Humboldtgymnasium Solingen

Subject work in grade Q1

TensorFlow
Representation of the principle and application on a self-chosen example

TensorFlow - Darstellung des Prinzips
und Anwendung an einem selbst gewählten Beispiel

Nathan Mossaad
Basic course Information Technology if1
Subject teacher: Mr. Pohler

School Year 2020/21
06.01.2021

Contents

1 Introduction

Why is digital learning mainly in the form of artificial intelligence and machine learning so talked about? The answer lies within how we humans perceive and understand the world around us. As an example, if you see a digit like 8, you can instantly understand it and categorize it no matter, how or from whom it has been written. But how are computers supposed to recognize something like this? The current best solution to this problem is machine learning and its varieties. But how do we write a program that can recognize these digits and give out the written digit from an image? [1]

[1] Base on: [1]

2 AI, ML, DL & NN

Technical terms which are used in the following can be looked up in the section "Definitions of terms".

2.1 Stages of digital learning

There are four basic stages of digital learning. These are:

- AI - Artificial Intelligence
- ML - Machine Learning
- DL - Deep Learning
- NN - Neural Networks

In this paper I will focus on the last three, but first of all: What are they? As seen in the image on the side, every higher stage or algorithm encases its subarea.

Artificial Intelligence is basically the equivalent to the human mind, able to apply the learned to new, not seen before situations.
In **Machine Learning** the algorithm replicates what it has learned prior to the application on a similar situation.
In **Deep Learning** we try to create the structures of a brain and not only similar building blocks, but entire interconnected "webs".
Finally, **Neural Networks** try to mimic neurons in a brain to create "similar" components and hopefully create similar results[2].

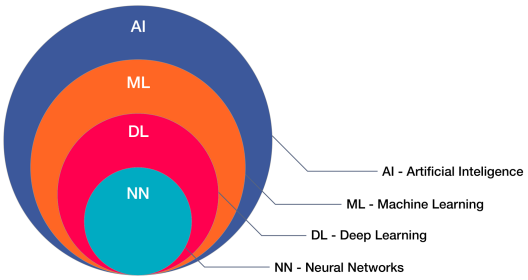


Figure 1: Stages of digital learning [2]

2.2 The first algorithm: "Hello World"

The easiest way to imagine these algorithms is in layers in which there are neurons that are connected. As an example, we will take the **"Hello World"** of machine learning: recognizing the digits 0 to 9. The MNIST database is a great starting point to understand the basics.

[2]Source: [11]

2.2.1 MNIST database and nodes

The MNIST database is a collection of 70,000 handwritten letters that are 28 x 28 pixel images together with the corresponding digits. As an input to our neural network, we can take these images and split them up into single pixels. Those can be fed into 784 nodes. These first nodes can read the brightness value of the pixel as an input and pass it on as an output.

Figure 2: MNIST database sample images [10]

2.2.2 Individual layers

This collection of first nodes is our first layer. Each node except for the output nodes will be a function that takes in values of the nodes from the previous layer, processes them and calculates a new value that is passed onto the nodes from the next layer. This layer does not have to be similar to its predecessor, it just has to have nodes with inputs and outputs, regardless what the functions do or how many there are. One possibility could be that a node takes the value of five previous nodes and creates an average of the previous outputs. Usually these are quite a lot more complex like taking in the aforementioned values and multiplying these with so-called weights. These are negative and positive values with which the input is multiplied by.

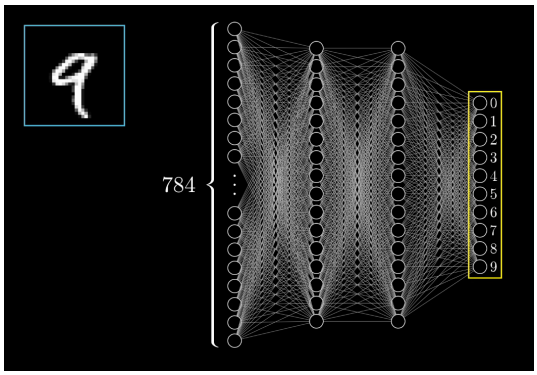


Figure 3: Representation of a Neural Network [1]

2.2.3 Layer structures

The results then get summed up and the result is inserted into the sigmoid function[3]

$$h_\theta(x) = \frac{1}{1 + e^{-x}}$$

which turns any numerical value into a number between -1 and +1. The bigger the number the closer it gets to 1 and the smaller the value the closer it nears to -1. This layer is then repeated multiple times, the more complicated the task

[3]Source: [9]

the more layers and nodes it will typically have. At the end of the chain the output nodes in our example ten, will then do the final calculations. Usually each has a value between 0 and 1, representing a percentage where each node represents how probable it is for its node to be the desired outcome.

2.3 Training a model

How is the model now trained?
Even in simple examples like this one, there are thousands of weights. Basically instead of reasoning and logic we just brute force it, similar to the concept of evolution. At the beginning we can just select random values for the weights and test it with multiple different values. Usually there are multiple hundred iterations, so-called epochs. The most accurate (sometimes multiple) model is then the winner of the epoch and is taken as a starting point for the next epoch. Instead of taking random values, these are altered and then the same happens as in the first epoch. The following epochs are doing the same as the second, but typically doing smaller and smaller alterations to the weights, which in the process are getting more and more accurate. Finally, after each epoch the overall accuracy is evaluated. After training to a degree one is satisfied with, the model is tested against examples, that it has not trained on.

2.3.1 Understanding the results

To get back to the MNIST database, there we have two sub databases, the first containing 60,000 images as training data and the second containing 10,000 as evaluation data. There has to be separate evaluation data, because only through never seen data the true accuracy can be evaluated.

In my simple tests I had an average accuracy of 98,56% after the 10th epoch on the training data, but a 98,12% accuracy on the testing data. This discrepancy can differ dramatically depending on multiple factors, especially with more complex problems.

This means that there will be exponentially increasing processing power required to improve the model.

3 The practical example with TensorFlow

3.1 What is TensorFlow?

In this example I will use TensorFlow which according to the project's website "is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications."[4]

An array can have multiple dimensions, imagine each as a spacial dimension converging from a single point.

- The first dimension would than be a line, a so-called vector: 1-axis tensor
- The second a surface, a so-called matrix: 2-axis tensor
- And the third a three-dimensional object like a pen: 3-axis tensor
- Higher dimensions can't be spacially comprehended

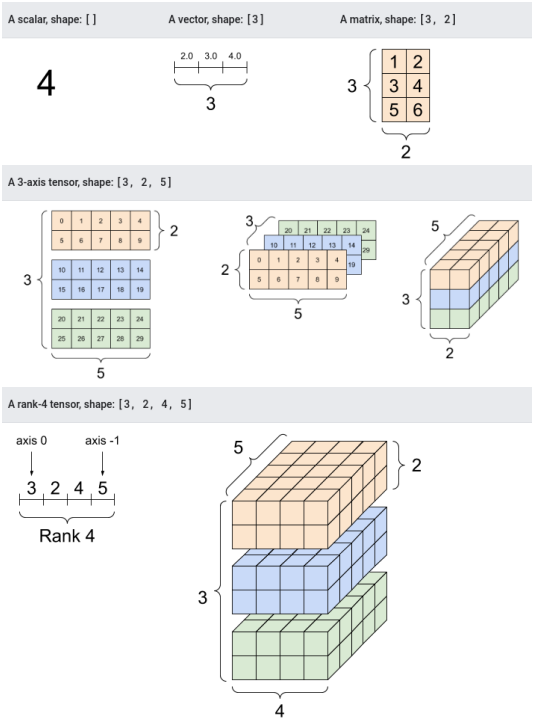


Figure 4: Graphical Representation of a Tensor [6]

On the right at the bottom you can see a "rank-4" tensor, with four dimensions. To further visualize this, there are simply three individual "rank-3" tensors, which make up this tensor.
But how does this differ from a simple node? This structure enables TensorFlow to combine lots of values etc. into one unified body to be dealt with. It saves processing power, especially in more complex models, with the downside of added complexity and less flexibility. This of course is an oversimplification and it is not possible for me to do it justice.[5]
In the following example I will use TensorFlow's python-API together with TensorBoard and run it with CUDA acceleration for increased performance. The source code can be found in the section "B. Source code" and should run on any GNU/Linux system with the required following dependencies: Python,

[4] From: [7]
[5] Source: [6]

TensorFlow, TensorBoard, matplotlib.

3.2 Structure

After loading in all needed libraries and clearing the data of previous runs, I first let TensorFlow download the built-in MNIST fashion database. MNIST fashion has an identical structure to the aforementioned MNIST database. Instead of digits there are more complicated images of clothing from Zalando, an online seller for clothing, to be evaluated.[6]
After that I create four variables:

- The first two containing 60,000 images with its appropriate labels
- The other two with 10,000 test images for later evaluation of the model

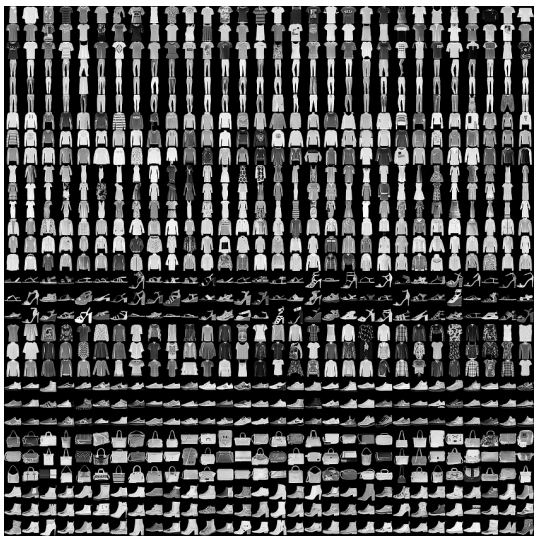


Figure 5: Sample Images from the Fashion MNIST database [4]

To later categorize the digits, I created a list with the corresponding names like “Dress” and “Coat”. After bringing the brightness values of the individual pixels into the range from 0 to 1 from the usual 0 to 255, by dividing them with 255, I display the first 25 images with their translated labels as seen on the right.
After verifying that everything worked, I begin with setting up the required layers. In this model I have three layers:

- The first being the input layer, which can take in an image with 28 x 28 pixels, flattened into 784 data points, which then are used as input for the second layer.
- The second layer has 128 nodes and takes in the data of the first layer and processes it.
- In the last layer, there are 10 nodes, which do the final calculations. Each of them representing one of the digits from zero to nine and giving a percentage of how probable it is, for it being the desired outcome.

After setting up the layers, the model can be compiled, which means given over to TensorFlow with all needed information:

[6] Source: [4]

- The optimizer, that controls which weights are updated in which way
- The loss function, that measures how accurate the model is
- Finally the target metrics, that tell TensorFlow what the goal is, in our case how accurate the model is

After that I set up the required logging directories and the log function for TensorBoard to tell how the model changed over time. This then leads to training where the raw training data, validation data together with the amount of desired epochs and the logging function is given to train. In this example there will be 10 epochs.
To finally know the accuracy after training, I just print it to the terminal and finally start the TensorBoard web-server to analyze the results. The code is based mostly on "Basic classification: Classify images of clothing"[7] and "Get started with TensorBoard". [8]

3.3 Results

3.3.1 Realtime output

The first output simply states which version of TensorFlow is being used, here it is version 2.4.0 released on the 14th December 2020[9].
The next visible output is the current epoch, followed by a progress-bar. The progress-bar shows while running:

- The current step at the beginning of the line
- An estimated time for running the current epoch
- The current loss based on the training data while learning
- The current accuracy based on the training data while learning

After each epoch further information is given, with the values of epoch 10. These are:

- The amount of run through steps, here 1875
- How long it took to run the epoch, here $\approx$ 12s
- How long the average step took, here $\approx$ 6ms/step
- How low the loss in the last step was while training, here $\approx$ 23,76%
- How high the accuracy in the last step was while training, here $\approx$ 90,96%
- How low the loss value in the last step was while training, here $\approx$ 33,89%

[7] Source: [3]
[8] Source: [5]
[9] Source: [8]

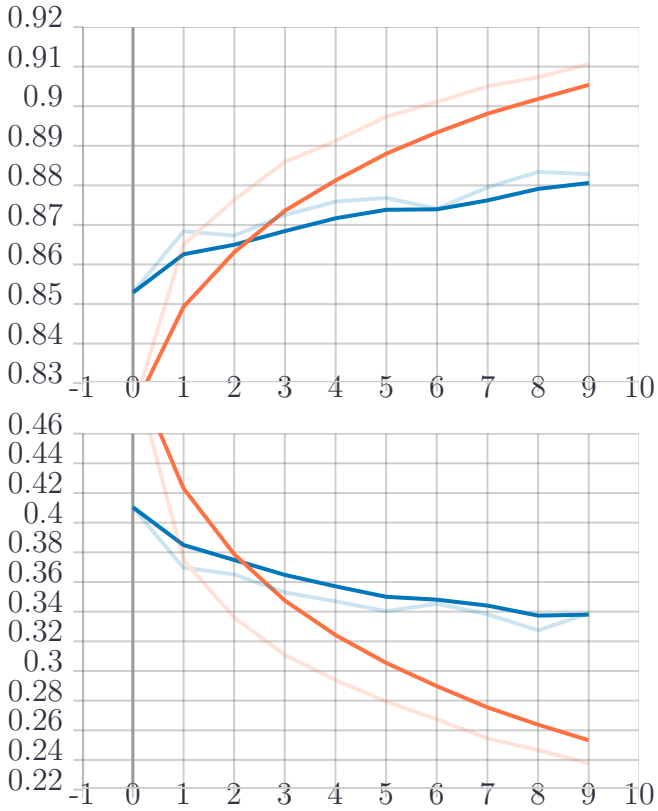
- How high the accuracy value in the last step was while training, here ≈ 88,28%

The difference between loss and loss value as well as accuracy and accuracy value lies within the different calculation methods. The first calculates how probable the desired outcome is and the ladder how accurate the individual output-nodes themselves are, without taking a collective average.
This than leads to a first evaluation, which outputs new information:

- How many steps there were, here 313
- The loss, here ≈ 33,89%
- The accuracy, here ≈ 88,28%

Then I ran a second evaluation with a more accurate result of ≈ 88,27999830245972% to verify the automated results.
Finally, TensorBoard is stared and because as mentioned before, there was continuous logging, there is data to see how the model changed over time.

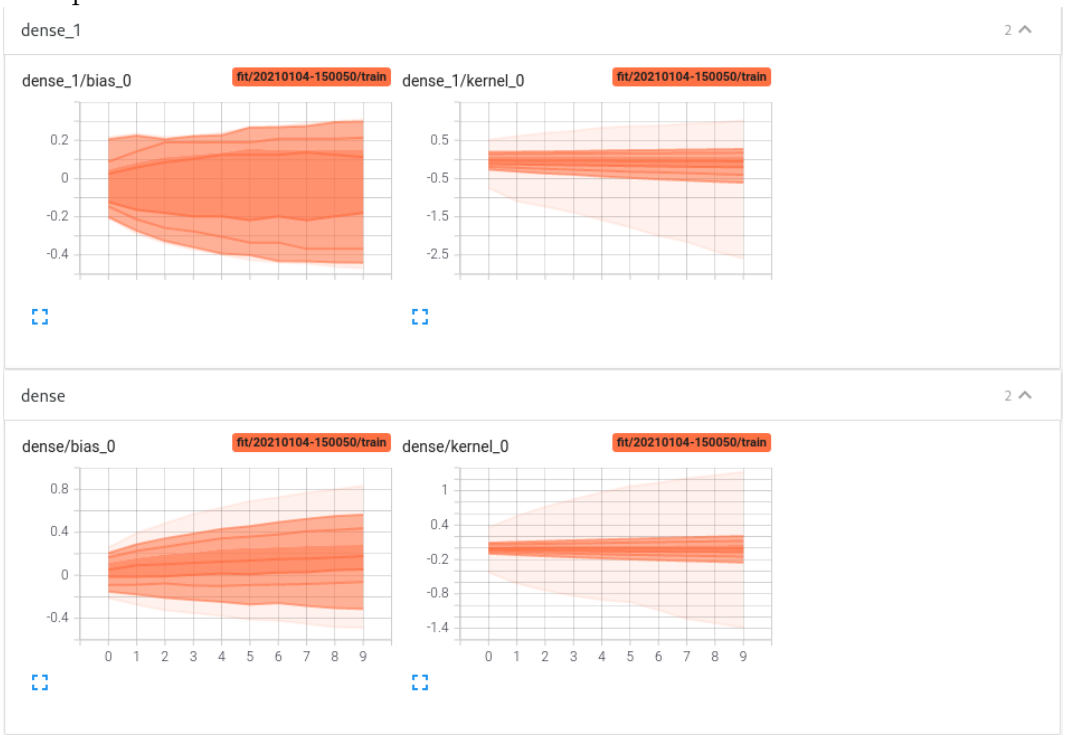
3.3.2 Graphical analysis



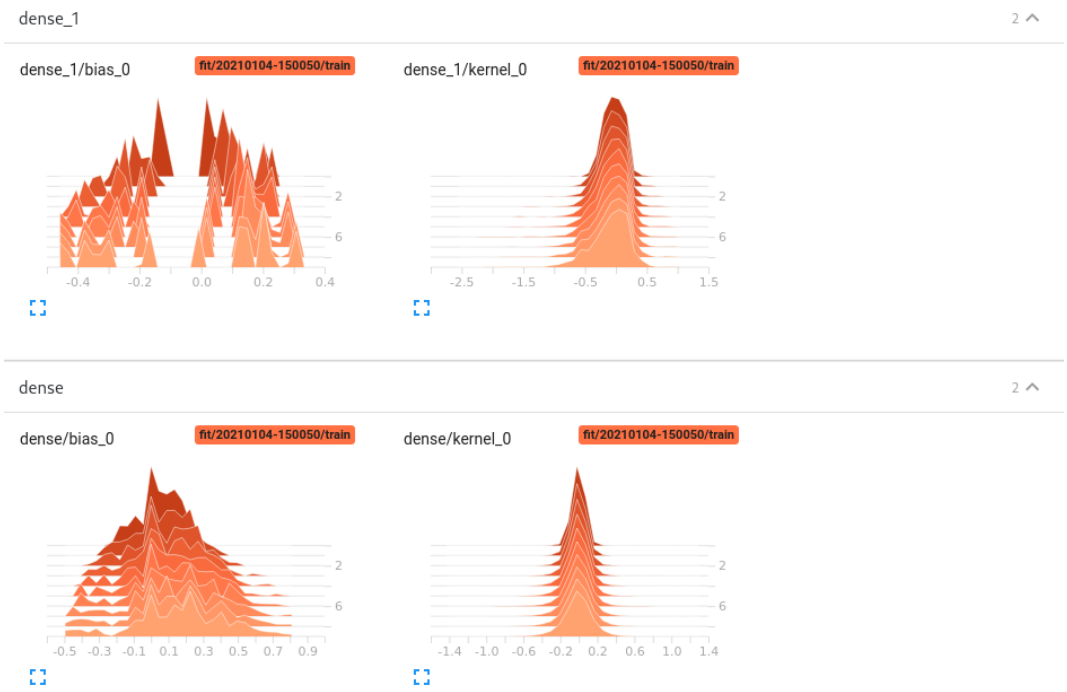
Generated with TensorBoard: epoch loss; x-axis epoch, y-axis accuracy / loss

In these figures multiple of the aforementioned effects can be seen. First of all a decline of improvement is clearly pronounced generation after generation.

This is due to a limit of how much improvement overall can be made. As this nears 0, the remaining gets smaller and smaller with each generation. Also interesting is that the accuracy on the training data is always improving, which isn't always the case with the testing data. This effect is especially pronounced in the lightly colored raw data. This is due to the model, not being able to train with unknown models, which is the point, it still averages an improvement after multiple epochs. Similar the opposite happens when it comes to the epoch loss which declines over time.



In the epoch distribution you can see what values are used in the weights of the model. As the model evolves, it gets more accurate and creates less random weights and more evenly spread out values. The values change less and less overtime as the model matures. In the epoch distribution you can see what values are used in the weights of the model. As the model evolves it gets more accurate and creates less random weights and more evenly spread out values. The values change less and less over time, as the model matures.



In the weight distribution, you can see how much the values from the weights of the model change. As the model evolves, it gets more accurate which leads to smaller adjustments. This can bee seen through the spikes getting smaller and more spread out as the model matures.

4 Conclusion

If you have a prefabricated dataset it has become quite easy to build a model that does what you want to do. It gets harder the more the model should do. We are not near stages where an artificial intelligence is viable. Even simple models may need hours of training to get great results. If this is then multiplied by hundreds of times for huge models, it is fairly easy to run out of processing power and small errors get multiplied hundreds or thousands of times.
In short as long as everything is strongly curated and predefined, it is possible for machine learning to do the job, but someone has to do these steps before given this preprocessed data to an algorithm.
In the example of MNIST only 28 x 28 pixel images of number in black and white and a brightness of between 0 and 1, can be given to the model for evaluation.

Humanity is nowhere near the intelligence as science-fiction authors have predicted, like Genesis in the Terminator franchise.

A. Definitions of terms

node	a single point in a model which takes inputs and creates a resulting output
layer	a collection of nodes which together make up one sub-advancement
weight	a value that can be tweaked in order to change the output of a node
model	the main part of the program with which we can learn
step	a single run of the model, after which the values will be changed
epoch	a set amount of steps after which evaluation can occur

B. Source code

```
# TensorFlow and tf.keras
import tensorflow as tf
# TensorBoard requirements
import datetime
import os
# Helper libraries for showing the image
import matplotlib.pyplot as plt

# Get TensorFlow version
print("TensorFlow Version:" + tf.__version__)

# Clear logs from previous runs
os.system("rm -r ./logs/")

# Get the MNIST fashion database and load it
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# define the names corresponding to the numbers for later classification
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal',
'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# prepare the images for training in range between 0 and 1
train_images = train_images / 255.0
test_images = test_images / 255.0

# Show the first 25 images with the corresponding name
plt.figure(figsize=(10, 10))
for i in range(25):
plt.subplot(5, 5, i + 1)
plt.xticks([])
plt.yticks([])
plt.grid(False)
plt.imshow(train_images[i], cmap=plt.cm.binary)
plt.xlabel(class_names[train_labels[i]])
plt.show()

# Set up layers
model = tf.keras.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)), # Transforms to Layer
tf.keras.layers.Dense(128, activation='relu'), # Layer with 128 Nodes
tf.keras.layers.Dense(10) # Layer with 10 Nodes, each one represents an out-
put
])

# Compile the Model
```

```
model.compile(optimizer='adam', # Optimizer, how the model is updated
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
# Loss function, measures how accurate the model is
metrics=['accuracy']) # Metrics, monitor training and testing steps

# Train The Model
# Set up the log directory for later analysis of the data
log_dir = "logs/fit/" + datetime.datetime.now().strftime("tensorboard_callback
= tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

# Fit the model to the training data
model.fit(train_images, train_labels, epochs=10, # Give Model the informa-
tion to fit it
validation_data=(test_images, test_labels), callbacks=[tensorboard_callback]
# add logging to monitor the Model
)

# Evaluate Accuracy
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(' \n Test accuracy:', test_acc)

# Load the TensorBoard Webserver
os.system("tensorboard –logdir=logs/ ")

# strongly based on Tensorflow examples
```

References

[1] 3Blue1Brown. *Simple neural network*. 5th Oct. 2017. URL: https://youtu.be/aircAruvnKk?t=232 (visited on 30/12/2020).

[2] aimlmarketplace.com. *What is the difference between AI,ML & DL*. 22nd Dec. 2017. URL: https://www.aimlmarketplace.com/read-ai-ml-blogs/what-is-the-difference-between-ai-ml-dl (visited on 30/12/2020).

[3] Multiple authors. *Basic classification: Classify images of clothing*. URL: https://www.tensorflow.org/tutorials/keras/classification (visited on 04/01/2021).

[4] Multiple authors. *Fashion-MNIST*. URL: https://github.com/zalandoresearch/fashion-mnist (visited on 04/01/2021).

[5] Multiple authors. *Get started with TensorBoard*. URL: https://www.tensorflow.org/tensorboard/get_started (visited on 04/01/2021).

[6] Multiple authors. *Introduction to Tensors*. URL: https://www.tensorflow.org/guide/tensor (visited on 04/01/2021).

[7] Multiple authors. *TensorFlow Hompage*. URL: https://www.tensorflow.org/ (visited on 04/01/2021).

[8] Multiple authors. *TensorFlow Releases*. URL: https://github.com/tensorflow/tensorflow/releases (visited on 04/01/2021).

[9] Multiple authors. *The sigmoid function*. 25th Dec. 2020. URL: https://en.wikipedia.org/wiki/Sigmoid_function (visited on 30/12/2020).

[10] Josef Steppan. *MnistExamples*. 14th Dec. 2017. URL: https://en.wikipedia.org/wiki/MNIST_database (visited on 30/12/2020).

[11] Akmel Syed. *AI vs. ML vs. DL (vs. NN)*. 20th Dec. 2020. URL: https://medium.com/a-coders-guide-to-ai/ai-vs-ml-vs-dl-vs-nn-f6968db769d1 (visited on 02/01/2021).

Work Report

Date	Time	Activity
15.11.2020	17:15 - 18:05	Subject formulation
27.11.2020	16:30 - 17:50	Writing the Structure
28.12.2020	08:30 - 14:25	Research
29.12.2020	08:30 - 15:35	Research
30.12.2020	08:30 - 13:50	Program development
30.12.2020	08:30 - 15:15	First testing
02.01.2021	08:30 - 15:05	Finalizing program
03.01.2021	08:30 - 14:45	Writing
04.01.2021	08:30 - 13:55	Writing
05.01.2021	08:30 - 15:20	Writing
06.01.2021	08:30 - 12:35	Finishing touches

Deceleration

I ensure that I have written the paper independently, that I have not used any sources or aids other than those indicated, and that any passages in the paper that I have taken from other works, either verbatim or in spirit, have been marked as borrowed, indicating the source in each case.

Ich versichere, dass ich die Facharbeit selbstständig verfasst, dass ich keine anderen Quellen und Hilfsmittel als die angegebenen benutzt und die Stellen der Facharbeit, die ich anderen Werken im Wortlaut oder dem Sinn nach entnommen habe, in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Solingen, den 06. Januar 2021

Nathan Mossaad