigma = 0.01$
VGG mod	mse	0.657	0.489	0.643	0.495	0.635	Gaussian noise with random σ
VGG mod	mse	0.661	0.567	0.647	0.573	0.640	Without noise
VGG mod	ploss	7.360	20.511	7.191	20.072	0.647	Without noise
PoseCNN	mse	0.658	0.515	0.657	0.516	0.657	Salt and pepper
PoseCNN	ploss	1.842	15.053	1.673	12.745	0.660	Gaussian noise with $\sigma = 0.01$
ResNet50	mse	1.142	12.900	1.141	12.900	0.677	Salt and pepper
PoseCNN	ploss	1.512	15.233	1.354	13.267	0.679	Salt and pepper
CNN	ploss	1.292	11.266	1.120	8.146	0.728	Gaussian noise with random σ
Line Search						0.732	Without noise
VGG mod	ploss	7.326	26.266	7.170	26.287	0.734	Gaussian noise with random σ
PoseCNN	ploss	6.204	21.166	6.037	20.162	0.748	Salt and pepper
CNN	mse	0.762	0.629	0.762	0.629	0.762	Gaussian noise with random σ
VGG mod	ploss	4.650	10.662	4.503	10.601	0.784	Salt and pepper
CNN	ploss	1.974	19.163	1.693	15.675	0.822	Salt and pepper

Table 16: Test noise: Without noise / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
ResNet50	mse	2.471	24.046	2.467	24.047	0.846	Salt and pepper
VGG mod	ploss	6.019	11.091	5.787	9.275	0.848	Without noise
VGG mod	mse	0.935	0.679	0.927	0.683	0.925	Gaussian noise with $\sigma = 0.01$
VGG mod	mse	2.578	24.008	2.562	24.010	0.943	Salt and pepper
CNN	mse	0.989	1.126	0.989	1.126	0.988	Gaussian noise with random σ
VGG mod	ploss	5.798	18.105	5.603	17.470	1.015	Salt and pepper
VGG mod	ploss	9.202	31.428	8.918	30.456	1.026	Gaussian noise with random σ
Mobilenet	mse	2.895	25.644	2.881	25.646	1.032	Without noise
ResNet50	mse	1.132	1.156	1.108	1.164	1.100	Gaussian noise with random σ
VGG mod	mse	1.225	0.540	1.224	0.541	1.223	Gaussian noise with $\sigma = 0.01$
CNN	mse	1.226	0.931	1.226	0.931	1.226	Without noise
CNN	mse	4.323	32.561	4.323	32.561	1.332	Gaussian noise with $\sigma = 0.01$
VGG mod	ploss	9.648	33.067	9.337	32.520	1.376	Gaussian noise with $\sigma = 0.01$
PoseCNN	mse	2.358	17.706	2.355	17.706	1.480	Salt and pepper
CNN	ploss	3.623	20.833	3.228	19.047	1.527	Without noise
PoseCNN	mse	2.216	15.675	2.215	15.676	1.528	Gaussian noise with random σ
ResNet50	mse	2.529	15.720	2.360	15.730	1.576	Without noise
PoseCNN	mse	1.988	8.925	1.967	8.927	1.739	Gaussian noise with $\sigma = 0.01$
ResNet50	mse	1.825	0.919	1.809	0.925	1.808	Gaussian noise with random σ
Mobilenet	mse	8.243	46.364	8.228	46.366	2.082	Gaussian noise with random σ
Mobilenet	mse	2.097	1.345	2.095	1.346	2.095	Gaussian noise with random σ
VGG mod	mse	2.104	1.051	2.103	1.051	2.103	Without noise

Table 16: Test noise: Without noise / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
Mobilenet	mse	10.631	35.607	10.612	35.610	6.936	Gaussian noise with $\sigma = 0.01$
Mobilenet	mse	30.659	83.220	30.596	83.236	8.998	Gaussian noise with $\sigma = 0.01$
ResNet50	ploss	36.457	79.823	30.352	67.036	16.108	Salt and pepper
ResNet50	mse	27.388	11.140	25.402	11.636	25.068	Gaussian noise with $\sigma = 0.01$
ResNet50	mse	43.120	40.756	43.072	40.789	40.358	Gaussian noise with $\sigma = 0.01$
Mobilenet	ploss	122.539	89.051	100.606	78.843	54.801	Without noise
Mobilenet	ploss	133.346	47.008	107.527	39.432	59.256	Salt and pepper
ResNet50	ploss	142.997	96.004	124.182	90.238	69.500	Without noise
Mobilenet	ploss	133.843	101.972	103.232	73.790	73.714	Gaussian noise with random σ
Mobilenet	ploss	139.056	100.732	108.878	81.308	75.672	Without noise
Mobilenet	ploss	197.267	104.860	167.129	100.544	83.321	Salt and pepper
ResNet50	ploss	161.677	93.680	134.699	85.134	87.635	Gaussian noise with random σ
ResNet50	ploss	145.839	93.504	107.467	64.346	87.741	Gaussian noise with random σ
ResNet50	ploss	117.078	39.100	99.370	40.497	89.422	Gaussian noise with $\sigma = 0.01$
Mobilenet	ploss	108.177	45.906	97.199	48.109	89.905	Gaussian noise with $\sigma = 0.01$
Mobilenet	ploss	107.300	41.669	97.432	42.909	89.973	Gaussian noise with $\sigma = 0.01$
ResNet50	ploss	171.224	78.736	138.702	66.184	90.338	Without noise
ResNet50	ploss	203.577	21.249	175.642	3.977	91.391	Gaussian noise with $\sigma = 0.01$
Mobilenet	ploss	171.823	95.643	132.858	73.739	95.066	Gaussian noise with random σ
ResNet50	ploss	207.780	49.450	163.067	37.740	117.023	Salt and pepper

Table 17: Test noise: SnP / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
PatMax						0.016	Without noise
CNN	mse	0.300	0.283	0.300	0.283	0.300	Salt and pepper
Mobilenet	mse	0.868	12.916	0.865	12.916	0.399	Salt and pepper
CNN	mse	0.405	0.324	0.404	0.325	0.404	Salt and pepper
VGG mod	mse	0.448	0.400	0.442	0.403	0.440	Salt and pepper
PoseCNN	ploss	1.133	9.590	1.102	9.582	0.538	Gaussian noise with $\sigma = 0.01$
Mobilenet	mse	3.599	32.643	3.574	32.645	0.574	Salt and pepper
CNN	ploss	1.336	14.053	1.174	11.817	0.606	Salt and pepper
PoseCNN	ploss	1.374	13.145	1.230	11.255	0.659	Salt and pepper
PoseCNN	ploss	2.372	22.468	2.346	22.463	0.661	Gaussian noise with random σ
ResNet50	mse	0.707	0.494	0.707	0.494	0.707	Salt and pepper
CNN	ploss	1.461	12.510	1.316	9.689	0.726	Gaussian noise with random σ
PoseCNN	ploss	1.128	8.250	0.998	5.727	0.727	Gaussian noise with random σ
Line Search						0.739	Without noise
PoseCNN	ploss	6.435	22.989	6.266	22.062	0.756	Salt and pepper
PoseCNN	mse	0.769	0.645	0.768	0.646	0.767	Salt and pepper
VGG mod	ploss	4.660	10.675	4.512	10.619	0.785	Salt and pepper
CNN	ploss	1.966	19.058	1.685	15.534	0.825	Salt and pepper
ResNet50	mse	1.807	18.208	1.803	18.209	0.876	Salt and pepper
PoseCNN	mse	0.935	0.710	0.935	0.710	0.935	Gaussian noise with random σ
VGG mod	mse	2.597	24.012	2.581	24.013	0.961	Salt and pepper
VGG mod	ploss	5.953	19.059	5.709	17.932	1.135	Salt and pepper

Table 17: Test noise: SnP / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
PoseCNN	mse	2.419	17.844	2.416	17.844	1.527	Salt and pepper
PoseCNN	mse	1.586	1.253	1.585	1.254	1.585	Without noise
CNN	ploss	4.970	28.769	4.616	28.120	1.911	Gaussian noise with $\sigma = 0.01$
PoseCNN	mse	2.789	8.953	2.788	8.953	2.574	Gaussian noise with $\sigma = 0.01$
PoseCNN	ploss	6.990	38.692	6.827	37.897	2.695	Without noise
PoseCNN	mse	2.938	8.913	2.923	8.915	2.709	Gaussian noise with $\sigma = 0.01$
PoseCNN	mse	3.744	15.546	3.743	15.546	3.085	Gaussian noise with random σ
PoseCNN	ploss	9.309	42.165	9.233	41.949	4.076	Without noise
CNN	mse	5.934	4.729	5.933	4.730	5.933	Gaussian noise with random σ
CNN	mse	7.361	20.494	7.361	20.494	6.212	Gaussian noise with random σ
CNN	ploss	17.095	54.197	16.632	53.738	8.174	Without noise
CNN	ploss	18.080	53.259	17.934	53.147	9.602	Gaussian noise with random σ
CNN	ploss	24.229	68.012	24.060	67.870	10.143	Without noise
PoseCNN	mse	11.083	16.576	11.083	16.576	10.864	Without noise
ResNet50	ploss	29.778	70.304	24.158	53.286	13.888	Salt and pepper
CNN	mse	17.323	11.327	17.323	11.327	17.323	Gaussian noise with $\sigma = 0.01$
CNN	mse	18.155	10.473	18.155	10.473	18.155	Gaussian noise with $\sigma = 0.01$
CNN	ploss	58.798	89.816	58.763	89.819	30.022	Gaussian noise with $\sigma = 0.01$
CNN	mse	31.858	19.016	31.858	19.016	31.858	Without noise
PoseCNN	ploss	53.425	86.451	51.929	85.793	32.675	Gaussian noise with $\sigma = 0.01$
Mobilenet	ploss	92.704	103.094	76.875	82.116	37.645	Salt and pepper
CNN	mse	38.934	22.763	38.934	22.763	38.934	Without noise

Table 17: Test noise: SnP / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
VGG mod	ploss	119.656	103.482	114.363	105.303	53.742	Gaussian noise with random σ
VGG mod	mse	163.588	112.158	163.523	112.239	69.333	Gaussian noise with random σ
VGG mod	mse	156.657	111.236	156.620	111.280	69.900	Gaussian noise with $\sigma = 0.01$
VGG mod	mse	168.014	110.005	167.991	110.032	73.616	Gaussian noise with random σ
Mobilenet	ploss	138.975	110.061	106.421	79.547	75.825	Salt and pepper
VGG mod	mse	170.751	101.746	170.737	101.764	81.837	Without noise
VGG mod	ploss	124.962	84.316	123.902	84.834	82.877	Without noise
Mobilenet	mse	83.491	46.136	83.466	46.163	83.462	Gaussian noise with $\sigma = 0.01$
VGG mod	ploss	143.024	92.830	141.245	93.413	84.303	Gaussian noise with random σ
ResNet50	mse	97.485	48.255	89.535	49.151	84.686	Gaussian noise with random σ
VGG mod	ploss	129.440	78.073	126.057	78.888	85.313	Gaussian noise with $\sigma = 0.01$
Mobilenet	mse	86.515	49.509	86.510	49.515	86.508	Gaussian noise with random σ
ResNet50	mse	102.909	42.781	91.522	46.303	87.316	Gaussian noise with $\sigma = 0.01$
Mobilenet	mse	87.665	50.319	87.664	50.320	87.639	Gaussian noise with random σ
ResNet50	ploss	145.363	64.124	122.612	70.346	88.847	Gaussian noise with random σ
Mobilenet	ploss	171.398	80.540	156.285	81.749	89.269	Without noise
Mobilenet	mse	89.397	51.440	89.380	51.458	89.357	Gaussian noise with $\sigma = 0.01$
ResNet50	ploss	118.400	38.900	100.442	39.972	89.469	Gaussian noise with $\sigma = 0.01$
Mobilenet	ploss	174.995	27.017	141.038	13.182	89.496	Without noise
Mobilenet	ploss	106.219	42.894	96.401	43.772	89.613	Gaussian noise with $\sigma = 0.01$
ResNet50	ploss	187.680	25.156	158.857	5.091	89.698	Gaussian noise with $\sigma = 0.01$
Mobilenet	ploss	112.792	47.415	100.193	49.448	90.079	Gaussian noise with $\sigma = 0.01$

Table 17: Test noise: SnP / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
Mobilenet	mse	91.674	52.124	90.501	52.985	90.136	Without noise
ResNet50	ploss	108.342	44.848	97.375	46.451	90.242	Without noise
ResNet50	mse	190.447	53.941	160.773	46.122	90.475	Gaussian noise with random σ
ResNet50	ploss	197.989	85.883	180.659	89.268	90.515	Gaussian noise with random σ
Mobilenet	mse	92.268	49.714	91.436	50.293	90.692	Without noise
ResNet50	mse	133.429	50.814	115.498	51.602	90.882	Without noise
ResNet50	ploss	206.671	54.451	174.225	44.036	91.223	Without noise
VGG mod	ploss	160.724	89.443	158.371	91.109	91.525	Without noise
Mobilenet	ploss	167.586	38.279	121.613	32.707	92.388	Gaussian noise with random σ
ResNet50	mse	127.991	53.057	109.004	58.181	94.775	Without noise
ResNet50	mse	102.210	52.218	99.564	50.121	97.082	Gaussian noise with $\sigma = 0.01$
VGG mod	mse	183.481	80.861	183.436	80.905	103.043	Gaussian noise with $\sigma = 0.01$
ResNet50	ploss	194.793	75.442	155.670	65.824	103.275	Salt and pepper
Mobilenet	ploss	215.592	59.579	174.716	50.182	103.495	Gaussian noise with random σ
VGG mod	ploss	167.540	68.191	162.423	71.054	110.243	Gaussian noise with $\sigma = 0.01$
VGG mod	mse	180.961	70.005	180.856	70.181	119.022	Without noise

Table 18: Test noise: Gauss $\sigma = 0.01$ / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
PatMax						0.037	Without noise
CNN	ploss	0.966	9.334	0.948	9.333	0.348	Gaussian noise with $\sigma = 0.01$
Mobilenet	mse	0.615	9.140	0.614	9.140	0.381	Gaussian noise with $\sigma = 0.01$
PoseCNN	ploss	1.619	17.507	1.553	16.952	0.429	Gaussian noise with $\sigma = 0.01$
VGG mod	ploss	5.956	17.082	5.850	17.093	0.441	Gaussian noise with $\sigma = 0.01$
CNN	ploss	1.386	11.907	1.268	10.869	0.498	Gaussian noise with $\sigma = 0.01$
CNN	mse	1.937	22.271	1.937	22.271	0.550	Gaussian noise with $\sigma = 0.01$
PoseCNN	ploss	1.804	15.045	1.637	12.818	0.609	Gaussian noise with $\sigma = 0.01$
CNN	ploss	1.616	15.467	1.472	13.352	0.643	Gaussian noise with random σ
VGG mod	mse	0.688	0.541	0.676	0.546	0.673	Gaussian noise with $\sigma = 0.01$
PoseCNN	mse	0.905	8.949	0.903	8.949	0.681	Gaussian noise with $\sigma = 0.01$
PoseCNN	ploss	1.145	9.253	1.006	6.593	0.706	Gaussian noise with random σ
Line Search						0.797	Without noise
PoseCNN	mse	0.947	0.668	0.947	0.668	0.947	Gaussian noise with random σ
PoseCNN	ploss	2.594	22.521	2.478	21.609	1.009	Gaussian noise with random σ
VGG mod	mse	2.026	18.167	2.024	18.167	1.100	Gaussian noise with $\sigma = 0.01$
PoseCNN	ploss	5.556	11.662	5.318	9.635	1.125	Salt and pepper
VGG mod	ploss	9.339	31.922	9.120	31.940	1.219	Gaussian noise with $\sigma = 0.01$
CNN	mse	4.089	31.279	4.088	31.280	1.331	Gaussian noise with $\sigma = 0.01$
PoseCNN	mse	1.380	1.047	1.379	1.048	1.379	Without noise
PoseCNN	mse	1.794	8.983	1.792	8.983	1.569	Salt and pepper
PoseCNN	ploss	3.307	22.787	3.197	22.037	1.627	Salt and pepper

Table 18: Test noise: Gauss $\sigma = 0.01$ / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
PoseCNN	mse	1.960	8.995	1.941	8.997	1.712	Gaussian noise with $\sigma = 0.01$
PoseCNN	mse	2.202	12.764	2.201	12.764	1.749	Gaussian noise with random σ
ResNet50	mse	2.116	1.053	2.112	1.055	2.112	Gaussian noise with $\sigma = 0.01$
ResNet50	mse	2.597	1.418	2.542	1.419	2.529	Gaussian noise with $\sigma = 0.01$
Mobilenet	mse	12.140	55.510	12.137	55.511	3.204	Gaussian noise with $\sigma = 0.01$
PoseCNN	mse	4.412	3.617	4.412	3.618	4.412	Salt and pepper
PoseCNN	ploss	11.442	46.319	11.394	46.327	4.963	Without noise
PoseCNN	mse	5.755	10.165	5.755	10.165	5.532	Without noise
PoseCNN	ploss	15.045	46.750	14.721	45.593	9.619	Without noise
ResNet50	ploss	45.940	53.402	38.658	48.156	14.064	Gaussian noise with $\sigma = 0.01$
CNN	mse	23.702	12.560	23.701	12.560	23.701	Gaussian noise with random σ
CNN	ploss	62.273	92.884	62.244	92.876	30.913	Salt and pepper
CNN	ploss	69.029	92.263	69.009	92.261	37.033	Gaussian noise with random σ
CNN	mse	42.305	22.960	42.305	22.960	42.305	Salt and pepper
CNN	mse	45.091	25.833	45.091	25.833	45.091	Gaussian noise with random σ
CNN	mse	52.897	30.207	52.897	30.207	52.897	Salt and pepper
VGG mod	ploss	123.709	104.059	118.227	106.080	54.978	Gaussian noise with random σ
CNN	ploss	68.795	55.838	68.787	55.827	59.603	Salt and pepper
CNN	ploss	98.192	90.905	98.189	90.903	62.244	Without noise
CNN	mse	66.416	38.487	66.416	38.487	66.416	Without noise
VGG mod	mse	159.818	113.850	159.742	113.942	66.711	Gaussian noise with random σ
VGG mod	mse	143.682	110.313	143.645	110.351	68.262	Salt and pepper

Table 18: Test noise: Gauss $\sigma = 0.01$ / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
CNN	mse	71.697	40.793	71.697	40.793	71.697	Without noise
VGG mod	mse	164.495	111.648	164.468	111.680	72.049	Gaussian noise with random σ
CNN	ploss	108.101	86.348	108.059	86.380	74.736	Without noise
ResNet50	ploss	141.647	105.331	109.035	77.485	75.781	Gaussian noise with $\sigma = 0.01$
ResNet50	mse	100.799	76.419	98.737	76.406	79.355	Salt and pepper
Mobilenet	ploss	149.584	109.434	115.371	81.114	79.818	Gaussian noise with $\sigma = 0.01$
VGG mod	mse	160.675	102.013	160.672	102.016	79.852	Salt and pepper
VGG mod	ploss	175.120	101.008	174.334	101.359	80.256	Salt and pepper
VGG mod	mse	169.019	102.666	169.002	102.689	80.852	Without noise
VGG mod	ploss	124.249	84.838	123.119	85.426	82.188	Without noise
Mobilenet	ploss	149.708	102.719	113.458	73.770	83.470	Gaussian noise with $\sigma = 0.01$
VGG mod	ploss	157.057	96.397	154.733	97.965	83.796	Without noise
ResNet50	mse	99.437	50.641	91.442	51.572	85.648	Gaussian noise with random σ
ResNet50	ploss	139.801	74.786	129.047	77.053	87.870	Salt and pepper
ResNet50	ploss	145.349	64.110	122.587	70.342	88.848	Gaussian noise with random σ
Mobilenet	mse	93.319	55.482	92.330	56.227	89.196	Without noise
Mobilenet	mse	89.267	52.280	89.267	52.281	89.233	Salt and pepper
Mobilenet	mse	96.482	60.410	95.534	61.088	89.322	Without noise
Mobilenet	ploss	169.666	27.761	137.431	14.247	89.454	Without noise
Mobilenet	ploss	144.205	31.786	117.095	26.032	89.495	Gaussian noise with random σ
Mobilenet	mse	89.524	52.051	89.523	52.052	89.506	Gaussian noise with random σ
Mobilenet	ploss	157.938	29.927	128.687	19.274	89.553	Salt and pepper

Table 18: Test noise: Gauss $\sigma = 0.01$ / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
Mobilenet	mse	89.646	52.133	89.629	52.152	89.626	Salt and pepper
Mobilenet	mse	89.808	52.193	89.803	52.198	89.781	Gaussian noise with random σ
Mobilenet	ploss	178.155	70.068	156.135	68.321	89.967	Without noise
Mobilenet	ploss	188.023	26.998	151.034	12.159	90.074	Salt and pepper
ResNet50	ploss	108.413	44.797	97.418	46.418	90.191	Without noise
ResNet50	ploss	194.519	29.089	166.252	11.034	90.219	Salt and pepper
ResNet50	ploss	198.049	85.866	180.688	89.262	90.522	Gaussian noise with random σ
ResNet50	mse	190.074	54.093	160.687	46.271	90.547	Gaussian noise with random σ
ResNet50	mse	134.037	50.484	115.824	51.042	90.860	Without noise
ResNet50	ploss	206.668	54.485	174.254	44.096	91.215	Without noise
ResNet50	mse	112.113	65.735	106.954	68.324	92.290	Salt and pepper
Mobilenet	ploss	146.218	83.568	137.141	85.444	92.705	Gaussian noise with random σ
ResNet50	mse	127.220	53.002	108.431	58.256	94.382	Without noise
VGG mod	ploss	179.591	84.494	178.096	85.451	99.217	Gaussian noise with random σ
VGG mod	ploss	184.492	73.525	182.634	75.444	107.203	Salt and pepper
VGG mod	mse	183.509	63.070	183.427	63.218	125.809	Without noise

Table 19: Test noise: Gauss random σ

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
PatMax						0.039	Without noise
CNN	ploss	1.279	12.624	1.194	11.552	0.349	Gaussian noise with random σ
PoseCNN	mse	0.380	0.354	0.380	0.354	0.380	Gaussian noise with random σ
PoseCNN	ploss	1.017	9.937	0.978	9.893	0.400	Gaussian noise with random σ
VGG mod	mse	0.550	0.427	0.536	0.431	0.530	Gaussian noise with random σ
VGG mod	mse	0.802	8.750	0.788	8.751	0.569	Gaussian noise with random σ
PoseCNN	ploss	0.993	7.158	0.918	5.507	0.598	Gaussian noise with random $\sigma = 0.01$
PoseCNN	ploss	1.640	16.851	1.454	14.942	0.698	Gaussian noise with random σ
Line Search						0.767	Without noise
VGG mod	ploss	7.275	26.240	7.112	26.262	0.778	Gaussian noise with random σ
CNN	ploss	1.431	12.963	1.226	9.419	0.818	Gaussian noise with random σ
CNN	mse	0.864	0.909	0.864	0.909	0.864	Gaussian noise with random σ
Mobilenet	mse	1.850	18.181	1.849	18.182	0.924	Gaussian noise with random σ
Mobilenet	mse	4.272	33.803	4.250	33.806	1.022	Gaussian noise with random σ
CNN	mse	1.512	12.844	1.512	12.844	1.052	Gaussian noise with random σ
ResNet50	mse	1.118	1.245	1.092	1.253	1.082	Gaussian noise with random σ
VGG mod	ploss	10.115	36.743	9.757	35.435	1.185	Gaussian noise with random σ
PoseCNN	ploss	6.364	19.519	5.990	17.132	1.495	Salt and pepper
PoseCNN	mse	2.022	12.863	2.021	12.864	1.561	Gaussian noise with random σ
PoseCNN	mse	1.877	9.116	1.876	9.116	1.653	Gaussian noise with random $\sigma = 0.01$
ResNet50	mse	1.705	0.925	1.686	0.935	1.682	Gaussian noise with random σ
PoseCNN	ploss	3.646	22.112	3.588	21.961	1.889	Salt and pepper

Table 19: Test noise: Gauss random σ

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
PoseCNN	mse	2.400	12.879	2.398	12.879	1.939	Salt and pepper
PoseCNN	mse	2.597	9.286	2.579	9.289	2.349	Gaussian noise with random $\sigma = 0.01$
PoseCNN	mse	3.352	11.038	3.351	11.038	3.121	Without noise
PoseCNN	mse	9.270	10.356	9.270	10.356	9.270	Salt and pepper
PoseCNN	mse	14.970	32.218	14.970	32.218	13.637	Without noise
PoseCNN	ploss	33.974	70.403	33.771	70.057	23.825	Without noise
CNN	ploss	75.864	100.011	75.835	100.009	37.538	Salt and pepper
PoseCNN	ploss	68.279	92.870	65.860	92.602	38.473	Gaussian noise with random $\sigma = 0.01$
Mobilenet	ploss	91.518	100.997	72.581	74.455	46.503	Gaussian noise with random σ
CNN	ploss	91.719	104.165	91.657	104.196	46.562	Gaussian noise with random $\sigma = 0.01$
CNN	mse	50.082	33.742	50.082	33.742	49.850	Salt and pepper
PoseCNN	ploss	59.773	79.566	59.645	79.400	52.484	Without noise
CNN	ploss	68.223	56.420	68.215	56.415	59.995	Salt and pepper
CNN	mse	62.124	40.804	62.123	40.804	62.123	Gaussian noise with random $\sigma = 0.01$
CNN	mse	62.382	39.942	62.382	39.942	62.382	Gaussian noise with random $\sigma = 0.01$
CNN	ploss	103.931	96.056	103.924	96.059	63.582	Gaussian noise with random $\sigma = 0.01$
CNN	mse	64.514	44.254	64.514	44.254	64.514	Salt and pepper
VGG mod	mse	155.581	116.475	155.516	116.552	65.086	Gaussian noise with random $\sigma = 0.01$
VGG mod	mse	142.891	110.544	142.859	110.575	67.798	Salt and pepper
CNN	ploss	95.419	83.530	95.390	83.524	69.201	Without noise
CNN	mse	73.582	50.176	73.582	50.176	73.582	Without noise
VGG mod	mse	148.415	104.810	148.406	104.817	74.470	Salt and pepper

Table 19: Test noise: Gauss random σ

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
CNN	mse	77.372	50.102	77.372	50.102	77.367	Without noise
CNN	ploss	104.572	83.553	104.518	83.597	77.442	Without noise
VGG mod	ploss	153.480	100.078	150.946	101.540	79.253	Without noise
VGG mod	ploss	175.293	101.877	174.506	102.259	79.654	Salt and pepper
VGG mod	ploss	121.377	85.216	120.015	85.912	80.116	Without noise
ResNet50	mse	104.876	76.322	101.658	77.089	80.478	Salt and pepper
VGG mod	mse	165.093	103.043	165.062	103.083	80.671	Without noise
ResNet50	mse	102.456	43.158	91.703	46.465	87.663	Gaussian noise with random $\sigma = 0.01$
ResNet50	ploss	138.384	74.421	127.780	76.764	88.076	Salt and pepper
Mobilenet	mse	96.669	61.487	96.026	61.956	88.914	Without noise
Mobilenet	mse	89.459	52.372	89.458	52.374	89.414	Salt and pepper
Mobilenet	ploss	169.821	27.726	137.607	14.132	89.475	Without noise
ResNet50	ploss	118.474	38.877	100.434	39.967	89.486	Gaussian noise with random $\sigma = 0.01$
Mobilenet	ploss	158.203	29.974	128.938	19.379	89.492	Salt and pepper
Mobilenet	ploss	105.154	43.514	95.810	44.501	89.615	Gaussian noise with random $\sigma = 0.01$
ResNet50	ploss	187.570	25.167	158.764	5.125	89.652	Gaussian noise with random $\sigma = 0.01$
Mobilenet	mse	93.332	55.193	92.504	55.836	89.671	Without noise
Mobilenet	mse	89.718	52.202	89.704	52.218	89.698	Salt and pepper
Mobilenet	mse	89.955	52.354	89.929	52.380	89.718	Gaussian noise with random $\sigma = 0.01$
Mobilenet	ploss	164.609	70.910	144.533	72.289	89.981	Without noise
ResNet50	ploss	108.541	44.743	97.492	46.361	90.154	Without noise
Mobilenet	mse	90.385	52.226	90.368	52.244	90.158	Gaussian noise with random $\sigma = 0.01$

Table 19: Test noise: Gauss random σ

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
Mobilenet	ploss	112.131	47.138	100.003	49.312	90.208	Gaussian noise with random $\sigma = 0.01$
ResNet50	ploss	194.522	29.092	166.253	11.042	90.217	Salt and pepper
Mobilenet	ploss	195.360	27.502	157.711	13.164	90.229	Salt and pepper
ResNet50	mse	134.709	50.268	116.262	50.631	90.869	Without noise
ResNet50	ploss	206.711	54.504	174.308	44.111	91.202	Without noise
ResNet50	mse	112.116	65.894	106.979	68.478	91.862	Salt and pepper
ResNet50	mse	126.922	53.088	108.251	58.348	94.067	Without noise
ResNet50	mse	101.762	52.867	99.186	51.488	97.235	Gaussian noise with random $\sigma = 0.01$
VGG mod	ploss	130.875	61.771	128.258	62.230	101.700	Gaussian noise with random $\sigma = 0.01$
VGG mod	mse	183.081	77.059	182.876	77.266	107.486	Gaussian noise with random $\sigma = 0.01$
VGG mod	ploss	185.668	72.190	183.542	74.467	108.075	Salt and pepper
ResNet50	ploss	187.907	87.965	140.654	66.788	110.014	Gaussian noise with random σ
ResNet50	ploss	187.276	75.178	148.455	62.870	117.224	Gaussian noise with random σ
Mobilenet	ploss	195.749	92.871	143.115	64.111	121.514	Gaussian noise with random σ
VGG mod	ploss	180.536	54.624	177.148	56.044	129.322	Gaussian noise with random $\sigma = 0.01$
VGG mod	mse	180.341	57.937	180.243	58.102	131.793	Without noise

Table 20: Test noise: Gauss $\sigma = 0.05$ / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
PoseCNN	ploss	0.873	3.330	0.840	3.227	0.488	Gaussian noise with $\sigma = 0.01$
PatMax						0.488	Without noise
PoseCNN	ploss	0.890	2.965	0.847	2.901	0.490	Gaussian noise with random σ
CNN	ploss	1.384	12.413	1.240	9.648	0.655	Gaussian noise with random σ
PoseCNN	ploss	2.614	22.636	2.577	22.586	0.810	Gaussian noise with random σ
Line Search						0.873	Without noise
PoseCNN	mse	1.121	8.855	1.119	8.856	0.902	Gaussian noise with $\sigma = 0.01$
PoseCNN	mse	0.935	0.657	0.935	0.657	0.935	Gaussian noise with random σ
PoseCNN	ploss	6.034	17.359	5.730	14.955	1.284	Salt and pepper
PoseCNN	mse	1.405	1.041	1.404	1.042	1.404	Without noise
PoseCNN	mse	1.817	9.046	1.815	9.046	1.589	Salt and pepper
PoseCNN	mse	1.936	8.974	1.914	8.976	1.686	Gaussian noise with $\sigma = 0.01$
PoseCNN	mse	2.858	20.038	2.857	20.038	1.733	Gaussian noise with random σ
PoseCNN	ploss	3.043	19.687	2.866	18.085	1.781	Salt and pepper
PoseCNN	mse	4.366	3.592	4.366	3.593	4.366	Salt and pepper
PoseCNN	ploss	11.748	47.521	11.653	47.313	5.062	Without noise
PoseCNN	mse	5.919	10.249	5.919	10.249	5.694	Without noise
PoseCNN	ploss	15.719	48.044	15.384	46.855	9.898	Without noise
CNN	mse	23.572	12.650	23.571	12.650	23.571	Gaussian noise with random σ
CNN	ploss	62.219	92.865	62.190	92.858	30.879	Salt and pepper
PoseCNN	ploss	63.986	92.820	61.684	92.514	35.265	Gaussian noise with $\sigma = 0.01$
CNN	ploss	69.011	92.178	68.992	92.177	37.077	Gaussian noise with random σ

Table 20: Test noise: Gauss $\sigma = 0.05$ / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
CNN	ploss	77.024	100.387	76.960	100.403	37.366	Gaussian noise with $\sigma = 0.01$
CNN	mse	42.391	23.111	42.391	23.111	42.391	Salt and pepper
CNN	mse	45.159	25.766	45.159	25.766	45.159	Gaussian noise with random σ
CNN	mse	50.659	27.558	50.659	27.558	50.659	Gaussian noise with $\sigma = 0.01$
CNN	mse	51.410	29.416	51.410	29.416	51.410	Gaussian noise with $\sigma = 0.01$
CNN	mse	52.885	30.162	52.885	30.162	52.885	Salt and pepper
VGG mod	ploss	124.281	104.187	118.864	106.237	55.127	Gaussian noise with random σ
CNN	ploss	106.146	102.997	106.139	102.997	57.143	Gaussian noise with $\sigma = 0.01$
CNN	ploss	68.795	55.892	68.787	55.882	59.579	Salt and pepper
CNN	ploss	98.128	90.851	98.125	90.850	62.246	Without noise
CNN	mse	66.449	38.595	66.449	38.595	66.449	Without noise
VGG mod	mse	159.758	114.032	159.684	114.122	66.519	Gaussian noise with random σ
VGG mod	mse	143.706	110.097	143.672	110.132	68.456	Salt and pepper
VGG mod	mse	157.472	113.234	157.426	113.290	68.583	Gaussian noise with $\sigma = 0.01$
CNN	mse	71.816	40.692	71.816	40.692	71.816	Without noise
VGG mod	mse	164.250	111.660	164.225	111.689	71.971	Gaussian noise with random σ
CNN	ploss	108.076	86.440	108.031	86.476	74.652	Without noise
ResNet50	mse	100.863	76.447	98.795	76.442	79.389	Salt and pepper
VGG mod	mse	158.912	101.547	158.908	101.550	80.015	Salt and pepper
VGG mod	ploss	174.575	100.909	173.789	101.253	80.340	Salt and pepper
VGG mod	mse	169.350	102.745	169.332	102.769	80.818	Without noise
VGG mod	ploss	123.684	84.440	122.538	85.040	82.188	Without noise

Table 20: Test noise: Gauss $\sigma = 0.05$ / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
VGG mod	ploss	156.578	96.235	154.235	97.813	83.705	Without noise
ResNet50	mse	99.407	50.561	91.379	51.546	85.652	Gaussian noise with random σ
ResNet50	mse	103.056	42.714	91.690	46.164	87.409	Gaussian noise with $\sigma = 0.01$
ResNet50	ploss	139.761	74.809	129.023	77.063	87.856	Salt and pepper
ResNet50	ploss	145.349	64.112	122.588	70.343	88.847	Gaussian noise with random σ
Mobilenet	mse	93.285	55.390	92.295	56.137	89.173	Without noise
Mobilenet	mse	89.277	52.264	89.276	52.265	89.249	Salt and pepper
Mobilenet	ploss	144.088	31.931	116.934	26.148	89.425	Gaussian noise with random σ
Mobilenet	mse	96.704	60.396	95.765	61.057	89.436	Without noise
ResNet50	ploss	118.376	38.903	100.425	39.976	89.476	Gaussian noise with $\sigma = 0.01$
Mobilenet	mse	89.712	52.083	89.689	52.107	89.508	Gaussian noise with $\sigma = 0.01$
Mobilenet	ploss	169.662	27.775	137.434	14.307	89.516	Without noise
Mobilenet	mse	89.539	52.080	89.539	52.081	89.521	Gaussian noise with random σ
Mobilenet	mse	89.666	52.149	89.648	52.168	89.645	Salt and pepper
Mobilenet	ploss	105.218	43.494	95.855	44.487	89.653	Gaussian noise with $\sigma = 0.01$
ResNet50	ploss	187.641	25.158	158.829	5.101	89.665	Gaussian noise with $\sigma = 0.01$
Mobilenet	mse	89.742	51.558	89.726	51.575	89.671	Gaussian noise with $\sigma = 0.01$
Mobilenet	ploss	157.965	29.945	128.680	19.298	89.693	Salt and pepper
Mobilenet	mse	89.808	52.195	89.803	52.200	89.776	Gaussian noise with random σ
Mobilenet	ploss	178.371	70.309	156.395	68.489	89.909	Without noise
Mobilenet	ploss	187.627	26.817	150.626	12.024	90.101	Salt and pepper
ResNet50	ploss	108.415	44.794	97.418	46.416	90.192	Without noise

Table 20: Test noise: Gauss $\sigma = 0.05$ / Red marked: PatMax / Green marked: Line Search

Archtecture	Loss	l2 mean	l2 std	linf mean	linf std	abs error in deg	Noise
ResNet50	ploss	194.518	29.089	166.250	11.035	90.220	Salt and pepper
Mobilenet	ploss	112.165	47.264	100.030	49.431	90.226	Gaussian noise with $\sigma = 0.01$
ResNet50	ploss	198.048	85.865	180.687	89.262	90.522	Gaussian noise with random σ
ResNet50	mse	190.075	54.088	160.679	46.266	90.559	Gaussian noise with random σ
ResNet50	mse	134.032	50.490	115.816	51.045	90.856	Without noise
ResNet50	ploss	206.669	54.489	174.254	44.100	91.212	Without noise
Mobilenet	ploss	144.805	83.678	135.676	85.359	92.154	Gaussian noise with random σ
ResNet50	mse	112.083	65.728	106.938	68.307	92.276	Salt and pepper
ResNet50	mse	127.237	53.008	108.451	58.267	94.366	Without noise
ResNet50	mse	101.459	51.491	98.754	49.905	97.214	Gaussian noise with $\sigma = 0.01$
VGG mod	ploss	180.477	84.402	179.047	85.325	99.191	Gaussian noise with random σ
VGG mod	ploss	133.972	65.240	131.231	65.652	100.239	Gaussian noise with $\sigma = 0.01$
VGG mod	mse	185.492	78.079	185.315	78.164	105.955	Gaussian noise with $\sigma = 0.01$
VGG mod	ploss	184.844	73.610	183.050	75.455	107.155	Salt and pepper
VGG mod	ploss	175.898	61.942	171.926	64.045	119.899	Gaussian noise with $\sigma = 0.01$
VGG mod	mse	183.567	63.598	183.488	63.738	125.602	Without noise

C.2 3D Pose Estimation Experiments

Table 21: Architecture comparison

Architecture	cameras	l2 mean	l2 std	abs error in deg	abs error x-axis in deg	abs error y-axis in deg	abs error z-axis in deg	noise
big cnn concat	2	0.977	1.079	1.492	0.488662	0.512398	0.490930	Without noise
big cnn concat	4	1.026	1.108	1.544	0.567772	0.376796	0.599652	Salt and pepper
big cnn concat	4	1.045	1.604	1.559	0.616588	0.423081	0.519268	Gaussian noise with random σ
big cnn concat	4	1.056	1.280	1.589	0.656218	0.438788	0.493534	Without noise
big cnn concat	2	1.109	1.401	1.667	0.674244	0.448630	0.543950	Salt and pepper
big cnn concat	4	1.129	1.656	1.657	0.742148	0.400163	0.514401	Salt and pepper
big cnn concat	2	1.159	1.375	1.764	0.626924	0.511846	0.625447	Gaussian noise with random σ
big cnn concat	2	1.198	1.333	1.819	0.746181	0.414285	0.658178	Salt and pepper
big cnn concat	4	1.246	1.170	1.851	0.746935	0.412924	0.691058	Without noise
big cnn concat	2	1.252	1.336	1.909	0.773635	0.463795	0.671723	Gaussian noise with random σ
big cnn concat	2	1.373	1.407	2.080	0.809165	0.573251	0.697227	Without noise
big cnn concat	4	1.429	1.521	2.145	0.884520	0.536299	0.724679	Gaussian noise with random σ
small cnn concat	4	1.446	1.936	2.158	0.878765	0.536254	0.742685	Salt and pepper
small cnn concat	2	1.564	1.959	2.397	0.897960	0.769364	0.730084	Without noise

Table 21: Architecture comparison

Architecture	cameras	l2 mean	l2 std	abs error in deg	abs error x-axis in deg	abs error y-axis in deg	abs error z-axis in deg	noise
small cnn concat	2	1.583	2.166	2.411	0.924626	0.682515	0.804276	Salt and pepper
small cnn concat	2	1.592	1.947	2.408	1.032270	0.607664	0.767985	Without noise
big cnn stack	2	1.608	1.992	2.377	0.992498	0.578747	0.805535	Without noise
small cnn concat	2	1.715	2.217	2.624	0.965437	0.810001	0.848607	Gaussian noise with random σ
small cnn concat	4	1.752	1.714	2.629	1.107673	0.554347	0.966693	Without noise
small cnn concat	2	1.758	2.013	2.642	1.124009	0.720170	0.797798	Salt and pepper
small cnn concat	4	1.769	1.991	2.708	1.100436	0.703002	0.905045	Without noise
big cnn stack	2	1.782	2.271	2.626	1.122711	0.643372	0.859463	Salt and pepper
big cnn stack	2	1.843	2.474	2.777	1.052491	0.824914	0.899106	Gaussian noise with random σ
small cnn concat	4	1.873	2.228	2.871	1.066446	0.811699	0.993231	Salt and pepper
small cnn concat	4	1.910	2.444	2.882	1.143505	0.860042	0.878712	Gaussian noise with random σ
small cnn concat	4	1.918	2.327	2.937	1.142877	0.711093	1.083481	Gaussian noise with random σ
big cnn stack	2	1.918	2.341	2.894	1.149080	0.824440	0.920043	Gaussian noise with random σ
big cnn stack	2	1.984	1.986	2.949	1.235930	0.801683	0.911025	Salt and pepper

Table 21: Architecture comparison

Architecture	cameras	l2 mean	l2 std	abs error in deg	abs error x-axis in deg	abs error y-axis in deg	abs error z-axis in deg	noise
big cnn stack	4	2.050	2.931	2.964	1.369117	0.727285	0.867315	Gaussian noise with random σ
small cnn concat	2	2.156	2.310	3.297	1.269447	1.010970	1.016204	Gaussian noise with random σ
big cnn stack	2	2.279	2.670	3.451	1.422202	0.932590	1.095770	Without noise
big cnn stack	4	2.528	3.160	3.796	1.664863	1.041168	1.089774	Gaussian noise with random σ
big cnn stack	4	2.833	3.485	4.135	1.682344	1.044394	1.407766	Without noise
big cnn stack	4	2.934	2.782	4.488	1.604138	1.388525	1.495246	Without noise
big cnn stack	4	3.020	4.157	4.427	1.790868	1.070318	1.565666	Salt and pepper
big cnn stack	4	3.258	3.313	4.945	1.691487	1.493073	1.760780	Salt and pepper
small cnn stack	2	3.963	3.817	5.949	2.423030	1.557538	1.968013	Salt and pepper
small cnn stack	2	4.690	4.514	7.235	2.730592	2.213037	2.291365	Salt and pepper
small cnn stack	4	4.948	4.413	7.788	2.586793	2.404511	2.797141	Without noise
small cnn stack	2	5.265	4.603	8.197	2.759518	2.643323	2.794590	Without noise
small cnn stack	2	5.902	5.192	9.176	2.977848	2.701037	3.497493	Gaussian noise with random σ
small cnn stack	2	6.083	5.066	9.180	4.216236	2.219303	2.744897	Without noise
small cnn stack	4	6.769	5.352	10.407	3.675838	3.542308	3.189000	Without noise
small cnn stack	4	6.883	5.187	10.682	3.996735	3.191781	3.493641	Salt and pepper

Table 21: Architecture comparison

Architecture	cameras	l2 mean	l2 std	abs error in deg	abs error x-axis in deg	abs error y-axis in deg	abs error z-axis in deg	noise
small cnn stack	2	7.443	5.738	11.293	4.501077	2.766679	4.025599	Gaussian noise with random σ
small cnn stack	4	8.537	6.125	13.203	5.138654	3.978835	4.085105	Gaussian noise with random σ
small cnn stack	4	8.736	5.876	13.615	4.506804	4.227186	4.881169	Salt and pepper
small cnn stack	4	9.016	5.675	14.065	4.182161	4.908478	4.974192	Gaussian noise with random σ

C.3 Rotation Estimation with Turntable Dataset

Table 22: All networks on the turntable dataset with different light influences

Architecture	Exp. regularized	l2 mean	l2 std	abs error in deg	abs error z-axis in deg	max abs error z-axis in deg	Noise
VGG16	False	0.323	0.328	0.356	0.318523	4.265930	Without noise
VGG16	False	0.398	0.425	0.432	0.395210	7.267123	Without noise
VGG16	True	0.431	0.304	0.475	0.426949	3.327395	Without noise
PoseCNN	False	0.445	0.354	0.484	0.441381	2.897656	Gaussian noise with $\sigma_0 = 0.01$
PoseCNN	True	0.433	0.351	0.485	0.426129	1.733418	Salt and Pepper
VGG16	False	0.443	0.406	0.486	0.438056	3.981858	Salt and Pepper
ResNet50	False	1.792	21.363	0.565	0.511142	5.945046	Salt and Pepper
VGG16	True	0.537	1.003	0.566	0.534031	16.901430	Salt and Pepper
CNN	True	0.565	1.218	0.574	0.564552	26.469033	Salt and Pepper
PoseCNN	True	1.160	15.128	0.576	0.517903	22.047687	Without noise
Mobilenet	False	1.164	15.122	0.582	0.520402	3.349986	Without noise
PoseCNN	False	0.536	0.495	0.598	0.528209	3.259047	Salt and Pepper
VGG16	True	0.513	0.658	0.604	0.495623	12.699728	Gaussian noise with $\sigma_0 = 0.01$
ResNet50	True	0.621	0.457	0.666	0.617848	4.015375	Gaussian noise with $\sigma_0 = 0.01$
CNN	True	0.676	1.136	0.699	0.674993	22.190703	Without noise
ResNet50	True	0.590	0.405	0.722	0.562886	2.916150	Salt and Pepper
PoseCNN	True	1.262	15.101	0.726	0.610545	20.385117	Gaussian noise with $\sigma_0 = 0.01$
PoseCNN	True	1.282	15.097	0.729	0.635871	8.010049	Salt and Pepper

Table 22: All networks on the turntable dataset with different light influences

Architecture	Exp. regularized	l2 mean	l2 std	abs error in deg	abs error z-axis in deg	max abs error z-axis in deg	Noise
VGG16	True	1.241	15.156	0.754	0.558614	18.080773	Without noise
CNN	False	1.378	15.124	0.761	0.740936	11.988791	Gaussian noise with $\sigma_0 = 0.01$
VGG16	False	2.002	21.296	0.789	0.729070	3.593137	Salt and Pepper
Mobilenet	False	3.281	29.939	0.827	0.758592	4.947852	Without noise
ResNet50	True	0.677	0.475	0.842	0.643921	5.193447	Salt and Pepper
PoseCNN	False	1.382	15.095	0.842	0.734329	4.342463	Without noise
VGG mod	True	0.838	0.669	0.884	0.834031	8.786044	Gaussian noise with $\sigma_0 = 0.01$
ResNet50	False	0.751	0.608	0.893	0.715558	4.800715	Gaussian noise with $\sigma_0 = 0.01$
Mobilenet	True	0.784	0.667	0.896	0.764191	8.095044	Without noise
CNN	True	1.524	15.166	0.912	0.885801	29.079274	Salt and Pepper
ResNet50	True	1.435	15.125	0.922	0.779866	9.685273	Without noise
Mobilenet	True	1.529	15.085	0.925	0.891476	3.521778	Salt and Pepper
ResNet50	False	2.115	21.365	0.955	0.819826	22.144181	Without noise
ResNet50	False	2.124	21.334	0.974	0.830381	4.719058	Without noise
Mobilenet V2	True	1.556	15.083	0.986	0.915967	3.312228	Without noise
Mobilenet	False	0.956	0.778	1.044	0.947564	3.599771	Salt and Pepper
CNN	True	1.667	15.124	1.048	1.031974	15.409757	Gaussian noise with $\sigma_0 = 0.01$
CNN	True	1.678	15.137	1.065	1.039573	14.326541	Gaussian noise with $\sigma_0 = 0.01$
VGG16	True	0.943	0.787	1.074	0.919818	6.627192	Salt and Pepper
CNN	False	2.975	26.112	1.079	1.069306	34.558393	Without noise

Table 22: All networks on the turntable dataset with different light influences

Architecture	Exp. regularized	l2 mean	l2 std	abs error in deg	abs error z-axis in deg	max abs error z-axis in deg	Noise
CNN	False	2.373	21.309	1.127	1.104121	19.235478	Without noise
ResNet50	False	1.714	15.116	1.142	1.071028	10.881138	Salt and Pepper
CNN	True	1.777	15.185	1.161	1.138510	19.593371	Without noise
PoseCNN	False	4.932	36.690	1.187	1.136623	30.385666	Salt and Pepper
CNN	False	2.317	19.681	1.272	1.250596	59.837289	Salt and Pepper
Mobilenet V2	False	2.589	21.321	1.438	1.296836	4.950887	Salt and Pepper
Mobilenet	True	1.873	15.109	1.439	1.203600	7.521288	Salt and Pepper
Mobilenet	True	2.607	21.286	1.467	1.319825	5.233228	Without noise
PoseCNN	True	5.162	36.380	1.506	1.420119	18.857074	Without noise
VGG mod	False	2.121	15.168	1.510	1.482839	27.081573	Without noise
VGG mod	True	2.561	20.075	1.510	1.440211	45.435270	Gaussian noise with $\sigma_0 = 0.01$
Mobilenet	False	2.640	21.657	1.559	1.326209	88.574532	Gaussian noise with $\sigma_0 = 0.01$
Mobilenet	True	2.048	15.110	1.577	1.383307	11.705015	Gaussian noise with $\sigma_0 = 0.01$
VGG mod	True	2.824	21.181	1.657	1.562323	12.432411	Without noise
VGG16	False	1.298	1.015	1.693	1.171008	8.043518	Gaussian noise with $\sigma_0 = 0.01$
VGG mod	False	3.211	23.966	1.699	1.643505	97.171264	Salt and Pepper
VGG mod	False	2.899	21.319	1.700	1.620632	5.643693	Gaussian noise with $\sigma_0 = 0.01$
VGG mod	False	3.323	24.403	1.716	1.675989	74.261746	Salt and Pepper
ResNet50	True	1.418	0.990	1.757	1.339825	8.186729	Without noise
ResNet50	True	1.385	0.646	1.961	1.127465	8.462208	Gaussian noise with $\sigma_0 = 0.01$

Table 22: All networks on the turntable dataset with different light influences

Architecture	Exp. regularized	l2 mean	l2 std	abs error in deg	abs error z-axis in deg	max abs error z-axis in deg	Noise
Mobilenet	True	1.897	2.370	2.016	1.890517	47.660725	Gaussian noise with $\sigma_0 = 0.01$
Mobilenet V2	True	2.452	15.090	2.079	1.787304	5.820090	Salt and Pepper
VGG mod	True	2.679	15.161	2.109	2.042010	36.982910	Salt and Pepper
VGG mod	False	3.412	21.337	2.187	2.141776	32.678224	Without noise
PoseCNN	True	4.618	29.840	2.203	2.108669	34.565343	Gaussian noise with $\sigma_0 = 0.01$
VGG mod	True	3.267	19.441	2.252	2.231828	68.656875	Salt and Pepper
ResNet50	False	3.098	21.251	2.333	1.715448	4.620199	Gaussian noise with $\sigma_0 = 0.01$
PoseCNN	False	2.790	15.971	2.393	2.123521	121.505809	Gaussian noise with $\sigma_0 = 0.01$
VGG mod	True	3.624	21.292	2.397	2.349444	22.766761	Without noise
Mobilenet V2	False	4.811	29.753	2.464	2.299008	10.332997	Without noise
VGG mod	False	5.663	29.966	3.191	3.135002	28.936809	Gaussian noise with $\sigma_0 = 0.01$
PoseCNN	False	7.876	41.210	3.360	3.114431	130.726313	Without noise
CNN	False	6.644	29.001	4.375	4.320896	71.982258	Salt and Pepper
Mobilenet V2	True	10.294	45.789	4.474	4.232308	42.898543	Salt and Pepper
Mobilenet V2	True	8.577	39.029	4.795	4.130407	15.542050	Without noise
Mobilenet V2	False	9.672	38.937	5.484	5.315211	18.116595	Without noise
Mobilenet V2	False	5.752	15.245	5.786	5.016416	14.184236	Salt and Pepper
CNN	False	10.551	40.576	5.928	5.843176	78.972630	Gaussian noise with $\sigma_0 = 0.01$
Mobilenet V2	False	9.761	36.389	6.301	5.997333	78.168771	Gaussian noise with $\sigma_0 = 0.01$
Mobilenet	False	10.671	43.925	6.675	4.577835	15.865588	Salt and Pepper

Table 22: All networks on the turntable dataset with different light influences

Architecture	Exp. regularized	l2 mean	l2 std	abs error in deg	abs error z-axis in deg	max abs error z-axis in deg	Noise
Mobilenet V2	True	26.395	71.048	11.431	11.267886	177.171385	Gaussian noise with $\sigma_0 = 0.01$
Mobilenet V2	True	18.964	51.209	12.232	11.120258	40.151294	Gaussian noise with $\sigma_0 = 0.01$
Mobilenet V2	False	16.979	44.672	12.421	11.366558	125.736694	Gaussian noise with $\sigma_0 = 0.01$
Mobilenet	False	21.864	47.017	16.991	15.628292	74.939426	Gaussian noise with $\sigma_0 = 0.01$
VGG16	False	182.808	104.607	88.788	88.778576	179.526625	Gaussian noise with $\sigma_0 = 0.01$
VGG16	True	183.108	104.613	88.839	88.763999	179.829498	Gaussian noise with $\sigma_0 = 0.01$

C.4 Accuracy Experiment with Different Gaussian Noise

In these experiments, we use the same setting as in Section 3.4. The differences in accuracy of the variants are so small that they can be neglected here. First, we consider the train accuracy of the different networks. Here, only variants are tested internally.

Table 23: Internal tests

Noise	MSE
'clipping'	0.00046
'reflect'	0.00049
'scale'	0.00044
'without edges'	0.00042

As already written above, the values hardly differ. In addition, we tested how the networks react to new noise variants. For this purpose, we tested each network with each noise variant. Furthermore, the noise variable σ for the test set was randomly and independently chosen between 0.0 and 0.75 for each image. We were able to determine the following table, which is sorted in ascending order in the MSE. Once again, we see

Table 24: Tests with unknown noise

Train noise	Test noise	MSE
'without edges'	'clipping'	0.171255
'without edges'	'reflect'	0.189909
'scale'	'scale'	0.190595
'reflect'	'clipping'	0.206997
'reflect'	'reflect'	0.208079
'clipping'	'clipping'	0.208206
'without edges'	'without edges'	0.211536
'clipping'	'reflect'	0.211801
'clipping'	'scale'	0.214411
'scale'	'clipping'	0.219263
'scale'	'reflect'	0.221312
'clipping'	'without edges'	0.229679
'reflect'	'without edges'	0.231358
'scale'	'without edges'	0.232638
'without edges'	'scale'	0.236858
'reflect'	'scale'	0.360751

that all values are close to each other. Only the last combination has a stronger increase

in the error. The best combination in which 'without edges' was trained and 'clipping' was tested can probably be explained by the fact that 'clipping' is included in 'without edges' and is therefore known. It is surprising that the variants with the same train and test noise did not perform best. However, this is probably also an indication of how close the results are. The worst combinations are dominated by 'scale' and 'without edges' as test noise. This probably comes from the lowered contrast in the images and the interference that is unknown for most networks. Nevertheless, it should be mentioned again here that it is hardly possible to make a statement about the different quality of the networks with such small separations.

C.5 Proof of Theorem 4.17 from [BCS18]

Proof. **(Proof of Theorem 4.17)**[23]
Let $\phi(t, x_0)$ be the solution of $x_{t+1} = f(x_k)$ at time t starting from x_0 at time $k = 0$ and $x_\star$ be an equilibrium point of the system. Let

$$V(x_0) = \sum_{t=0}^{N-1} (\phi(t, x_0) - x_\star)^\perp (\phi(t, x_0) - x_\star)$$

for some integer variable N to be set. Then

$$\begin{aligned} V(x_0) &= (x_0 - x_\star)^\perp (x_0 - x_\star) + \sum_{t=1}^{N-1} (\phi(t, x_0) - x_\star)^\perp (\phi(t, x_0) - x_\star) \\ &\geq (x_0 - x_\star)^\perp (x_0 - x_\star) = \|x_0 - x_\star\| \end{aligned}$$

and on the other hand, using (4.17) we have

$$\begin{aligned} V(x_0) &= \sum_{t=0}^{N-1} (x_t - x_\star)^\perp (x_t - x_\star) \\ &\leq \sum_{t=0}^{N-1} k^2 \|x_0 - x_\star\|^2 e^{-2\lambda t} \\ &\leq k^2 \left(\frac{1 - e^{-2\lambda N}}{1 - e^{-2\lambda}} \right) \|x_0 - x_\star\|^2 \end{aligned}$$

We have shown that there exists c_1 and c_2 such that

$$c_1 \|x_0 - x_\star\|^2 \leq V(x_0) \leq c_2 \|x_0 - x_\star\|^2$$

[23] This proof was published in [BCS18].
Note: In the proof of this theorem the norm denotes the 2-norm. The equivalence of norms then easily transfers the result to other norms.

is satisfied. Now, since $\phi(t, f(x_0)) = \phi(t, \phi(1, x_0)) = \phi(t+1, x_0)$,

$$
\begin{aligned}
&V(f(x)) - V(x) \\
=&\sum_{t=0}^{N-1} (\phi(t+1, x_0) - x_\star)^\perp (\phi(t+1, x_0) - x_\star) - \sum_{t=0}^{N-1} (\phi(t, x_0) - x_\star)^\perp (\phi(t, x_0) - x_\star) \\
=&\sum_{j=1}^{N} (\phi(j, x_0) - x_\star)^\perp (\phi(j, x_0) - x_\star) - \sum_{t=0}^{N-1} (\phi(t, x_0) - x_\star)^\perp (\phi(t, x_0) - x_\star) \\
=&(\phi(N, x_0) - x_\star)^\perp (\phi(N, x_0) - x_\star) - (x_0 - x_\star)^\perp (x_0 - x_\star) \\
\leq&k^2 e^{-2\lambda n} \|x_0 - x_\star\|^2 - \|x_0 - x_\star\|^2 \\
=&-\left(1 - k^2 e^{-2\lambda N}\right) \|x_0 - x_\star\|^2
\end{aligned}
$$

Now we choose N big enough so that $1 - k^2 e^{-2\lambda N}$ is greater than zero and also the second property has been proven. For the third property, since f is continuously differentiable it is also Lipschitz over the bounded domain D, with a Lipschitz constant L, for which it holds $\|f(x) - f(y)\| \leq L \|x - y\|$. Then

$$
\begin{aligned}
\|\phi(t+1, x_0) - \phi(t+1, y_0)\| =& \|f(\phi(t, x_0)) - f(\phi(t, y_0))\| \\
\leq& L \|\phi(t, x_0) - \phi(t, y_0)\|
\end{aligned}
$$

and by induction

$$
\|\phi(t, x_0) - \phi(t, y_0)\| \leq L^t \|x_0 - y_0\|
$$

Consider now the difference $|V(x_0) - V(y_0)|$

$$
\begin{aligned}
&= \Big| \sum_{t=1}^{N-1} \Big((\phi(t,x_0) - x_\star)^\perp (\phi(t,x_0) - x_\star) - (\phi(t,y_0) - x_\star)^\perp (\phi(t,y_0) - x_\star) \Big) \Big| \\
&= \Big| \sum_{t=0}^{N-1} \big((\phi(t,x_0) - x_\star)^\perp ((\phi(t,x_0) - x_\star) - (\phi(t,y_0) - x_\star)) \\
&\quad + (\phi(t,y_0) - x_\star)^\perp ((\phi(t,x_0) - x_\star) - (\phi(t,y_0) - x_\star)) \big) \Big| \\
&\leq \sum_{t=0}^{N-1} \big(\|\phi(t,x_0) - x_\star\| \, \|\phi(t,x_0) - \phi(t,y_0)\| + \|\phi(t,y_0) - x_\star\| \, \|\phi(t,x_0) - \phi(t,y_0)\| \big) \\
&\leq \sum_{t=0}^{N-1} \left(\|\phi(t,x_0) - x_\star\| + \|\phi(t,y_0) - x_\star\| \right) L^t \, \|x_0 - y_0\| \\
&\leq \left[\sum_{t=0}^{N-1} k e^{-\lambda t} L^t \right] \left(\|x_0 - x_\star\| + \|y_0 - x_\star\| \right) \|x_0 - y_0\| \\
&\leq c_4 \left(\|x_0 - x_\star\| + \|y_0 - x_\star\| \right) \|x_0 - y_0\|
\end{aligned}
$$

and so we have proven the last inequality. □

C.6 Experiments for SGD and RMSProp

To prove the validity of the inequalities in Table 14, we make the same experiments as in Subsection 4.4.2. We set the fixed hyperparameter $\beta = 0.1$ and iterate over $\epsilon \in \{10^{-2}, \cdots, 1\}$ and $\alpha \in \{0.01, \cdots, 1\}$. In Figure 54 on the left side we can see our predicted vertical border at $\alpha = 0.88$ and after a short yellow area – where the SGD also converge but our hyperparameter bounding is not fulfilled – we see the red area – SGD is not converging – at around $\alpha = 0.9$.

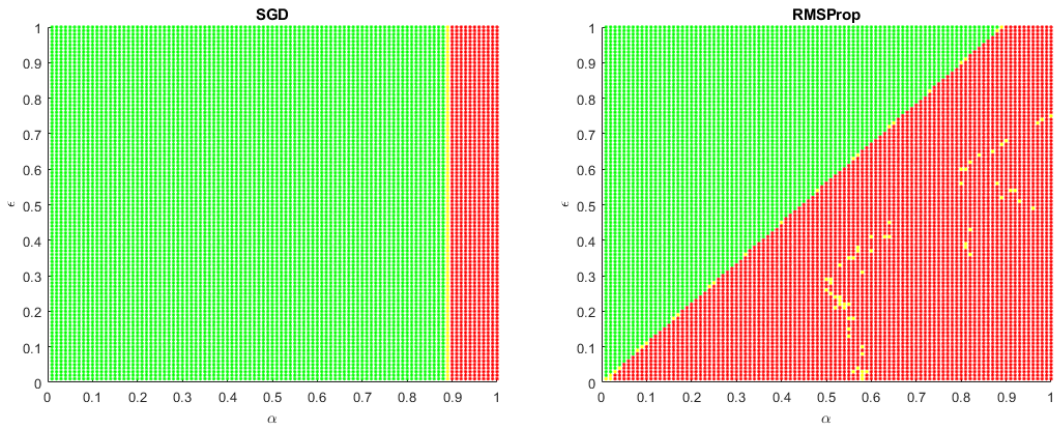


Figure 54: Iterating over ϵ and α.

In Figure 54 on the right side we can see a similar behavior. We can detect some yellow spots, where our inequality is not fulfilled but the RMSProp converges. However, there are no black areas where this is reversed. Denote that if we set x_0 far away from $x_\star$ we get some black area due to the fact that we only prove local convergence. In the experiments the SGD had a much smaller convergence area as RMSProp.

References

[AAB+16] Martin Abadi, Ashish Agarwal, Paul Barham, et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: *CoRR* abs/1603.04467 (2016).

[ANT00] Ravi P. Agarwal, Zuhair Nashed, and Earl Taft. *Difference Equations and Inequalities: Theory, Methods, and Applications*. 2nd ed. Chapman and Hall/CRC Pure and Applied Mathematics Ser. Boca Raton: Chapman and Hall/CRC, 2000. ISBN: 9780824790073. URL: https://ebookcentral.proquest.com/lib/gbv/detail.action?docID=5322044.

[AT21] Mohamed Akrout and Douglas Tweed. *On Bock's Conjecture Regarding the Adam Optimizer*. 2021. URL: http://arxiv.org/pdf/2111.08162v3.

[BCS18] Nicoletta Bof, Ruggero Carli, and Luca Schenato. *Lyapunov Theory for Discrete Time Systems*. 2018. URL: https://arxiv.org/abs/1809.05289.

[Bel+19] Mikhail Belkin et al. "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences* 116 (2019), pp. 15849–15854.

[Boc17] Sebastian Bock. "Rotationsermittlung von Bauteilen basierend auf neuronalen Netzen (unpublished)". M.Sc. thesis. Regensburg: Ostbayerische Technische Hochschule Regensburg, 2017.

[BSW24] Sebastian Bock, Philipp Schwarz, and Martin G. Weiß. "U-Shape Phenomenon with Gaussian Noise and Clipped Inputs". In: *Proceedings of Eighth International Congress on Information and Communication Technology*. Ed. by Xin-She Yang et al. Singapore: Springer Nature Singapore, 2024, pp. 569–579. ISBN: 978-981-99-3043-2.

[BW19a] Sebastian Bock and Martin G. Weiß. "A Proof of Local Convergence for the Adam Optimizer". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852239.

[BW19b] Sebastian Bock and Martin G. Weiß. "Non-Convergence and Limit Cycles in the Adam Optimizer". In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, 2019, pp. 232–243. ISBN: 978-3-030-30483-6. DOI: 10.1007/978-3-030-30484-3_20.

[BW20] Sebastian Bock and Martin G. Weiß. *Rotation Detection of Components with Convolutional Neural Networks*. 2020. URL: https://opus4.kobv.de/opus4-oth-regensburg/frontdoor/deliver/index/docId/412/file/Rotation_Detection_of_Components_with_CNNs.pdf.

[BW21] Sebastian Bock and Martin G. Weiß. *Local Convergence of Adaptive Gradient Descent Optimizers*. 2021. URL: https://arxiv.org/abs/2102.09804.

[BWG18] Sebastian Bock, Martin G. Weiß, and Josef Goppold. *An improvement of the convergence proof of the ADAM-Optimizer*. OTH Clusterkonferenz 2018, 2018. URL: https://arxiv.org/abs/1804.10587.

[Che+19] Xiangyi Chen et al. "On the Convergence of A Class of Adam-Type Algorithms for Non-Convex Optimization". In: *Proceedings of the 7th International Conference on Learning Representations*. New Orleans, LA, 2019.

[dG18] André Belotto da Silva and Maxime Gazeau. *A general system of differential equations to model first order adaptive algorithms*. 2018. URL: https://arxiv.org/pdf/1810.13108.

[DHS11] John C. Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12 (2011), pp. 2121–2159. URL: http://dl.acm.org/citation.cfm?id=2021068.

[Don+92] David L. Donoho et al. "Maximum Entropy and the Nearly Black Object". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 54.1 (1992), pp. 41–81. ISSN: 00359246. URL: http://www.jstor.org/stable/2345948.

[Doz16] Timothy Dozat. *Incorporating Nesterov Momentum into Adam*. Ed. by ICLR Workshop. 2016.

[DSW99] Christian Demant, Bernd Streicher-Abel, and Peter Waszkewitz. *Industrial Image Processing: Visual Quality Control in Manufacturing*. Berlin and Heidelberg: Springer, 1999. ISBN: 9783642585500. DOI: 10.1007/978-3-642-58550-0.

[Ela05] Saber Elaydi. *An Introduction to Difference Equations*. Third Edition. Undergraduate Texts in Mathematics. New York, NY: Springer Science+Business Media Inc, 2005. ISBN: 9780387230597. DOI: 10.1007/0-387-27602-5. URL: http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10229207.

[FDB15] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. "Image Orientation Estimation with Convolutional Networks". In: *Pattern Recognition*. Ed. by Juergen Gall, Peter Gehler, and Bastian Leibe. Cham: Springer International Publishing, 2015, pp. 368–378. ISBN: 978-3-319-24947-6.

[He+16] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[Hin+12a] Stefan Hinterstoisser et al. "Model Based Training, Detection and Pose Estimation of Texture-Less 3d Objects in Heavily Cluttered Scenes". In: *Proceedings of the 11th Asian Conference on Computer Vision - Volume Part I*. ACCV'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 548–562. ISBN: 9783642373305. DOI: 10.1007/978-3-642-37331-2_42.

[Hin+12b] Geoffrey Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. ISSN: 1053-5888. DOI: 10.1109/MSP.2012.2205597.

[How+17] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017).

[HRS16] Moritz Hardt, Ben Recht, and Yoram Singer. "Train faster generalize better: Stability of stochastic gradient descent". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 1225–1234. URL: http://proceedings.mlr.press/v48/hardt16.html.

[HSS12] Geoffrey E. Hinton, Nitish Srivastava, and Kevin Swersky. *Overview of mini-batch gradient descent (unpublished)*. Toronto, 2012. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (visited on 11/28/2018).

[HTF09] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of statistical learning: Data mining, inference, and prediction*. Second edition, corrected 7th printing. Springer Series in Statistics. New York: Springer, 2009. ISBN: 1282126741.

[igu22] igus GmbH. *robolink D — RL-D-20-A0202 Datenblatt*. 2022. URL: https://www.igus.de/product/19040?artNr=RL-D-20-A0202 (visited on 10/05/2022).

[Int22] Intel Corporation. *Intel RealSense Product Family D400 Series*. 2022. URL: https://www.intelrealsense.com/wp-content/uploads/2022/05/Intel-RealSense-D400-Series-Datasheet-April-2022.pdf?_ga=2.53437544.1906577651.1667209493-875038429.1664973457 (visited on 10/31/2022).

[Jam+17] Gareth James et al. *An introduction to statistical learning: With applications in R*. Corrected at 8th printing 2017. Springer texts in statistics. New York: Springer, 2017. ISBN: 9781461471387. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6312402.

[KB15] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A Method for Stochastic Optimization". In: *Proceedings of the 3rd International Conference on Learning Representations*. San Diego, CA, 2015.

[KH92] Anders Krogh and John Hertz. "A Simple Weight Decay Can Improve Generalization". In: *Advances in Neural Information Processing Systems*. Ed. by J. Moody, S. Hanson, and R. P. Lippmann. Vol. 4. Morgan-Kaufmann, 1992. URL: https://proceedings.neurips.cc/paper/1991/file/8eefcfdf5990e441f0fb6f3fad709e21-Paper.pdf.

[KP01] Walter G. Kelley and Allan C. Peterson. *Difference equations: An introduction with applications*. 2. ed. San Diego, Calif.: Harcourt/Academic Press, 2001. ISBN: 012403330X. URL: http://www.loc.gov/catdir/description/els033/99069847.html.

[Kra15] Ulrich Krause. *Positive Dynamical Systems in Discrete Time: Theory, Models, and Applications*. Vol. v.62. De Gruyter Studies in Mathematics. Berlin/-Boston: De Gruyter, 2015. ISBN: 9783110365696. DOI: 10.1515/9783110365696. URL: http://dx.doi.org/10.1515/9783110365696.

[Kra21] Philipp Kramer. "Genauigkeitsabschätzung für neuronale Netze (unpublished)". Bachelor Thesis. Regensburg: Ostbayerische Technische Hochschule Regensburg, 2021.

[LeC+89] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6795724.

[LH19] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

[Li+18] Shuxin Li et al. “Multiscale Rotated Bounding Box-Based Deep Learning Method for Detecting Ship Targets in Remote Sensing Images”. In: *Sensors (Basel, Switzerland)* 18.8 (2018). DOI: 10.3390/s18082702. (Visited on 01/21/2020).

[Liu+16] Zikun Liu et al. “Ship Rotated Bounding Box Space for Ship Extraction From High-Resolution Optical Satellite Images With Complex Backgrounds”. In: *IEEE Geoscience and Remote Sensing Letters* 13.8 (2016), pp. 1074–1078. ISSN: 1545-598X. DOI: 10.1109/LGRS.2016.2565705. (Visited on 01/21/2020).

[LS11] Ulrike von Luxburg and Bernhard Schölkopf. “Statistical Learning Theory: Models, Concepts, and Results”. In: *Handbook of the History of Logic, Vol. 10: Inductive Logic*. Vol. 10. Amsterdam, Netherlands: Elsevier North Holland, 2011, pp. 651–706. DOI: 10.1016/B978-0-444-52936-7.50016-1.

[Mah+22] Hamid Maher et al. *AI Is Essential for Solving the Climate Crisis*. Ed. by Boston Consulting Group. 2022. URL: https://www.bcg.com/publications/2022/how-ai-can-help-climate-change (visited on 09/05/2022).

[Mio+17] Riccardo Miotto et al. “Deep learning for healthcare: review, opportunities and challenges”. In: *Briefings in Bioinformatics* 19.6 (2017), pp. 1236–1246. ISSN: 1477-4054. DOI: 10.1093/bib/bbx044.

[MM19] Song Mei and Andrea Montanari. *The generalization error of random features regression: Precise asymptotics and double descent curve*. 2019. DOI: 10.48550/ARXIV.1908.05355. URL: https://arxiv.org/abs/1908.05355.

[MT11] Leigh S. McCue and Armin W. Troesch. “Use of Lyapunov Exponents to Predict Chaotic Vessel Motions”. In: *Contemporary ideas on ship stability and capsizing in waves*. Ed. by Marcelo Almeida Santos Neves. Vol. 97. Fluid Mechanics and Its Applications. Dordrecht: Springer, 2011, pp. 415–432. ISBN: 978-94-007-1481-6. DOI: 10.1007/978-94-007-1482-3_23.

[Orh22] Seymen Orhan. “Neuronale Netze zur Lageerkennung (unpublished)”. Bachelor Thesis. Regensburg: Ostbayerische Technische Hochschule Regensburg, 2022.

[Pas+19] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc, 2019.

[RHG13] Rüdiger Reinhardt, Armin Hoffmann, and Tobias Gerlach. *Nichtlineare Optimierung: Theorie Numerik und Experimente*. Berlin: Springer Spektrum, 2013. ISBN: 9783827429483.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0.pdf.

[RKK19] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. "On the Convergence of Adam and Beyond". In: *CoRR* abs/1904.09237 (2019).

[RM51] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. ISSN: 2168-8990. DOI: 10.1214/aoms/1177729586. URL: https://projecteuclid.org/download/pdf_1/euclid.aoms/1177729586.

[Rud16] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016).

[San+18] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[Sta21] Andreas Stautner. "Support Vector Regression (unpublished)". Bachelor Thesis. Regensburg: Ostbayerische Technische Hochschule Regensburg, 2021.

[SWW] William Silver, Aaron Wallack, and Adam Wagman. "Fast high-accuracy multi-dimensional pattern inspection". US6975764B1. URL: https://patents.google.com/patent/US6975764B1/en?q=Cognex&before=priority:19971231&after=priority:19970101&oq=Cognex+1997.

[SZ15] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2015).

[Wol+85] Alan Wolf et al. "Determining Lyapunov exponents from a time series". In: *Physica D: Nonlinear Phenomena* 16.3 (1985), pp. 285–317. ISSN: 01672789. DOI: 10.1016/0167-2789(85)90011-9.

[Xia+18] Yu Xiang et al. "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes". In: *Robotics: Science and Systems RSS*. 2018.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: (2017). URL: http://arxiv.org/pdf/1708.07747v2.

[Yud08] Eliezer Yudkowsky. *Artificial Intelligence as a Positive and Negative Factor in Global Risk.* Global Catastrophic Risks. Oxford: Oxford University Press, 2008. ISBN: 978-0199606504.

[Zei12] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *CoRR* abs/1212.5701 (2012).

[Zou+19] Fangyu Zou et al. "A Sufficient Condition for Convergences of Adam and RMSProp". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 11127–11135. DOI: 10.1109/CVPR.2019.01138. URL: http://openaccess.thecvf.com/content_CVPR_2019/html/Zou_A_Sufficient_Condition_for_Convergences_of_Adam_and_RMSProp_CVPR_2019_paper.html.

[ZSI19] Sergey Zakharov, Ivan S. Shugurov, and Slobodan Ilic. "DPOD: 6D Pose Object Detector and Refiner". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 1941–1950.