Submitted by
Philipp Ehrenmüller

Submitted at
Institute for Machine Learning

Supervisor
Prof. Sepp Hochreiter

Co-Supervisor
Dipl.-Ing. Pieter-Jan Hoedt

August 2020

Reproduction Study on Self-Normalizing Neural Networks

Bachelor Thesis
to confer the academic degree of
Bachelor of Science
in the Bachelor's Program
Informatik

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Contents

Abstract

Machine learning has become an increasingly important part of research in computer science in recent years. With the help of machine learning, one could solve various problems that were impossible to solve before.
In this bachelor thesis, we are specifically working with neural networks. We will deal with the paper "Self-Normalizing Neural Networks". This paper introduces, the self-normalizing neural network (SNN) as well as the SELU activation function. Besides, this paper also provided a performance comparison between the self-normalizing neural network (SNN) architecture and other machine learning techniques.

Our goal is to reproduce the performance results of the self-normalizing neural network. For this purpose, we are conducting a reproduction study, in which we will focus explicitly on the datasets of the UCI machine learning repository.

The results of our reproduction study show that very performant machine learning models could be trained by using a self-normalizing neural network. Through our reproduction study, we could verify the performance results presented by Klambauer et al. We also accomplished to improve the performance on several datasets using a self-normalizing neural network (SNN).

1 Introduction

Nowadays, deep learning is a widely used technique in the field of machine learning. It allows computers to learn from experience and understand the world of hierarchical representations [29]. With Neural networks, it is possible to create hierarchical representations with increasing abstraction by combining several layers [29].

However, during the training process, neural networks with many layers often suffer from several problems like the vanishing or exploding gradient [23]. These problems could occur when the neural network uses gradient-based learning techniques for weight optimization. A way to solve these problems is by applying batch normalization [23].

Today, with the development of better weight initialization techniques and new activation function, there are alternatives to batch normalization. These alternatives are neural networks with self-normalizing properties [3]. That means a neural network is constructed so that the activations have a zero mean and a unitary variance compared to the input data [3]. Such a neural network that has these properties is called a self-normalizing neural network (SNN).

1.1 Research Description

The main objective of our bachelor thesis is to empirically verify the performance of the self-normalizing neural network (SNN) machine learning algorithm. The self-normalizing neural network (SNN) was introduced in the paper "Self-Normalizing Neural Networks". Since the performance analysis by Klambauer et al. [3] is very extensive, we have decided to limit our reproduction study to their analysis of the UCI machine learning repository datasets [3].

1.2 Goals

In order to perform the reproduction study, we are implementing a program that makes it possible to perform hyperparameter optimization on multiple datasets. This program should have the ability to save all results that one needs to carry out a hyperparameter optimization. To verify the performance results of the self-normalizing neural network (SNN) we carry out a performance analysis on 121 UCI machine learning repository datasets. We will then use our results to compare them with the results from the paper "Self-Normalizing Neural Networks".

2 Machine Learning

Machine learning refers to a field of studying computer algorithms, that focus on the development of computer applications that are able to use data and learn from it [1]. Machine Learning in general could be considered as an subset of artificial intelligence and is closely related to the study of mathematical optimization as well as computational statistics.
The term machine learning was coined by Arthur L. Samuel in the year 1959 [2]. By dealing with machine learning algorithms. In his paper on machine learning algorithms, he was able to verify that computers could be programmed to play checkers better than the programmer himself [2].

There are several applications in machine learning. Probably one of the biggest is data mining. Analyzing data may be associated with various problems. When humans analyze data errors may occur. This happens when trying to create relationships between multiple features. Since errors may occur in data analysis, one could conclude that this could lead to even more errors [5]. With machine learning, we could avoid such errors and at the same time improve the efficiency of our system.

2.1 General Concepts in Machine Learning

In the following chapter, one gets explained the basic concepts of machine learning that one should understand. We try to explain these concepts as simply and precisely as possible.

2.1.1 Accuracy

For someone to determine whether our current machine learning model works, we need some metric that reflects the current performance of our machine learning model. Such a metric could be accuracy. Accuracy indicates the relationship between the correct prediction and the total amount of tested instances, in this case, we could also speak of classification accuracy [15].

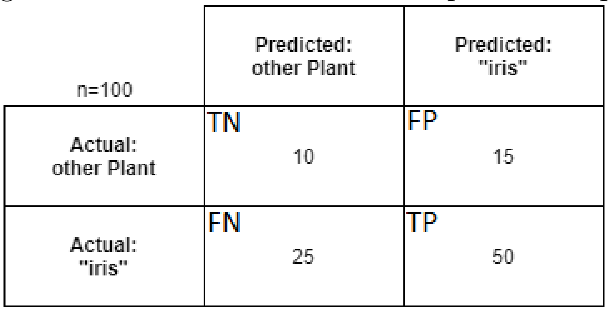
One could understand how the accuracy metric could be interpreted by looking at an example with different plant species. An example of this would be determining a plant species. One could assume that we have 100 photos of plants and want to know how many of these photos show the plant "iris". 60 out of 100 images are correctly determined. This gives us an accuracy of 60%. In this example only correctly predicted images are taken into account.

To create an even more precise overview of the complete performance of our machine learning model, we could set up a so-called confusion matrix. The confusion matrix shows correctly classified instances and also incorrectly classified ones [16]. The following diagram illustrates how such a confusion matrix may be structured:

For the confusion matrix there are four different terms we need to know:

- **True Positives:** The cases in which we predicted "iris" and the actual output was also "iris".

Figure 1: Confusion Matrix of the plant example



n=100	Predicted: other Plant	Predicted: "iris"
Actual: other Plant	TN 10	FP 15
Actual: "iris"	FN 25	TP 50

- **True Negatives:** The cases in which we predicted "other Plant" and the actual output was "other Plant".
- **False Positives:** The cases in which we predicted "iris" and the actual output was "other Plant".
- **False Negatives:** The cases in which we predicted "other Plant" and the actual output was "iris".

To calculate the accuracy we could sum up the values lying across the diagonal and divide them by the total number of tested instances:

$$Accuracy = \frac{FalseNegatives + TruePositives}{TotalAmountOfSamples}$$
$$Accuracy = \frac{10 + 50}{100} = 0.6$$

2.1.2 Generalization

Generalization is an essential ability for any learning system. It could be described as the observation of a set of training examples and identifying essential features common to this training examples [9].
As an example of generalization, one could show a picture of a cat and one wants to know if the animal on the image could be "classified" as a cat. Assuming it is classified correctly as a cat one could alter the image by moving, rotating or mirroring the image, a human would still be able to identify the animal on the picture as a cat. With this process, one could create extra data that might be useful during the training process.

Generalization of Information offers several benefits. It is easily adaptable and is not dependent on any kind of machine learning technique therefore it could be applied toward all existing data. Generalization techniques are easy to apply since they are based on basic information theory mathematics [10].

2.1.3 Pattern Classification

Pattern classification is a learning procedure which is used in machine learning. Many algorithms in pattern classification are probabilistic. They use statistical interference to determine the best class for a given input.

The goal of pattern classification is to assign a specific class based on a certain input. In other words, we want to assign different data records to a specific class/group. We could assume that we have a certain number of inputs $x_1, x_2 ... x_n$. These inputs could also be called features. As output, we would have one respective class of N classes which could be represented as a finite number [14].

The use of pattern classification requires a selection of different features. These features should have properties that describe the object we want to classify. Only if this requirement is fulfilled a correct differentiation/classification between the data records is possible.

Pattern classification could not be used optimally in every area of application. If pattern classification is used in areas such as speech, vision, robotics or artificial intelligence, problems arise because complex real-world data is used in these application areas. Real-time response is therefore required in these areas [14].

2.1.4 Validation

Simply developing a machine learning model is not enough so that one could rely on the model's predictions. For this reason, one has to check whether the model performs similarly with unknown data as with the data that we used for training.
In machine learning, there are various validation techniques that one could use to determine the performance of our machine learning models. Choosing the right validation method is particularly important because one wants to rely on the result of this validation. For this reason, we would like to present two different validation methods in the following sections that are often used in machine learning.

2.1.4.1 Hold-out Validation

Holdout set validation is one of the best known and at the same time one of the simplest of the validation methods. Here one uses the training data and splits it into a train and a test set [18]. Now one could train the machine learning model with the train set and then check the performance of the model with the test set. Based on this check, one could determine the performance of our model. This method usually involves an 80% train and 20% test split, but this may vary.

2.1.4.2 Cross-Validation

Cross-validation is a tool for assessing the performance of generalization quantifier. This quantifier sorts data into different classes based on certain criteria [34]. Cross-validation is then used to predict the performance of the quantifier, which is based on a training dataset. This set is then applied to future cases. The data is split into k equally sized sets. Then $k - 1$ subsets are used as a training set to calculate quantifiers and the remaining subset is then used to calculate the quantifiers performance [24]. The process is repeated k times. Every time another subset is used as test set. The total average performance is then used to evaluate the quantifier. Cross-validation also prevents/avoids the model from over/under-fitting by estimating the optimal quantifier [24].

2.1.5 Overfitting

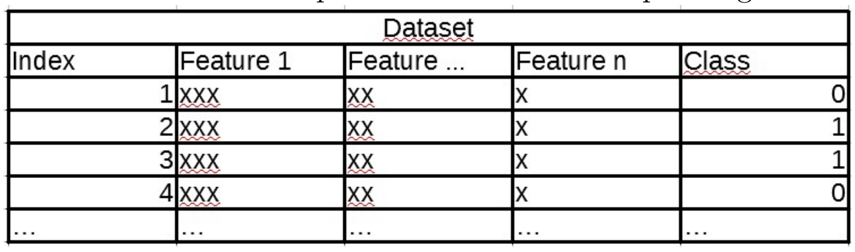
Overfitting means that a model will produce predictions that are optimal for the training data but using unseen data might lead to bad predictions [11]. The problem here is if one uses a data which the trained model has never seen before and then lets the model predict these data, the accuracy usually turns out much worse than with the training data.

One could assume that the machine learning model predicts 95% of the training data correctly.But the model predicts only 60% of the test data correct. This may be a sign that the trained machine learning model is overfitting on the dataset. Therefore the trained model is not reliable when one wants to predict new data.

2.2 Supervised Learning

Supervised learning is one of the three major areas of machine learning. In supervised learning, we focus on teaching a machine learning model how it should react using sample data. We call the reaction of the machine learning model to such sample data prediction. This sample data consists of different features, based on these features the data may be assigned to a predefined class [24]. The features of such sample data could be either continuous, categorical or binary.

Figure 2: Instances of sample data with the corresponding class label



Dataset				
Index	Feature 1	Feature ...	Feature n	Class
1	xxx	xx	x	0
2	xxx	xx	x	1
3	xxx	xx	x	1
4	xxx	xx	x	0
...	...	...	...	...

Supervised machine learning may be divided into the tasks of classification, regression and logistic regression [24]. In this section, we will go into the details about the specific differences between the different tasks. We would like to mention that these techniques work with a wide variety of machine learning algorithms, but we refer here specifically to the use of these techniques in relation to artificial neural network (ANN).

Which of these three techniques we need to apply depends entirely on the dataset we use, then depending on what our goal for this data is we need to adapt the artificial neural network (ANN) to it.

We have to adjust the activation function of the output layer because depending on the task, we may or may not change our output values. The second parameter we want to modify is the number of neurons in the output layer. This depends on the data we use and if we would like to predict a single value or multiple values. Another option would be if one wants to predict the probability of different classes. In this case, one would need to adjust the number of neurons in the output layer to the total amount of different classes.

2.2.1 Classification

Classification is a technique used in supervised machine learning to train a machine learning algorithm using existing data. As we have already described in the section supervised machine learning, this data consists of various features. Based on these features, we train the machine learning algorithm so that the algorithm could predict a previously determined target value [24].

We may differentiate between binary and multi-class classifier. In binary classification, our machine learning model is limited to two possible answers. These answers could be either state whether something is within the target class or not (example: positive or negative sentiment; whether a lender will pay the loan or not; etc.) [24].
A multi-class classifier has more than two target classes (example: whether an image is an apple or banana or orange). We can assume that each sample is assigned to exactly one target class [24].

2.2.2 Regression

Regression is another important area in supervised machine learning. Regression is relatively similar to classification. We do not want to determine a specific class in regression we want our machine learning model to predict a real number [24].

Assume that someone wants to predict the delay a specific train will have on a route before this train arrives at the particular railway station. Another prominent example of regression in machine learning would be predicting stock prices.

The regression model establishes a connection between the input dimensions and the output dimension. If we want to build a machine learning model for regression we have to make sure that we do not falsify our result, therefore we use a linear activation function in the output layer of the machine learning model.

One major characteristic of regression is that it is impossible to determine the accuracy of the machine learning model. In regression the model will probably never predict the correct value given in the training dataset it will only give us a number which will be relatively close to training value.

Since one could not determine how many of the training samples were classified correctly like in classification one needs to use something else. Instead of using accuracy to measure the model's performance, we use a so-called loss function (e.g. mean squared error (MSE)) to determine the performance of our regression model [24].

2.2.3 Logistic regression

Logistic regression is a machine learning technique that could be applied to classification problems. This technique does not directly predict a specific class. Instead, it will give us a probability that indicates which class is most likely one [24].

An example of such a problem would be to find out whether an email is a spam email or not. As one could see from this example logistic regression is a binary classification technique, this means that as we mentioned already in section 2.2.1, it is not possible to solve multi-class problems with logistic regression.

2.3 Unsupervised Learning

Unsupervised machine learning could be seen as the opposite of supervised machine learning. In unsupervised machine learning, we use unlabeled data, that means one does not know anything about the data behind the respective feature. One neither knows anything about if there exists a target class or how many target classes exist. One also does not know the differences between these target classes [24].

The main goal of unsupervised machine learning is to find relationships or patterns in the data. Various computer algorithms such as clustering are used in unsupervised machine learning [12]. In clustering, we want to group the dataset records in different clusters. We are using various mathematical techniques, like pattern classification (section 2.1.3) or different statistical methods.

An example of such a cluster algorithm that could be used in unsupervised machine learning is the k-means algorithm. The k-means algorithm is a method for vector quantization that puts all of the data into several clusters. The algorithm now tries to minimize its variance by shifting the centre of each cluster. When shifting the centre of each cluster the distance between the data records and the centre also changes [13]. That means that some data records might be assigned to another cluster in the following iteration when there exists a cluster which is closer to the data record than the current cluster.
To calculate the distance between these two points one could use the Euclidean distance:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{1}$$

The distance d between two vectors $pandq$ where p is the vector of the data record and q is the vector of the cluster is calculated by summing the squared difference of the vectors p and q. We need to mention one could also use other functions to calculate the distance e.g. Manhattan distance.

To shift the center of each cluster we are using the following formula:

$$m_i = \frac{1}{|S_i|} * \sum_{p_j \in S_i} p_j \tag{2}$$

We calculate the centre m_i of a cluster by calculating the mean vector of all data records S_j that are assigned to a cluster.

2.4 Deep Learning

Deep learning allows computers to learn from experience and understand the world in the form of a hierarchy of concepts. All of these concepts are defined by its relationship to simpler concepts [29]. In deep learning, this avoids having to formally specify the complete knowledge with which a machine learning algorithm learns. This hierarchy of concepts enables the machine learning algorithm to learn complicated concepts by assembling them from simpler ones [29].

The performance of a machine learning algorithm depends heavily on the representation of the data the algorithm uses to learn [29]. Any information used in the representation is also referred to as a feature. As we mentioned in the section classification 2.2.1, this information is used by in machine learning to make certain predictions. One should know that the machine learning algorithm does not influence how these features are defined.

Assume that logistic regression would be used for a magnetic resonance imaging (MRI) examination of a patient and no formalized report from the doctor was received. It would then be impossible for this algorithm to make a useful prediction since individual pixels in an MRI examination have a negligible correlation [29].

Representation learning provides a solution to this problem. Here, it is no longer just a matter of learning to map the representation (input) to the output, but the machine learning algorithm will also learn the representation itself [29]. Representation learning has the advantage that this technique could often achieve far better performance results than with self-designed features [29].

An advantage of representation learning is that it quickly to new tasks. For instance, it could discover a good set of features for simple tasks in minutes, for complex tasks in hours [29]. Manually designing features for a complex task, on the other hand, could take decades [29].

3 Neural Networks

A neural network is an association of neurons. These neurons are the smallest unit of a neural network. The term neuron originally comes from biology and describes a type of cell that occurs in the human brain. The first mathematical abstraction of a biological neuron was introduced by McCulloch and Pitts in the year 1943 [19]. This mathematical abstraction was the first step for neural networks in computer science.

Neural networks are currently a popular method in machine learning. One of the advantages of neural networks is that they have the ability to recognize complex nonlinear relationships between dependent and independent variables [20]. In this chapter, we deal with the construction of an artificial neural network (ANN) and how neural networks are functioning. We will also introduce different types of neural networks in this chapter.

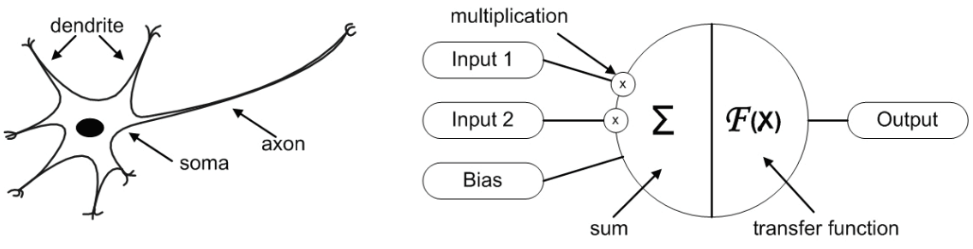
3.1 Neuron

As we mentioned earlier, a neuron is the smallest part of an artificial neural network (ANN). The artificial neuron (AN) itself is a mathematical abstraction of the biological neuron (BN). Like the biological neuron (BN), the artificial neuron (AN) consists of various components. In the image 3 we could see that a biological neuron (BN) consists of three different components [21].

The dendrites through which the neuron receives information. The body of the neuron itself called soma, that processes the incoming information. Lastly, the axon which passes the processed information on to the next neuron. For comparison, the artificial neuron (AN) that could be seen in figure 3 has a relatively similar design as the biological neuron (BN). Therefore we could describe the structure of an artificial neuron (AN) as follows:

The artificial neuron (AN) has 1 to n inputs, which corresponds to the dendrites of a biological neuron (BN). The transfer function which replicates the functionality of the soma and therefore processes the data obtained from the inputs. An output that corresponds to the axon that passes the processed data to its connected neurons. The artificial neuron (AN) has an extra component called bias, which we will discuss later on.

Figure 3: Biological and artificial neuron design [21].



Now that we have briefly explained the various components of an artificial neuron (AN), we would like to go into more detail about how it works. The artificial neuron (AN) receives data via the inputs. These inputs could be weighted. It is also possible to weight each input individually. The sum of the weighted inputs and the bias is then formed in the core of the artificial neuron (AN). This calculated sum is then passed to the transfer function. The return value of this transfer function is then sent to the output, which forwards the result to all other connected neurons. This process described here could also be described mathematically and looks like this [21]:

$$y(k) = F(\sum_{i=0}^{m} w_i(k) * x_i(k) + b) \tag{3}$$

Where:

- $x_i(k)$ is input value in discrete time k where i goes from 0 to m,
- $w_i(k)$ is weight value in discrete time k where i goes from 0 to m,
- b is bias,
- F is a transfer function,
- $y_i(k)$ is output value in discrete time k.

The only major unknown that we haven't covered in detail yet is the transfer function. The transfer function is responsible for ensuring that the artificial neuron gets its properties. The transfer function indicates how the artificial neuron will change the result of the precalculations [21].

We need to mention that the term transfer function is a rather ancient term. The term transfer function was often used earlier in the 80s and 90s, nowadays one speaks of the activation function. For this reason, from now on we will only use the term activation function in the following chapters to avoid confusion. Since the choice of the correct activation function is crucial for the functioning of the neural network and has an impact on the result of the neural network, we would like to go into more detail in a later chapter on the various existing activation functions.

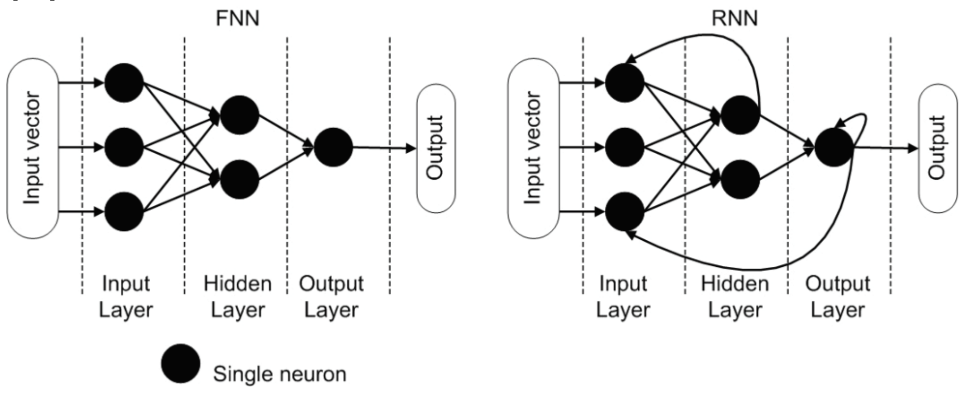
3.2 Artificial Neural Network (ANN)

If we interconnect several artificial neurons (ANs) with each other, we get an artificial neural network (ANN). The reason we are interconnecting those artificial neurons (ANs) is that the usefulness of a single neuron is limited to simple calculations. That means a single neuron could not solve complex real-life problems [21]. By interconnecting several neurons the network gains the ability to solve complex problems and problems for which there is no algorithm.

If we now connect several artificial neurons, we create a so-called topology. One could also speak of the architecture of the ANN. The architecture of an ANN is often graphically displayed using a directed graph.

The simple structure of the artificial neuron enables us to create many different architectures of an artificial neural network (ANN). We may divide these architectures into two groups. These two groups are differentiated by how the information flows through the neural network. If the information flows only in one direction we speak of feed-forward neural networks. If it flows in both directions we call them recurrent neural networks [21].

Figure 4: Comparison between the architecture of a feed-forward and a recurrent neural network [21].



3.2.1 Feed-Forward Neural Network (FNN)

An artificial neural network is called a feed-forward neural network if its architecture fulfils the following condition: The information flowing through the artificial neural network may only flow in one direction. That means that the artificial neural network must not include any back-loops in its architecture [21].

There are no other architecture restrictions. The network may contain any number of layers with a varying number of neurons. It is not relevant for the topology which activation function is used in the different layers and how the individual neurons are connected as long as there is no back-loop. A simple example of such a feed-forward neural network would be a neural network consisting of a single layer of output neurons.

3.2.2 Recurrent Neural Network (RNN)

The recurrent neural network is very similar to the feed-forward neural network. An artificial neural network is called recurrent if it has no back loop restrictions. As a result, the information flow is no longer tied to a specific direction, that means the information could now flow in both directions without any restrictions [21].

The most basic recurrent neural network is a fully connected artificial network in which every neuron interconnected to every other neuron in both directions.

3.3 Activation Function

As mentioned earlier in section 3.1, the activation function is an essential part of an artificial neural network. The activation function manipulates the artificial neuron's data, i.e. the weighted sum of the inputs and biases, through gradient processing, usually gradient descent [23]. The result of this calculation produces the output of the neurons. We could also say that the activation function is used to control the output of the neurons.

Activation functions could be either linear or non-linear, but this depends entirely on the mathematical function used as the activation function. In this section, we will take a closer look at some of these activation functions. We also want to mention that this list of activation functions is by no means complete.

3.3.1 Linear Function

The linear activation function linear is given as follows:

$$f(x) = x$$

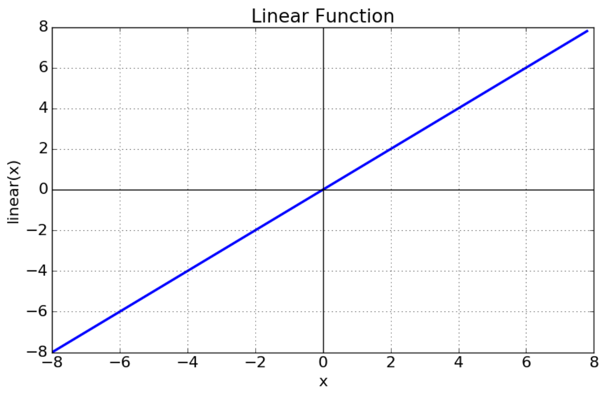
The linear activation function is also known as the identity function. This function will not change the given input in any way and is also not bound to any limit. The linear activation function is often used for regression tasks. For example when one trains a model that could predict the price of a house. The output, in this case, would be a real number.

3.3.2 Sigmoid Function

The sigmoid activation function is sometimes referred to as a logistic function or squashing function [23]. It is a non-linear activation function that is often used in feed-forward neural networks (FNNs). It is a bounded differentiable real function that is defined for real input values [23]. The formula for the sigmoid activation function is defined as follows:

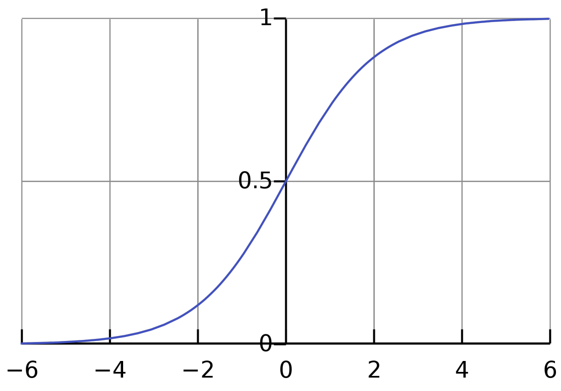
$$S(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

Figure 5: Linear function



We have to mention that the sigmoid activation function also has some disadvantages. These disadvantages include sharp damp gradients during backpropagation from deeper hidden layers to the input layers, gradient saturation, slow convergence and non-zero centred output [23].

Figure 6: Sigmoid function



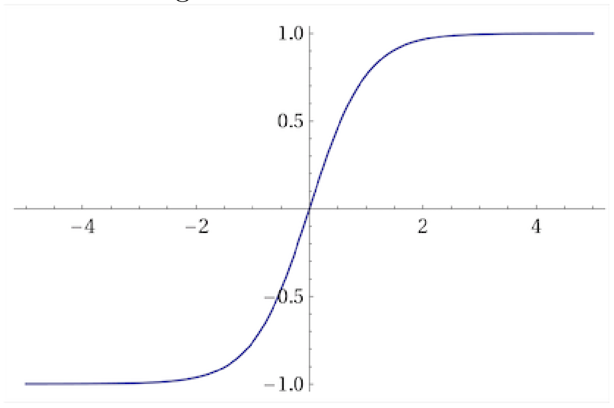
3.3.3 Hyperbolic Tangent Function (tanh)

The hyperbolic tangent function (tanh) is another type of activation function used in deep learning. The hyperbolic tangent function, is a smooth, zero-centred function in the range -1 to 1 [23]. Due to the zero-centred property of the tanh activation function it has an advantage in the backpropagation process. The formula for the tanh function is defined as follows:

$$f(x) = (\frac{e^x - e^{-x}}{e^x + e^{-x}}) \tag{5}$$

The tanh activation function is the preferred variant over the sigmoid activation function in the area of multi-layer neural networks because it offers better training performance there [23]. The tanh function does not manage to solve the vanishing gradient problem. This problem could also occur when using the sigmoid activation function.

Figure 7: tanh function



3.3.4 Rectified Linear Unit (ReLU) Function

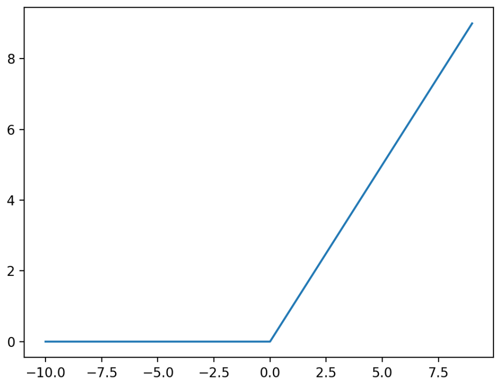
The ReLU activation function was first introduced by Nair and Hinton in 2010 [23]. Since then, the activation function has been widely used in the field of deep learning. The ReLU activation function offers better performance and better generalization in deep learning compared to the tanh and sigmoid activation functions.

The ReLU function represents an almost linear function, which means that it retains the property of simple optimization that linear models have when using gradient-descent methods for optimization [23]. The ReLU activation function performs a threshold operation for each input, where any value less than 0 is set to 0. The ReLU activation function is therefore determined by the following mathematical formula:

$$f(x) = max(0, x) = \left\{ \begin{array}{ll} 0 & \text{for } x < 0 \\ x & \text{for } x >= 0 \end{array} \right\} \tag{6}$$

This function rectifies the values of the inputs that are less than zero, by forcing them to zero. Thus, the vanishing gradient problem that occurs in the previous activation functions could be eliminated [23].

Figure 8: ReLU function



As we mentioned before, the ReLU activation function offers better performance compared to other activation functions. This is because the ReLU function does not have to calculate exponential divisions, which improves the overall calculation time. The ReLU function does not only offer advantages because the ReLU function has the unpleasant property that it could easily overfit compared to the sigmoid activation function [23]. For this reason, with the use of the ReLU activation function, the dropout technique is often used to counteract the effect of overfitting.

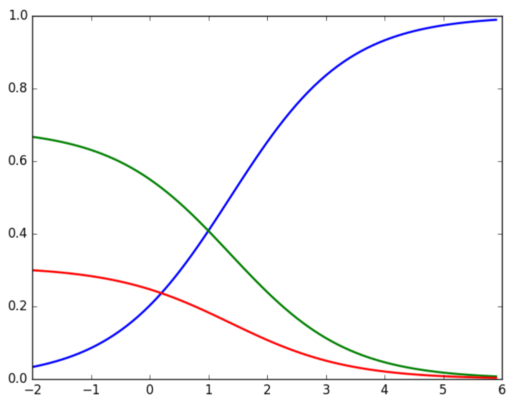
3.3.5 Softmax Function

The softmax activation function is used in neural networks to calculate the probability distributions for multiple classes [23]. It takes a real number vector K as input and normalizes the inputs into a probability distribution consisting out of K probabilities. Each probability value will be in the interval $(0, 1)$, where the sum of the individual output probabilities will sum up to 1. Therefore, the softmax function could be mathematically defined as follows:

$$f(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{K} exp(x_j)} \text{ for } i = 1, ..., K \text{ and } x = (x_1, ..., x_K) \in R^K \tag{7}$$

The softmax activation function is primarily used in the output layer of neural networks. If one would compare the sigmoid and the softmax activation function one would find out that the softmax function is used for multivariate classification tasks whereas the sigmoid function is used for binary classification tasks [23].

Figure 9: Softmax function: This diagram shows an example of the probability distribution over three different classes.



3.3.6 Scaled Exponential Linear Units (SELU) function

During the training of a neural network, the distribution of network activations changes with each training step, which could slow down the training [25]. If this distribution keeps changing during the training process one could speak of the internal covariate shift problem. Here the hidden layers try to adapt to the new distribution of the network activations which will effectively slow down the training of the neural network [25].

A possibility to counteract this problem is by using batch normalization which normalizes the inputs of the layers. With the help of the batch normalization, the inputs will have the standard normal distribution with zero mean and unit variance [25].

The scaled exponential linear unit (SELU) activation function was introduced in the paper "Self-Normalizing Neural Networks" by Klambauer et al. [3]. The SELU activation function has self-normalizing properties, so that the node activations remain centred around zero and have unitary variance [25]. This allows the SELU to counteract the internal covariate shift problem. One also should know that in comparison to batch normalization, the SELU activation function has a lower computational complexity [25].

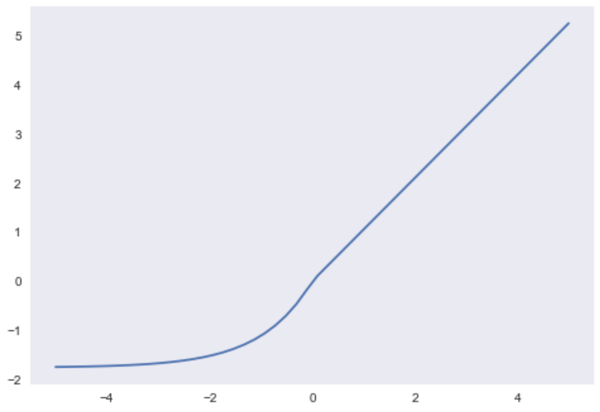
These properties make machine learning with the SELU activation function highly robust, and it is also possible to train neural networks that have many layers [3].

The SELU activation function is parameterized by $\alpha = 1.6733$ and $\lambda = 1.0507$ which control the mean and variance of the output distribution [3]. The parameter $\lambda > 1$ ensures a slope larger than one for positive inputs. The SELU activation function is defined as [3]:

$$selu(x) = \lambda \left\{ \begin{array}{ll} x & \text{if } x > 0 \\ \sigma e^x - \sigma & \text{if } x \leq 0 \end{array} \right\} \tag{8}$$

The SELU function changes its form from an exponential function for $x \leq 0$ to a linear form for $x > 0$ as seen in Figure 10.

Figure 10: SELU activation function



3.4 Optimization of Neural Networks

For an artificial neural network to learn, we have to update the weights associated with the connections of the neurons [27]. These weights need to be adjusted to minimize the differences between the actual and predicted results for subsequent training runs [24].

We could ask ourselves what would be needed to change the weights of a neuron so that the neuron could make a better prediction. Measuring the error between the predicted value and the actual value would be a good approach here [29]. The calculated error is typically carried backwards to a previous layer in neural networks, where it is used to modify the weights and bias to minimize the error. This process is also called backpropagation of the error.

If the neural network consisted of only a few neurons, the weights could also be adjusted manually. There are very few cases, where a neural network only consists of a few neurons.

When someone would adjust the weights of the neurons in a neural network with many neurons manually, this would be a bad decision because a normal human being would need to make unlogical guesses for the weights adjustments.

For this reason, a so-called loss function is used in neural networks. This loss function indicates how big the difference between the actual and the predicted value is. We could then optimize the weights of the neurons based on this loss function [28]. There are several functions we could use to calculate this error an example of such functions would be the mean squared error (MSE) or the mean absolute error (MAS).

It is our goal to adjust the weights so that we minimize the calculated error of the loss function [28], i.e. to determine the optimal value (minimum). When training neural networks, we have to distinguish whether the minimum found is a local or a global minimum.

3.4.1 Gradient Descent (GD)

The gradient descent (GD) is an optimization algorithm that is used to minimize the error. The idea of the gradient descent (GD) algorithm is that starting from a starting value, we move in every iterative step in the direction of the negative gradient since this gradient indicates the steepest descent [24]. That means each iterative step will gradually approach the minimum. The gradient descent (GD) optimizer is defined as follows:

$$w^{t+1} = w^t - \frac{\alpha}{n} \sum_{i=1}^{n} \Delta f_i(w^t) \tag{9}$$

Here α is defined as the step size (also known as the learning rate), t stands for the current iteration. The gradient descent algorithm always converges towards an optimal solution [24]. If the learning rate is too low, the optimizer needs more iterations to approach the minimum, since only a small step towards the minimum is made per iteration. If the learning rate is chosen too high, the minimum may be skipped.

A disadvantage of the gradient descent optimizer is the great computing effort since all instances of the dataset that were used during the current iteration are used to calculate the gradient for each iteration. We want to mention that gradient descent (GD) converges to an optimal point but, we do not know if this point is a local or a global minimum.

3.4.2 Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) is an optimization method that could be seen as a stochastic approximation of the gradient descent (GD) optimizer. This optimizer is called stochastic because it approximates the gradient by randomly selecting a single instance of the data at a time. This approximation allows us to update the classifier as new data comes in [24]. We need to mention that one could also randomly select a subset of the data to calculate the gradient.

With this calculated estimate, stochastic gradient descent (SGD) has the advantage that it reduces the computational effort, particularly in the case of high-dimensional optimization problems. In contrast to the gradient descent optimizer, the step direction is no longer the mean value of all gradients, but only the mean gradient from the randomly selected subset. That means that there is no guarantee that the machine learning model will approach the minimum with each iteration [24]. The stochastic gradient descent (SGD) could be mathematically defined as follows:

$$w^{t+1} = w^t - \alpha \Delta f_i(w^t) \tag{10}$$

The stochastic gradient descent (SGD) optimizer offers further parameters one could tune during the optimization process. One of these parameters is momentum. If one trains a neural network with high-dimensional input parameters it could happen that during the learning process a local minimum is found. To overcome such a local minimum one could apply momentum during the training process.

When stochastic gradient descent (SGD) uses momentum then the optimizer remembers the update Δw at each iteration and determines the next update with a linear combination of the current gradient and the previous update [26]. Also, we want to mention that it is not possible to apply momentum to every optimizer i.e gradient descent (GD) could not use momentum.

3.5 Dropout

Deep neural networks with a large number of parameters are powerful machine learning systems. Nevertheless, such networks have a serious problem with overfitting. The so-called dropout technique offers a solution to this problem [30].

The goal of dropout is to remove random neurons from the neural network during the training phase. Removing random neurons from the neural network prevents them from adjusting too much with each other [30]. The dropout technique reduces the effect of overfitting significantly and offers great improvements compared to other regularization methods.

One also should know that Srivastava et al. [30] showed that the dropout technique could not only be applied to feed-forward neural networks (FNNs) they additionally showed that this technique could also be used with graphic models like the Boltzmann machine [30].

3.6 Self-Normalizing Neural Network (SNN)

Neural networks like feed-forward neural network (FNN) lack the property of normalizing outputs. In such neural networks, this requires the use of an additional layer like batch normalization for normalizing hidden layer outputs [31]. To avoid having to use batch normalization between the hidden layers Klambauer et al. [3] have introduced a new kind of neural network called self-normalizing neural network (SNN). This self-normalizing neural network (SNN) possesses the ability that the activations of the neurons automatically converge to a fixed point at zero mean and unit variance for normalized weights [3].

Such an SNN possesses a mapping $g : \Omega \to \Omega$ for each activation y that map mean and variance from one to the next layer [3]. Therefore self-normalizing neural networks (SNNs) keep normalization of activations when propagating them through the layers of the network.

This normalization effect could be observed across the layers of the neural network.

In comparison, a feed-forward neural network (FNN) suffers from high variance when trained with normalization techniques. The self-normalizing neural network (SNN) allows us to train deep neural networks with many layers, employ strong regularization schemes. Due to the special properties of a self-normalizing neural network (SNN) the training is highly robust [3].

To use the self-normalizing neural network (SNN) correctly one needs to make sure that the dataset is standardized as we will describe later in section 5.2. That means that the mean of the dataset needs to be 0 and the standard deviation is 1 [3].

Weight initialization is an important step in training neural networks. This operation initializes the weights of the weight matrix. For the self-normalizing neural network (SNN) one needs to use the so-called lecun normal weight initialization. The lecun normal weight initialization draws samples centred around zero with standard deviation as [31]:

$$stddev = \sqrt{\frac{1}{in}} \tag{11}$$

Where in is the dimension of the weight matrix corresponding to the number of neurons in the previous layer [31]. When using an self-normalizing neural network (SNN) one needs to consider the SELU activation function which we already mentioned in section 3.3.6.
One also needs to consider using alpha dropout between the hidden layers. Alpha dropout was introduced by Klambauer et. al. in their paper about self-normalizing neural networks (SNNs) [3]. How alpha dropout works will be described in the following section 3.6.1.

3.6.1 Alpha Dropout

The normal dropout function drops neurons by randomly setting its weights to 0 with a probability of $1 - p$. This would go against the desired functionality of the self-normalizing neural networks (SNNs) to keep the mean and variance. The alpha dropout preserves the variance by applying an affine transformation [3]. So instead of setting the input to 0 with the normal dropout function, the alpha dropout sets the input to the low variance limit of the SELU function which is defined as $selu(x) = -\lambda\alpha =: \alpha'$ [3].

4 Hyperparameters

Hyperparameters are parameters in a machine learning system that are fixed before one trains the machine learning system. Every machine learning system has hyperparameters. These hyperparameters could be used to optimize the performance of the machine learning system [7]. This is because these hyperparameters directly influence the learning process of the system [8].

Deep learning neural networks, in particular, have a large number of different hyperparameters that are crucial for the performance of the system. Such hyperparameters include, for example, the architecture, regularization or optimization of the neural network [7].

If we remember the architecture of a neural network which we already discussed in chapter 3, there are the following hyperparameters that one could adjust to modify the architecture of a neural network:

- **Number of hidden layers:** Indicates how many hidden layers the neural network will have.
- **Number of neurons:** Indicates how many neurons each hidden layer of our neural network will have. This also depends on the layer form we use for our neural network.
- **Layer form:** The layer form is used to directly change the number of neurons in each hidden layer. Later on in section 6.3.3 we will introduce different layer forms that we used in the experiment.

One needs to know that there are other hyperparameters that one could alter that do not change the architecture of the neural network:

- **Weight initialization:** Which technique is used to initialize the weights of the neural network.
- **Activation function:** What type of activation function our neural network uses in the hidden layers.
- **Optimizer:** The optimizer specifies the guidelines according to which the weights of the neural network updated during the training process.
- **Learning rate**: Indicates how strong the weights in the neural network are adjusted during each weight update.
- **Dropout:** Specifies which dropout technique our neural network uses in the hidden layers.
- **Dropout rate**: Defines how high the dropout rate is.

These hyperparameters could not be modified like the hyperparameters that are used to alter the architecture of the neural network. Depending on which kind of neural network we want to use, we need to use a specific weight initialization technique or a certain type of dropout technique. For example, we have earlier introduced the self-normalizing neural network (SNN), when one wants to use the full potential of this type of neural network, one needs to use alpha dropout and the SELU activation function [3].

For the optimizer methods, there are several options which one could optimize. For example, every optimizer has a learning rate parameter. The SGD optimizer that we introduced in section 3.4.2 also has the momentum parameter that one could modify, but not all optimization algorithms have this parameter.

So when someone wants to optimize the parameters of activation functions, dropout techniques or optimizers it depends entirely on the different components one uses in the neural network. We need to mention that this list of hyperparameters is not complete, it should be more like a general overview of hyperparameters that one could modify to optimize the performance of a neural network.

4.1 Hyperparameter Optimization

Optimizing hyperparameters is a relatively important task for us because in our experiment we work with a large number of different datasets as well as with many different configurations of the hyperparameters that are used to optimize our machine learning algorithms.

But the optimization of hyperparameters is not only important to us. The problem of optimizing hyperparameters is dating back to the 1990s [7]. Since tuned hyperparameters improve over the standard settings that are provided in many machine learning libraries and the increase of machine learning in various application fields in the commercial sector, hyperparameter optimization became more popular [7].

The optimization of hyperparameters also comes with several challenges:

- The optimization of hyperparameters could be an expensive task, this is because the models one wants to optimize could get rather large (e.g. in deep learning) [7].
- The training process could also be a time-consuming task, if one uses huge datasets or when one wants to optimize complex machine learning pipelines [7].
- The set of hyperparameters often contains many different types of hyperparameters (continuous, categorical and conditional hyperparameters) [7].
- One does not know which hyperparameters will increase the performance and in which range these hyperparameters need to be optimized [7].

Generally spoken one could differentiate between two types of hyperparameter optimization methods, i.e. manual search and automatic search methods [8].
In manual search, the hyperparameters sets are defined by hand. Therefore this method depends on the fundamental intuition and knowledge of the users. In this process, one should be able to identify important parameters that have a bigger impact on the results. One should be able to determine the relationship between certain parameters and the final result with the help of visualization tools [8]. Due to the increase of hyperparameters, the optimization process becomes quite difficult to manage for humans. Since humans might misinterpret such high dimensional data [8].

For this reason, it is not practical for us to optimize the machine learning models by hand. Therefore we have to automate the optimization of hyperparameters. In the following sections, we will look into different techniques to automate the hyperparameter optimization process.

4.1.1 Grid Search

Grid search is one of the simplest methods for hyperparameter optimization, also known as full factorial design [7]. In Grid search, the user specifies a finite set of values for each hyperparameter. Out of all hyperparameter with all their defined values the cartesian product is formed. This cartesian product will result in the total amount of different configuration one has to try out to perform a hyperparameter optimization.
Due to the creation of a cartesian product, this method of optimizing hyperparameters suffers from the curse of dimensionality. The number of configuration increases exponentially

with the dimensionality of the configuration space [7]. By adding more values to the different hyperparameters, which would increase the resolution of discretization, the number of different configuration will also increase further [7].
Later on in section 6.2 we will perform a runtime estimation based on the datasets that we use in our reproduction study. For this runtime estimation, we will use the grid search optimization approach.

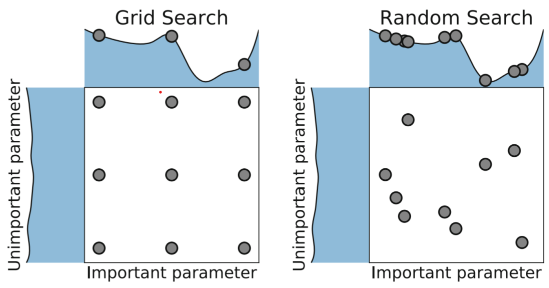
4.1.2 Random Search

An alternative to grid search would be random search. The random search algorithm trains machine learning models on randomly created hyperparameter configurations until a certain budget, that was specifically defined for this search is exhausted [7]. This approach especially works better than grid search when some of these hyperparameters are much more important than others [7].

Random search does not make any assumptions on the machine learning algorithm that is optimized. It is a useful baseline approach that could be used in the hyperparameter optimization process [7].This is due to the properties of the random search itself as it explores the whole configuration space rather than a large number of different values for a single hyperparameter [7].

An advantage of random search is that if the budget for the random search is big enough, it will be able to find a hyperparameter configuration with which the machine learning algorithm could achieve a performance close to the optimum [7].

Figure 11: Comparison between Grid and Random search, using an important and an unimportant feature, for minimizing a function [6] [7].



The fact that random search explores the whole configuration space is interesting if it is only feasible to train models with a certain amount of different configuration.
Assume that one has a given number of hyperparameters N and a fixed number of configurations (budget) B for the optimization process. The amount of values that one could try out per hyperparameter differs between grid and random search. With Grid search, one could use $B^{\frac{1}{N}}$ values per hyperparameter, whereas with random search one could use B different values per hyperparameter [7].

We need to mention that the random search optimization technique has further advantages compared to other optimization techniques. Random search allows easier parallelization since the workers do not need to communicate with each other [7]. Another advantage of this approach is the possibility of flexible resource allocation since one could add an arbitrary number of random points and still preserve a random search design.

5 Datasets and Preprocessing

Before one could train a machine learning model, one needs to select a dataset. With the dataset, one could then train a machine learning model so that it might be able to predict the correct results. During the training process, the machine learning model will be optimized for the selected dataset. That means that we train our model for exactly one task that it should perform.
For the trained machine learning model to provide the best possible predictions, we have to look at different characteristics of the dataset we use:

- **Features:** The number of features is interesting when constructing a machine learning model because the dimension of the input layer of our neural network needs to be adjusted. Different types of features (continuous, categorical or numerical) may need a different kind of preprocessing. One needs to know that some types of features could not be used in the training process before they are preprocessed.

- **Size of the dataset:** When the dataset only contains only a small number of instances the results could be relatively inaccurate because there is not enough data one could train with. One also should know those small datasets are prone to overfitting [33]. On the other hand, if the dataset contains a huge number of instances or features, it will take longer to train the machine learning model.

- **Quality:** The quality of the dataset depends on several metrics e.g. missing values, syntactical errors or duplicates. When one wants to use a dataset that has these problems, one needs to perform further preprocessing.

5.1 UCI Machine Learning Repository

For the reproduction study on self-normalizing neural networks (SNNs) we are using 121 different datasets. All these datasets are obtained from the UCI machine learning repository. The UCI machine learning repository is a platform that currently maintains 488 different datasets (as of December 2019) [38]. The UCI machine learning repository contains various datasets from all possible categories e.g. medical data collections, statistics and data collections about plants such as the Iris dataset that we already used as an example in section 2.1.1.

5.2 Data Normalization

The data in a dataset may not be directly suitable for training a machine learning model [32]. This is because this data of the dataset is often collected in its original form. Most of this raw data is designed so that it works with the original system from which the data comes. Some of these attributes may not be useful in its original form so that it could be used as a feature in a machine learning model.

So that we could use this data to design a machine learning model, it is common to carry out a series of manipulation steps to transform the original data or to generate new features with better properties from the existing attributes. These newly generated features are also called modelling variables or analytic variables [32].

In the following sections, we will mainly focus on different transformation techniques that allow us to transform the data attributes from its original form to a form with the desired properties.

5.2.1 Decimal Scaling Normalization

Decimal scaling normalization is one of the simplest approaches to normalize the values of a numerical feature [32]. When performing decimal scaling on a feature we are normalizing this feature by shifting the decimal point until the maximum absolute value of this feature is smaller than 1.

5.2.2 Min-Max Normalization

The goal of the min-max normalization technique is to scale all numerical values of a numerical feature to a specific range [32]. In literature, normalization refers to a particular case of min-max scaling with an interval of $[0, 1]$. Where the minimum of the original data is scaled to 0 and the maximum is scaled to 1 [32]. Also, when normalizing data for machine learning the interval $[-1, 1]$ is often used.

The main advantage of using normalization to re-scale data in machine learning is that all the data of the normalized feature is in the same range. Therefore, using normalization it will avoid that the min or max values of the normalized feature are dominating over other values of this feature [32]. Without normalization, those values would mislead the machine learning algorithm during the learning process. Using normalization when training artificial neural network (ANN) will speed up the learning process by helping the weights to converge faster [32].

5.2.3 Z-Score Normalization

Sometimes it is not possible to apply min-max normalization on our data. This could be when e.g. we do not know the minimum or maximum values of a feature. Even if do know the minimum and maximum of a feature, the data we want to normalize could still contain outliers that would bias the min-max normalization [32].

If this is the case, we could apply the so-called z-score normalization. By applying the z-score transformation to our data, the transformed data will have a zero mean and a standard deviation of one [32]. This is a relatively convenient feature of the z-score normalization because when using a self-normalizing neural network (SNN) we need to use standardized data to make the most effective use of this neural network.

The z-score transformation could be described mathematically as follows:

$$v^{'} = \frac{v - \overline{A}}{\sigma_A} \tag{12}$$

Where $\overline{A}$ is the mean of the values of feature A, σ_A is the standard deviation and v is the original value.

5.3 Data Transformation

In the previous section, we looked at some techniques for transforming features. In this section, we will introduce some techniques for generating new features. When transforming data, usually already existing features are used to generate new features with the help of different mathematical formulas [32].

Linear transformation: The easiest technique to create new features is the linear transformation. Several features are used to perform simple algebraic transformations (i.e. average,sum, translations ...) [32].

Rank transformation: A change in the distribution of the feature could change the model performance because one could uncover relationships that were hidden by the previous feature distribution. The easiest transformation to change the distribution of the values for numerical features is the rank transformation.

When using rank transformation we are taking a numerical feature out of our dataset and transform them into a new feature which contains the rank values ranging from 1 to m, where m is the number of instances in the dataset [32].

After transforming the numerical feature to a rank feature we could then perform normalization on this feature to get the probability of each feature value [32]. One needs to make sure that the rank transformation is performed simultaneously on the train and test set otherwise if performed separately the transformed feature will contain the wrong ranks [32].

5.4 Missing Values

When one is preprocessing a dataset one should know that real-world data is usually not perfect. This data could either be incomplete, contain errors or contain certain inconsistencies. One of these inconsistencies that could occur are missing values. Missing values in a dataset could have various reasons, like equipment errors or incorrect measurements [32]. How missing data is displayed depends on the respective dataset an example of how to display missing values would be NaN, empty string, -1, etc.

When training a machine learning model with missing values this could result in a loss of efficiency, complication in handling or analyzing the data or biased results between data record with missing values and complete data records [32]. For this reason, we could say that it is generally important to avoid missing values or to perform some preprocessing in which we clean up or repair the dataset.

Since we use datasets from the UCI machine learning repository for our experiment, it is relatively easy for us to find out whether our dataset contains any missing values, since this is indicated directly on the website for the respective dataset [38]. When one does not know whether the dataset contains missing values one will have to inspect the dataset to find out if the dataset contains any known missing value identifiers or other inconsistencies.
In the following sections, we will introduce some methods to reconstruct data records with missing values and how to remove data records with missing values if there is no possibility to reconstruct them reliable.

5.4.1 Preprocessing of Missing Values

When we are sure that our dataset contains missing values, we need to perform some preprocessing. There are different approaches to how we could preprocess our dataset. Referring to section 5.4 it is important to preprocess missing values in our dataset since we could relatively easy improve the performance of our machine learning models by removing or reconstructing them.

One of the simplest technique that used even today is the removal of data records that contain missing values. This will reduce the size of the dataset, but it also has certain advantages. Assuming that the values are missing completely at random (MCAR), by removing these data records would this result in an unbiased parameter estimates [32]. Even if these values are missing at random, this approach is not always optimal because when using this approach on a dataset with a large number of missing values we would remove many data records.
If these values are not accidentally missing but have been intentionally omitted, we may not remove them as this would lead to biased results [32]. An example would be if people with a low income were less likely to report their income level. This would lead to a biased mean.

Replacing missing values with the most common attribute for nominal features and the average for numerical features is a widely used way to replace missing values [32]. Especially when many instances of the dataset contain missing values.

6 Empirical Results

In this chapter, we will give a short overview of the paper "Self-Normalizing Neural Networks" [3]. Later on, we will present the project structure of our experiment and our results on the reproduction study. The source code of this project can be found on GitHub[1].

In this paper, Klambauer et al. [3] introduced a new kind of neural network that solves one of the main problems of a feed-forward neural network (FNN). The problem with FNNs is that they usually have to be kept relatively small and shallow since if they get too big they will no longer provide optimal results [3].
This new type of neural network is called self-normalizing neural network (SNN) which we already discussed in detail in section 3.6. Due to the convergence properties of this network, it is possible to train neural networks with many layers [3].
To verify the performance of the self-normalizing neural network (SNN), Klambauer et al. [3] selected different machine learning methods to compare the performance of the self-normalizing neural network (SNN) with the performance of theses already existing machine learning methods. For this experiment, Klambauer et al. [3] used 121 different UCI dataset and the Tox21 dataset [3].

Our main topic in this work is to implement a program that makes it possible to reproduce these results. We are mainly focusing on the newly introduced self-normalizing neural network (SNN) and the performance results, Klambauer et al. [3] produced on the UCI datasets.

[1]GitHub repository: `https://github.com/ehreph/SnnHyperparameterOptimization`

6.1 Libraries

We use a variety of different libraries and frameworks for the implementation of our experiment. For this reason, we would like to go into more detail in the following section and explain which libraries we used in our program.

Since we mainly deal with machine learning in this experiment, we decided to use the programming language Python in version 3.x. The reason why we use Python as the programming language for this experiment is that the programming language is easy to learn, and it also offers many different libraries that we could apply in the area of machine learning.

Another reason why we use Python is that Python offers certain advantages due to the simplicity of the data structures. An example of this would be to multiply two matrices. Such a function would first have to be implemented in Java, but Python offers direct support.

This would not be a decisive reason for Python, but since we mainly work on machine learning in our experiment, this is relatively important for us. Since we often have to perform matrix operations in various stages of data processing, such as preprocessing, as well as when calculating different statistical values.

6.1.1 TensorFlow

Tensorflow is an open-source library developed by Google that is used for numerical computation and large-scale machine learning [35]. The TensorFlow library is a bundle of machine learning and deep learning models and algorithms.

This is, of course, a big advantage for us because TensorFlow offers an abstraction between our implementation and existing machine learning algorithms.
Since we do not need a too complicated neural network architecture for our experiment, we decided to use the high-level API Keras in connection with TensorFlow.

6.1.2 Pandas

Pandas is an, highly performance-optimized, open-source python library that is used to process data. The core features of the pandas library are data processing and data manipulation. The library also provides intelligent data alignment and integrated handling of missing data [37].

We mainly used pandas in our project to load the different datasets and hyperparameter configurations. We also used pandas to store and process our results.

6.1.3 Keras

Keras is an open-source library designed for the fast implementation of neural networks. Keras should not be seen as a deep learning framework but rather as an interface for it [36]. Since Keras is to be seen more as an interface, there is also the possibility to use various deep learning frameworks in the backend. Keras offers backend support for TensorFlow, Theano, or MXNet. As we mentioned earlier in the 6.1.1 section, Keras provides a high-level API that greatly simplifies the implementation of neural networks.

Although Keras is only a high-level API for using various deep learning frameworks, Keras offers several advantages [17]:

- Possibility to use CPUs and GPUs equally
- Multi-input and multi-output
- Simple prototyping of neural networks
- Modularity and expandability

6.2 Runtime Estimation

In section 4.1 we already talked about which approaches there are in machine learning with which one could optimize the performance of a machine learning algorithm. But all of these approaches need a rather large amount of time to be carried out.

That means if we want to optimize our machine learning algorithm, we have no choice but to try out different configurations. To show how time-consuming this could become, we would like to present a small abstract example of a runtime estimation. Later on, we will compare this example with our runtime estimation of the 121 UCI datasets.

Let's say we have a commercially available computer that trains a model X for the dataset Y for 100 epochs. Assuming that our model X will need about 30 seconds of training time per epoch. That means that our program will need approximately 50 minutes to fully train our model X for the dataset Y.

Now we have a fully trained model, but the problem is we do not know if this model has the best performance. To validate whether our model is optimal or not we need to perform a hyperparameter optimization.

To record the hyperparameter configuration for the optimization process in our project, we store the hyperparameter configuration in a JSON file. By changing the hyperparameter configuration in the JSON file we can directly change the hyperparameters that are used in the optimization process without changing anything in the code itself. Such a hyperparameter configuration could look like this:

```
{
  "dense_units": [1024, 512, 256],
  "n_dense": [2, 3, 4, 8, 16, 32],
  "dropout_rate": [0, 0.5],
  "learning_rate": [0.01, 0.1, 1],
  "layer_form": ["rect", "conic"],
}
```

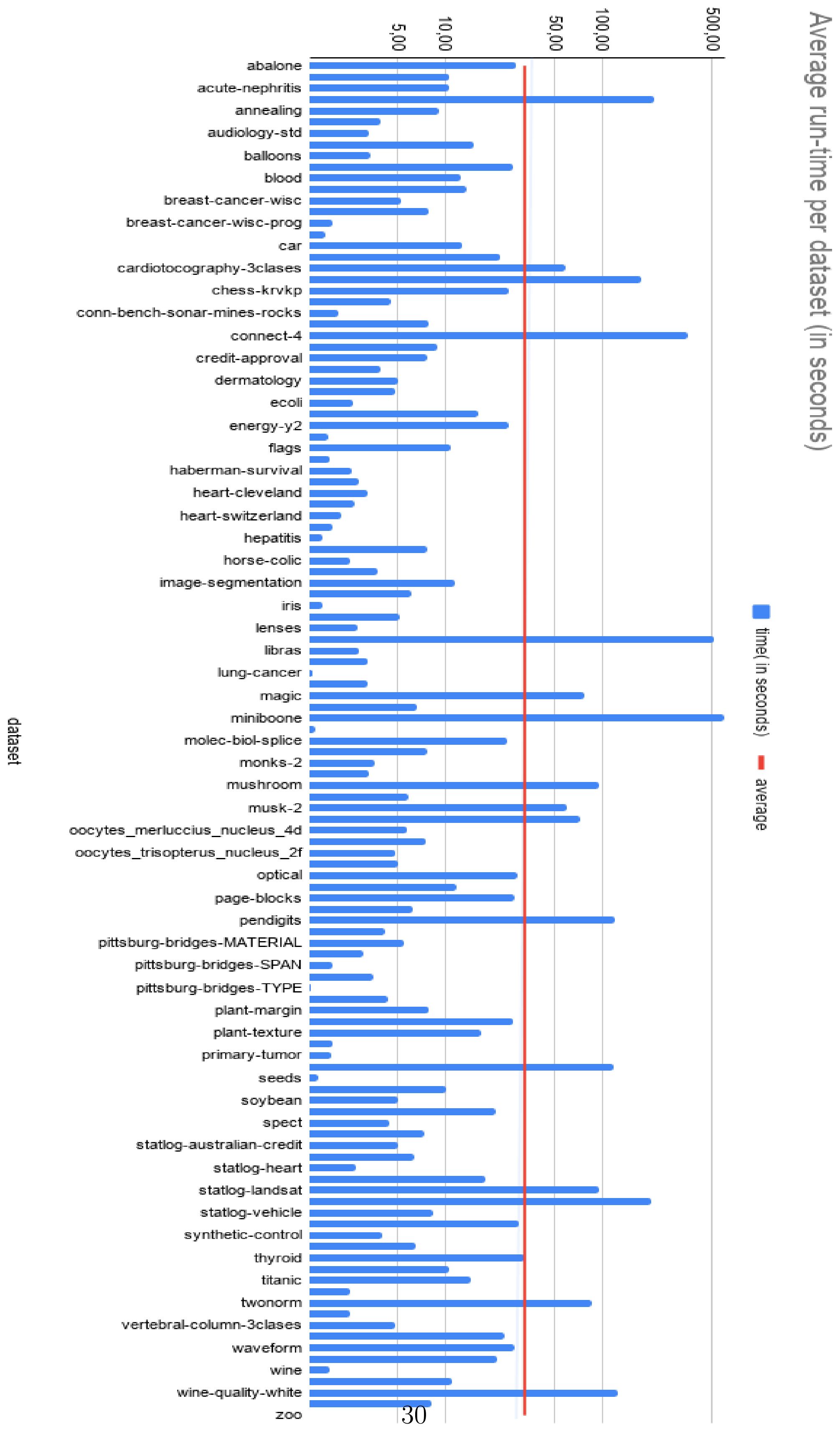
To calculate the number of models that need to be trained, we simply multiply the number of values per parameter. This is also called permutation without repetition. The previously defined set of hyperparameter values gives us a total of 216 different hyperparameter set. Conversely, this means that our program will need around 180 hours of pure computing time to train a model for each hyperparameter set for dataset Y.
Although this was only a fictional example, we could use this example to imagine how much time is needed to optimize a machine learning model on a single dataset.

Of course, each of the 121 UCI datasets has a different amount of features and data records, therefore one could not exactly determine how much time it would take to perform a hyperparameter optimization on these datasets.

For this reason, we measured the time our program needs to train all datasets with a minimal configuration. We want to get a rough estimate of how long our program would run if we performed the hyperparameter optimization for all datasets. For this minimal configuration, we train a single model with a single hyper-parameter configuration, per dataset, for 100 epochs.

In figure 12 one could see the result of this runtime analysis. The average runtime per dataset here is approximately 32 seconds. With 121 datasets, the average time it takes to train a self-normalizing neural network (SNN) for each dataset is approximately one hour and 272 seconds.
If we were to perform a hyperparameter optimization on all 121 UCI dataset with the 216 possible hyperparameter sets, this process would roughly need about 232 hours of pure training time.
As one could see from this analysis, training all configurations for all datasets would take a relatively long time, for this reason, we decided to use the hyperparameter configurations that Klambauer et al. [3] used. This allows us to reduce the training time to a minimum.

Figure 12: In this diagram, one could see the average training time of the different datasets. The dataset “lung cancer” needs the shortest time to train with approximately 1.44 seconds and the dataset “miniboone” takes the longest to train with 602.24 seconds. The average training time one needs for training a model on a dataset is approximately 32 seconds.



6.3 Project Structure

We initially had to train a lot of machine learning models as we followed the approach of hyperparameter optimization. We thought about how we could record the current progress of the hyperparameter optimization process.

For us it was not possible to train models for all hyperparameter configurations in one run due to the lack of resources (memory) and computational power. Therefore, it was not possible for us to directly compare the performance of the different models.

So that we could compare the results we had to implement a functionality that would save all of our results. We also wanted to store the models that we trained during this process.

For this reason, we introduced a so-called "target" folder. This target folder will be created when one starts the program. During the training process, it will create a separate folder for all of the datasets we use during the training process. In this target folder, we also added a results folder that will contain the performance data of all models that we have trained.
If one uses the neural networks classes of Keras (Sequential, Functional model, Model subclass), Keras offers an implemented save function for saving the trained machine learning models. This save function offers three different options on what data of our model should be saved [36]:

- Saving everything into a single archive in the TensorFlow SavedModel format (or in the older Keras H5 format).
- Only saving the architecture and configuration of the model.
- Saving the values of the weights only.

6.3.1 Prototyping

In this section, we want to show how easy it is to implement a neural network with Keras. But first of all, we need to know what the goal of our model will be. Should the result be a classification or simply a real number. Now one could start building a neural network by adding different layers to the model. Keras offers a variety of different layers with which one could build a neural network. If needed it is also possible to implement new layers [36]. Here one could see how such a construction of a neural network in Keras could look like:

```python
nstart = 256
input_size = 12
output_size = 1

model = Sequential()
model.add(Dense(units=nstart, input_shape=(input_size,),
    kernel_initializer='lecun_normal', activation='relu'))
model.add(Dense(output_size, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['
    accuracy'])
```

Modeling a simple neural network in Keras could be divided into three steps:

- Creation of a new model by calling the function "Sequential()".
- Adding different layers (input, output, hidden layers).
- Compile the previously defined model.

6.3.2 Session Management

Every model that we create and train in Keras needs memory. For this reason, we have to deal with how and when we release the currently used memory. If our computer doesn't have enough memory available to compute new models our program may crash.

To ensure that our computer always has enough memory while training our models, we have to deal with how we delete old models that are no longer needed and how to release the memory they used.

Keras offers a relatively simple approach to this problem. When using Keras we could free the used memory by deleting all data from the current session [36]. After that, we could create a new session.

In this process, we need to make sure that the current session will not be deleted while we are training one of our models. If one deletes the session during the training process of a model the program will crash. To prevent this we decided to create a new session each time the program has finished training all models for the current dataset.
Deleting the data of our current Keras session works as follows:

```python
import keras as K
import tensorflow as tf

def create_new_tf_session():
    K.backend.clear_session()
    s = tf.Session(config)
    K.backend.set_session(s)
```

6.3.3 Hidden Layer Architecture

One of the hyperparameters that Klambauer et al. [3] used in his paper is the so-called layer form. Two different layer forms were used during the hyperparameter optimization. We will now shortly introduce these two types of hidden layer architectures.

When using the rectangular layer form all of the hidden layers will have the same constant number of neurons throughout the neural network [3].
When we want to apply the conic layer form to a neural network only the first hidden layer will have the number of neurons we previously defined. The subsequent hidden layers then each have a smaller number of neurons. The number of neurons in the subsequent hidden layers is determined using the formula of geometric progression [3].
The formula for geometric progression is defined as follows:

$$factor = (n_{out}/n_{in})^{(1/n_{dense})} \tag{13}$$

$$n_{size} = neurons * factor \tag{14}$$

Where:

- $factor$ multiplication factor
- n_{out} number of neurons in the output layer,

- n_{in} number of neurons in the input layer
- n_{dense} amount of hidden layers that still follow
- $neurons$ number of neurons of the current hidden layer
- n_size number of neurons of the next layer

6.4 Results

In this section, we would like to present our results on the reproduction study. As we already mentioned in section 6.2 we are using the hyperparameter configurations that were used to create the performance results for the paper from Klambauer et al. [3]. These results contained all of the performance results for each hyperparameter configuration. So before we could use these results we needed to extract the hyperparameter sets which resulted in the best performance.

To accomplish this we created a small python script that iterated through all result files. From these results files, we extracted the hyperparameter sets that resulted in the best performance. After conducting this data extraction we combined the selected hyperparameter sets and saved them in a file which we could use later to perform the reproduction of the results.

To reproduce the performance results we created a python script which used all of the UCI datasets and the hyperparameter sets that we have extracted from the files Klambauer et al. provided to us. This python script trained a model for each dataset and its corresponding hyperparameter set. After the training process of the machine learning models was finished we saved these results so that we could later analyze these results.

Since different results could occur when training machine learning models, we have decided to carry out the training process a total of four times. By conducting several runs we could determine the best performance of the machine learning models and the standard deviation between the machine learning models from several results. By calculating the standard deviation for the machine learning models per dataset we can better compare our results with the results of Klambauer et al. [3].

Now we would like to present our results. The following table contains the performance results for all UCI datasets we and Klambauer et al. [3] used. The following table contains per dataset the best performance, the standard deviation, the performance results of Klambauer et al. [3] and the difference between our results and the results of Klambauer et al. [3]. Following this table, we would like to discuss the results in more detail.

dataset	**test_accuracy**	**stddev**	**paper_accuracy**	**difference**
abalone	0.654	0.019	0.678	-0.024
acute-inflammation	1.000	0.014	1.000	0
acute-nephritis	1.000	0.000	1.000	0
adult	0.849	0.050	0.854	-0.005
annealing	0.889	0.059	0.760	0.129
arrhythmia	0.676	0.067	0.699	-0.023

audiology-std	0.867	0.109	0.840	0.027
balance-scale	0.989	0.128	0.981	0.008
balloons	1.000	0.250	1.000	0
bank	0.898	0.012	0.901	-0.003
blood	0.894	0.061	0.813	0.081
breast-cancer	0.837	0.096	0.993	-0.156
breast-cancer-wisc	1.000	0.021	0.796	0.204
breast-cancer-wisc-diag	0.977	0.017	0.989	-0.012
breast-cancer-wisc-prog	0.850	0.042	0.789	0.061
breast-tissue	0.812	0.187	0.808	0.004
car	0.981	0.194	0.844	0.137
cardiotocography-10clases	0.859	0.028	0.944	-0.085
cardiotocography-3clases	0.928	0.047	0.988	-0.06
chess-krvk	0.166	0.037	0.996	-0.83
chess-krvkp	0.996	0.002	0.881	0.115
congressional-voting	0.727	0.044	0.661	0.066
conn-bench-sonar-mines-rocks	0.875	0.041	0.885	-0.01
conn-bench-vowel-deterding	0.987	0.431	0.998	-0.011
connect-4	0.883	0.044	0.885	-0.002
contrac	0.588	0.018	0.579	0.009
credit-approval	0.894	0.033	0.890	0.004
cylinder-bands	0.844	0.047	0.828	0.016
dermatology	0.982	0.370	0.989	-0.007
echocardiogram	0.825	0.066	0.909	-0.084
ecoli	0.922	0.072	0.917	0.005
energy-y1	0.974	0.193	0.964	0.01
energy-y2	0.897	0.071	0.911	-0.014
fertility	0.933	0.094	0.960	-0.027
flags	0.600	0.195	0.563	0.037
glass	0.606	0.154	0.774	-0.168
haberman-survival	0.761	0.019	0.776	-0.015
hayes-roth	0.875	0.161	0.964	-0.089
heart-cleveland	0.630	0.053	0.671	-0.041
heart-hungarian	0.844	0.098	0.836	0.008
heart-switzerland	0.421	0.096	0.613	-0.192
heart-va	0.367	0.047	0.440	-0.073
hepatitis	0.792	0.040	0.872	-0.08
hill-valley	0.692	0.094	0.592	0.1
horse-colic	0.821	0.052	0.882	-0.061
ilpd-indian-liver	0.744	0.013	0.760	-0.016
image-segmentation	0.965	0.465	0.915	0.05
ionosphere	0.981	0.024	0.955	0.026
iris	0.957	0.061	1.000	-0.043
led-display	0.540	0.092	0.792	-0.252
lenses	0.750	0.125	1.000	-0.25
letter	0.979	0.003	0.978	0.001
libras	0.889	0.439	0.867	0.022
low-res-spect	0.925	0.036	0.925	0
lung-cancer	0.600	0.100	0.750	-0.15

lymphography	0.826	0.022	0.973	-0.147
magic	0.876	0.106	0.874	0.002
mammographic	0.807	0.094	0.850	-0.043
miniboone	0.932	0.377	0.932	0
molec-biol-promoter	0.875	0.121	0.923	-0.048
molec-biol-splice	0.898	0.016	0.901	-0.003
monks-1	0.976	0.289	0.877	0.099
monks-2	0.918	0.129	0.722	0.196
monks-3	0.964	0.023	0.766	0.198
mushroom	1.000	0.276	1.000	0
musk-1	0.958	0.026	0.916	0.042
musk-2	0.998	0.003	0.993	0.005
nursery	0.999	0.311	0.999	0
oocytes_merluccius_nucleus_4d	0.828	0.019	0.847	-0.019
oocytes_merluccius_states_2f	0.961	0.029	0.953	0.008
oocytes_trisopterus_nucleus_2f	0.869	0.030	0.838	0.031
oocytes_trisopterus_states_5b	0.993	0.050	0.965	0.028
optical	0.992	0.447	0.973	0.019
ozone	0.976	0.014	0.975	0.001
page-blocks	0.976	0.008	0.970	0.006
parkinsons	0.933	0.083	0.918	0.015
pendigits	0.993	0.517	0.976	0.017
pima	0.802	0.033	0.786	0.016
pittsburg-bridges-MATERIAL	0.938	0.157	0.962	-0.024
pittsburg-bridges-REL-L	0.750	0.149	0.885	-0.135
pittsburg-bridges-SPAN	0.571	0.058	0.739	-0.168
pittsburg-bridges-T-OR-D	0.938	0.114	0.960	-0.022
pittsburg-bridges-TYPE	0.562	0.081	0.769	-0.207
planning	0.607	0.053	0.778	-0.171
plant-margin	0.846	0.471	0.855	-0.009
plant-shape	0.742	0.041	0.743	-0.001
plant-texture	0.863	0.015	0.812	0,051
post-operative	0.786	0.107	0.727	0.059
primary-tumor	0.520	0.044	0.573	-0.053
ringnorm	0.983	0.002	0.988	-0.005
seeds	0.969	0.040	0.981	-0.012
semeion	0.184	0.049	0.960	-0.776
soybean	0.903	0.053	0.918	-0.015
spambase	0.946	0.195	0.946	0
spect	0.750	0.052	0.920	-0.17
spectf	0.854	0.133	0.726	0.128
statlog-australian-credit	0.712	0.025	0.703	0.009
statlog-german-credit	0.770	0.034	0.776	-0.006
statlog-heart	0.890	0.090	0.940	-0.05
statlog-image	0.847	0.399	0.971	-0.124
statlog-landsat	0.936	0.004	0.919	0.017
statlog-shuttle	0.997	0.114	0.999	-0.002
statlog-vehicle	0.858	0.061	0.844	0.014
steel-plates	0.774	0.024	0.784	-0.01

synthetic-control	0.989	0.202	1.000	-0.011
teaching	0.565	0.096	0.658	-0.093
thyroid	0.982	0.004	0.982	0
tic-tac-toe	0.993	0.007	0.987	0.006
titanic	0.785	0.017	0.793	-0.008
trains	1.000	0.239	1.000	0
twonorm	0.975	0.004	0.984	-0.009
vertebral-column-2clases	0.915	0.077	0.883	0.032
vertebral-column-3clases	0.851	0.291	0.883	-0.032
wall-following	0.923	0.259	0.924	-0.001
waveform	0.897	0.018	0.872	0.025
waveform-noise	0.876	0.012	0.883	-0.007
wine	1.000	0.035	0.652	0.348
wine-quality-red	0.654	0.026	0.637	0.017
wine-quality-white	0.646	0.163	1.000	-0.354
yeast	0.605	0.099	0.639	-0.034
zoo	1.000	0.120	1.000	0

From the table, one could see that we were able to reproduce the results of Klambauer et al. [3] relatively well. Most of our results are in the range of the results from Klambauer et al. [3] $\pm$ of the standard deviation. As one could see from the results, there are significant differences in some of the datasets.

For this reason, we have taken another look at the log files of our program. We found out that some of the models started to fluctuate during the training of certain datasets. One way of counteracting this would be to use an early stopping mechanism that would allow us to stop the training process early if the accuracy deteriorates over time.

We would like to mention that by conducting the reproduction study we have achieved significantly better performance results with some datasets. Especially the performance on the datasets annealing, blood, breast-cancer-wisc, chess-krvkp, monks-2, monks-3 and wine were improved significantly.

7 Conclusion

In the theoretical part of our thesis, we have introduced the general concepts of machine learning. We also created an overview of the neural network machine learning technique. We have introduced several concepts and problems one should know when using neural networks. We also discussed the use of hyperparameters in neural networks and the optimization of them.

In the last chapter of our thesis, we presented the structure of our program that we have written to reproduce the performance results of Klambauer et al. By performing this reproduction study we have implemented a program that makes it possible to perform a hyperparameter optimization with an arbitrary hyperparameter configuration on an arbitrary dataset for the self-normalizing neural network (SNN) architecture.

As our research goal states, we performed a reproduction of the performance results of the UCI machine learning repository datasets that Klambauer et al. used. With this reproduction study, we could empirically verify the performance results for the self-normalizing neural network (SNN) architecture on the UCI machine learning repository datasets. We also accomplished to train several machine learning models with the self-normalizing neural network (SNN) architecture that outperformed the original results.

References

[1] D. Michie, " "Memo" Function and Machine Learning"
https://stacks.stanford.edu/file/druid:jt687kv7146/jt687kv7146.pdf
Accessed: 02.03.2020

[2] A. L. Samuel, "Some studies in machine learning using the game of Checkers"
https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.368.2254
Accessed: 02.03.2020

[3] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, "Self-Normalizing Neural Networks"
https://arxiv.org/abs/1706.02515
Accessed: 11.12.2019

[4] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, "Algorithms for Hyper-Parameter Optimization"
http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf
Accessed: 11.12.2019

[5] S. B. Kotsiantis, Department of Computer Science and Technology "Supervised Machine Learning: A Review of Classification Techniques"
http://www.informatica.si/index.php/informatica/article/viewFile/148/140
Accessed: 22.12.2019

[6] J. Bergstra, Y. Bengio, "Random Search for Hyper-Parameter Optimization"
http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf
Accessed: 22.12.2019

[7] M. Feurer, F. Hutter, "Hyperparameter Optimization. In Automated Machine Learning (pp. 3-33)"
https://library.oapen.org/handle/20.500.12657/23012
Accessed: 22.12.2019

[8] J. Wu, X. Chen, H. Zhang, L. Xiong. H. Wei,"Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization"
https://www.sciencedirect.com/science/article/pii/S1674862X19300047
Accessed: 20.7.2020

[9] T.M. Mitchell, R.M. Keller, S.T. Kedar-Cabelli "Explanation-Based Generalization: A Unifying View"
https://link.springer.com/content/pdf/10.1023/A:1022691120807.pdf
Accessed: 02.03.2020

[10] J. Bartlett, E. Holloway, "Generalized Information, A Straightforward Method for Judging Machine Learning Models"
https://journals.blythinstitute.org/ojs/index.php/cbi/article/view/52/42
Accessed: 02.03.2020

[11] D. M. Hawkings, “The Problem of Overfitting“
https://pubs.acs.org/doi/pdfplus/10.1021/ci0342472
Accessed: 02.03.2020

[12] J. Brownlee, “Supervised and Unsupervised Machine Learning Algorithms“
https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/
Accessed: 03.04.2020

[13] M. Wiedenbeck, C. Züll, “M. Wiedenbeck, C. Züll, “Klassifikation mit Clusteranalyse: grundlegende Techniken hierarchischer und K-means-Verfahren“
https://www.ssoar.info/ssoar/bitstream/handle/document/20142/ssoar-2001-wiedenbeck_et_al-klassifikation_mit_clusteranalyse.pdf?sequence=1
Accessed: 03.04.2020

[14] R. P. Lippmann, “Pattern Classification Using Neural Networks“
https://www.researchgate.net/profile/Lee_Feldkamp/publication/220932765_Multiclass_Pattern_Classification_Using_Neural_Networks/links/550c7bd90cf2ac2905a42559/Multiclass-Pattern-Classification-Using-Neural-Networks.pdf
Accessed: 03.04.2020

[15] S. Garcıá, A. Fernández, J. Luengo, F. Herrera, “A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability“
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.477.5243\&rep=rep1\&type=pdf
Accessed: 22.04.2020

[16] A. Mishra, “Metrics to Evaluate your Machine Learning Algorithm“
https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234
Accessed: 22.04.2020

[17] S. Luber, N. Litzel, “Was ist Keras?“
https://www.bigdata-insider.de/was-ist-keras-a-726546/
Accessed: 10.05.2020

[18] Cogito, “How to Validate Machine Learning Models: ML Model Validation Methods“
https://www.cogitotech.com/blog/how-to-validate-machine-learning-models/
Accessed: 11.06.2020

[19] w. S. Mcculloch, W. H. Pitts, “ A logical calculus of the ideas immanent in Nervous Activity“
http://aiplaybook.a16z.com/reference-material/mcculloch-pitts-1943-neural-networks.pdf
Accessed: 13.06.2020

[20] J. V. Tu,“Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes“

https://www.sciencedirect.com/science/article/abs/pii/S0895435696000029#! Accessed: 13.06.2020

[21] A. Krenker, J. Bešter, A. Kos,“Introduction to the Artificial Neural Networks“, Chapter 1
https://www.researchgate.net/publication/319316102_Artificial_Neural_Networks_-_Methodological_Advances_and_Biomedical_Applications
Accessed: 13.06.2020

[22] K. Suzuki,“Artificial Neural Networks - Methodological Advances and Biomedical Applications“
https://www.researchgate.net/publication/319316102_Artificial_Neural_Networks_-_Methodological_Advances_and_Biomedical_Applications
Accessed: 13.06.2020

[23] C. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall, “Activation Functions: Comparison of trends in Practice and Research for Deep Learning“
https://arxiv.org/abs/1811.03378 Accessed: 23.06.2020

[24] P. Harrington, “Machine Learning in Action“
Accessed: 02.07.2020

[25] F. Sakketou, N. Ampazis , “On the Invariance of the SELU Activation Function on Algorithm and Hyperparameter Selection in Neural Network Recommenders“
https://www.researchgate.net/publication/333108063_On_the_Invariance_of_the_SELU_Activation_Function_on_Algorithm_and_Hyperparameter_Selection_in_Neural_Network_Recommenders
Accessed: 05.07.2020

[26] T. Dozat, “Incorporating Nesterov Momentum into Adam“
https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ
Accessed: 05.07.2020

[27] G. W. Burr , R. M. Shelby , S. Sidler , C. di Nolfo , J. Jang , I. Boybat , R. S. Shenoy , P. Narayanan , K. Virwani , E. U. Giacometti , B. N. Kurdi , H. Hwang, “Experimental Demonstration and Tolerancing of a Large-Scale Neural Network (165 000 Synapses) Using Phase-Change Memory as the Synaptic Weight Element“
https://ieeexplore.ieee.org/abstract/document/7151827
Accessed: 05.07.2020

[28] N. H. Christiansen, P. E. Torbergsen Voie, O. Winther, J. Høgsberg, “Comparison of Neural Network Error Measures for Simulation of Slender Marine Structures“
https://www.hindawi.com/journals/jam/2014/759834/
Accessed: 05.07.2020

[29] I. Goodfellow, Y. Bengio, A. Courville, “Deep Learning“
http://www.deeplearningbook.org Accessed: 05.07.2020

[30] H. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting“
https://dl.acm.org/doi/abs/10.5555/2627435.2670313
Accessed: 15.07.2020

[31] A. Madasu, V. A. Rao, “Effectiveness of Self Normalizing Neural Networks for Text Classification“
https://arxiv.org/abs/1905.01338
Accessed: 15.07.2020

[32] S. García, J. Luengo, F. Herrera, “Data preprocessing in data mining“
https://www.springer.com/de/book/9783319102467
Accessed: 15.07.2020

[33] Ng, Hong-Wei, et al., “Deep learning for emotion recognition on small datasets using transfer learning.“
https://dl.acm.org/doi/abs/10.1145/2818346.2830593
Accessed: 20.07.2020

[34] J. Brownlee, “How to Train a Final Machine Learning Model“
https://machinelearningmastery.com/train-final-machine-learning-model/
Accessed: 7.05.2020

[35] Google Brain Team, “Tensorflow“
https://www.tensorflow.org/learn
Accessed: 11.05.2020

[36] F. Chollet, “Keras“
https://keras.io/api/
Accessed: 11.05.2020

[37] “Pandas“ https://pandas.pydata.org/about/
Accessed: 11.05.2020

[38] “UCI Machine Learning Repository“
https://archive.ics.uci.edu/ml/index.php
Accessed: 11.12.2019

List of Figures