**Neural Networks**

Julia Hose

Julia.Hose@stud.leuphana.de

Matrikel-Nr.: 3044818

Karoline Tanck

Karoline.Tanck@stud.leuphana.de

Matrikel-Nr.: 3044699

**Table of contents**

1. The Idea
2. Dataset
3. Training, Validation and Test set
4. Preprocessing the data
5. Training the model
6. Conclusion
7. The Idea

You should never judge a book by its cover. So why not judge it by its title instead? While naturally this has to be taken with a grain of salt, it might actually be a job for a neural network to do exactly that. Books are very often sorted and recognized by title. The title is - together with the cover - supposed to catch a reader’s interest and get them to pick up the book, to start reading and in the end to buy the book. So, we thought a good title is probably essential for marketing a book and getting it to sell well. However, not only the amount of sold copies is important, but also the rating given by readers in the end. Therefore, we started to question, whether there is a connection or even correlation between the title of a book and its rating, e.g. specific words often used in the title of the best rated books. Going from there, we came up with the idea to build a neural network, that can rate real and made-up book titles.

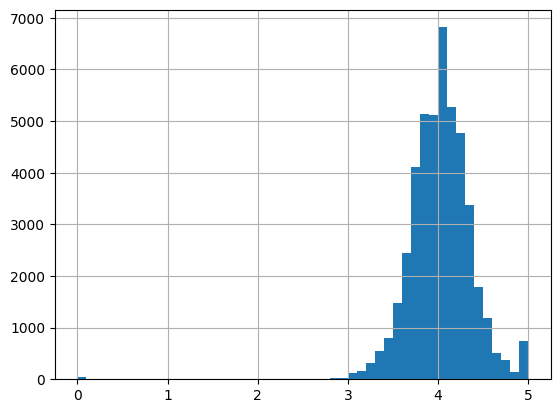
The general idea of the project is to train a network that can rate a book title with a float value between zero and five - comparable with an average star rating used by various accessible book rating providers like Goodreads, Storygraph or even Amazon. The former is where the data used in the project has been taken from. More information on the dataset itself will be presented in the next chapter. Naturally, it is very easy to deduce that there is no actual causal relationship between the title and the rating. Therefore, this might not have a high economic value - even though it might be interesting for marketing purposes -, it is simply fun and therefore has some artistic value. Furthermore, the first impression still counts, even when buying books, so the title might have an impact on the buying decision. Resulting from this, the project might be interesting to determine whether a title has good potential to be a bestseller title and help authors deciding on one. In conclusion, with a little imagination and a far stretch, it is possible to say the network might even have some use - aside from being a fun project, of course.

To complete the project, the network has been built, trained and tested with various test cases, some of them real and already existing books, some of them made up by us. The various steps that have been taken to accomplish that, are presented in the following. Furthermore, attempts to break the network were made and documented below.

1. Dataset

At the beginning of a project it is necessary to find data which was chosen for this project from the website Zenodo. The dataset is called “Best Books Ever Dataset“[[1]](#footnote-2) and features 52478 books scraped from the GoodReads Best Books Ever list. For each book a total of 25 columns provide vast amount of information about the work, such as title, rating, author, language, and genre – the first two of which were relevant for our project. In the following, the limitations of the dataset as well as the process of data cleansing are elaborated.

When choosing a dataset to work with, one must not only look at the benefits of the data, but also its limitations. The dataset we chose scraped the book ratings from Goodreads, an Amazon owned application for organizing and reviewing books.[[2]](#footnote-3) After completing a book, users can give it a rating out of five stars. Therefore, the ratings corresponding to the titles are not created by a higher instance like a committee but are very democratic and seemingly more reliable. However, one has to factor in that similar to many internet phenomena, hate or joke ratings are also given. Furthermore, some people tend to rate a book by their favourite author 5 stars, before ever having read the book. Thus, although one can assume that these biases in the data level out over the mass, they are important to keep in mind.

Moreover, the dataset we are using for our network also does not follow a logic in their selection of titles. The books don’t really have a logical relationship with each other. In our case, that can be seen as an advantage, as our network does not have a bias towards a certain genre of titles.

Lastly, the data skew is important to look at. The data is distributed evenly around the 3-star mark. Although this might seem like the data has a bias, that bias is rather natural. Since most people only read books they think will interest them, a minimum of 3 stars can be seen as guaranteed. From personal experience we can also say, that a lower star rating is pretty uncommon. However, the books that do have a very low star rating mostly don’t have many ratings at all. Although they are not known, the correlation between the title and the rating is still given as the title then just isn’t interesting enough. Especially books without any rating - which results in a rating of 0.0 stars in the datasheet - were discussed while preparing the dataset. Those books are that unpopular, that no-one has read them - or at least bothered to review them, which again speaks for itself. Therefore, it was decided to keep those books in the dataset, since we considered no rating a rating in itself.

Figure 1: Histogram of the rating column in the dataset

Before using the dataset, the data needs to be cleansed, which will be explained in the following. For the training of the network, the title and the rating of the books are the most vital parts of the dataset. Therefore, unnecessary columns have been dropped from the data frame, including the columns price, pages, awards, bbeVotes, bbeScore, isbn, characters, bookFormat etc. After that, only the columns rating, title, description and language were left in the data frame.

Furthermore, empty cells were inspected to find out how many there were per column. It was concluded, that neither the title column, nor the rating column had any missing values. Nonetheless, books without a description were deleted, on the assumption that there might be incorrect info on the book if there is no description, which is a very vital part of a book’s information. Furthermore, the dataset is of such a big size, that decision was made to better be safe than sorry in that case and delete the rows, since they only made a tiny dent in the size of the dataset. After that, the language column was the next to be worked with. First, all the unique values in the language column were inspected. Then, the individual languages were counted, to determine how many of them were not English. The non-English values made up a negligible amount, resulting in them being deleted from the dataset, as to not run into problems later on with different scripts or languages, as the dataset also included Arabic for example. Next, the rows without any values in the language columns were looked up. Since up to that point, quite a few rows had been deleted, the decision was made, to not delete those rows if possible. Hence, it had to be determined, whether the corresponding books were in English or not. To test that, the rows without a value in the language column were moved to a second data frame. Then, a for loop was used to check the description of all the books for the word “the“, since it is one of the most common words in the English language. Any books that did not have the word “the“ in the description was removed from the dataset. While the rows without a language were filtered in the second dataset, they were also removed entirely from the first one, as to not have any duplicate in the end. After the for loop had processed the whole dataset, the remaining rows where then joined onto the first dataset again.

Next, there also were some duplicates in the titles. After some discussion, it was decided, that duplicates should remain in the dataset, since there were different editions of books (for example a British and a US edition) or box sets, that include a whole series. These different editions, while the same in title, are different books and might therefore be rated differently, which is why they were kept in the dataset.

All of these steps were executed with help of the *pandas* library. The corresponding code can be found in the Jupyter notebook file belonging with this documentation.

After completing all the aforementioned processes, the data was then saved as a .csv file for further use, which will be explained in the following.

1. Training, Validation and Test set

For the proper training of the planned neural network the data needs to be separated into a training, a validation and a test set. The training set, being the largest, is to teach the network the correlations between the titles and the ratings of various books. With it, the networks is supposed to learn how to rate book titles. With the validation set, which is considerably smaller, the trained network can be checked to determine whether the training has been successful. This step can be repeated, after correcting the training process. Finally, the test set, which consists of data that has neither been used for training, nor for validation and is therefore entirely unknown by the network, is there to test the network and see whether everything works the way it is supposed to. Theoretically, the test set is the final set as it can effectively only be used once, since after a first test the test set is no longer unknown to the network. This is why there are a validation and a test set and not one set for both purposes. All three sets should be parts of the same original dataset, so that categories and source are the same. Therefore, the original „Best books ever Dataset“ from Zenodo had to be split into three parts. This was done using python’s *pandas* library in Jupyter notebooks again, to ensure fast checking of data skew.

To split the dataset into a training, validation and test set, sample sizes had to be determined. For the training set a size of 70% of the overall data was chosen, while the validation set consists of 20% and the test set of 10%. This distribution was chosen because we wanted to have a big enough training sample (70%), while also not having to little data to actually do the validating (20%) and testing (10%) with. With the method pandas random sample, random data can be put into a new data frame. After creating the training set, the used data was dropped from the original dataset, resulting in a split of the remaining data into the validation set (66.7% of the remaining data, equaling 20% of the overall data) and the test set (the rest). Since the amount of lower ratings (below three) is very low in the full dataset, histograms of each new dataset were created and compared to the histogram of the original dataset, to check whether the distribution of ratings was similar, as to not train, validate or test the data on a different distribution (see appendix). Since all three data sets passed that test, the data was further processed, which will be described in the following chapter.

1. Preprocessing the data

Before loading the data into the model, it has to be preprocessed, since the model itself very often cannot handle the data in its original form. This applies to our project as well. The data from the "best books ever Dataset“ is stored in a huge table and has different types. The book titles, for example, are stored as strings, which the neural network cannot work with. All steps described in this chapter are used on all three sets, the training, validation and the test set. Since the network expects to work with numbers, the words need to be encoded. Therefore, the strings in the title column first have to be tokenized, meaning that the strings - in this case the book titles - are being split into the individual words. To do so, the *torch* library has been imported and a tokenizer has been used, splitting the titles into smaller portions of the longer title. In a first attempt, n-grams have been used, that connect words in tokens to not only include the words but also their relationship. For example, the first and second word in the title would be a token, the second and the third are a token and so on. Although the inclusion of n-grams or alternatives like bi-grams leads to better results, due to problems that arose later on regarding the amount of data and the memory space needed for it, n-grams were not possible to include. Instead, each word is now a single token, and each token only contains one word. The used tokenizer put the tokens for each title in a corresponding list. Those lists were then put into a separate column in the data frame and the original title column was dropped. Therefore, the data frame now has a column with the rating of the book and one with the tokenized lists.

Since computers can only work with numbers and not text, in the next step the tokens needed to be encoded. Hence, the method of one hot encoding has been used: One hot encoding is a way to convert categorical variables like words into a format more fitting for machine learning.[[3]](#footnote-4) „The basic idea of one-hot encoding is to create new variables that take on values 0 and 1 to represent the original categorical values.“[[4]](#footnote-5) One-hot encoding is used, when the categorical variables do not have any ordinal relationships, meaning there is no order to the values. In the method, each variable is replaced by a binary variable. Each category has a new binary variable, all the other categories get a zero in the corresponding row. These binary variables are often referred to as „dummy variables“, especially in other fields like statistics.[[5]](#footnote-6) As a result, the data frame becomes a huge frame consisting of our binary encoded tokens, with each token now being a new column in the data frame and with binary variables describing the encoded tokens. Code wise, the *sklearn.preprocessing* library has been used to achieve this. From the library, the *OneHotEncoder* method has been used to encode the data. Both the library and the method were imported at the beginning of the python file. To encode the data, the tokenized data, that was earlier saved in a new file, is being imported into the program. Then, the target and the train data are being defined. Since the target data (the rating) is already numerical, it does not have to be encoded, leaving us to only encode the train data, consisting of the tokenized titles. In the encoder variable, the tokens are being encoded, the variable *handle\_unknown* is set to *‘ignore‘*, the variable *sparse\_output* is set to ’*True’*, meaning that sparsing is being active.

While this might sound simple at fist, different errors occurred while attempting to pre-process the data. First, the column with the unencoded tokens had to be removed from the dataset, meaning it had to be dropped with the *drop* function. This resulted in an error, since the device used did not have enough memory space to access the data frame and work with it. As mentioned before, the n-grams were then removed to downsize the data. After that still did not work, the next idea was to try using a different method altogether, so we tried the pandas get dummies method, which also did not work. Since that did not work either, we switched back to the original method in the *sklearn.preprocessing* library. After trying out several solution ideas, the way to go was going back to the very beginning and working with a smaller dataset all together. For that, half the dataset was randomly chosen, using a method from the *pandas* library. The data was then processed in the same way as described in chapter two. After that had worked, the sample size was set to 75%, to have a dataset as big as possible. After processing the data the same way as before, the 75% dataset also worked and was then used as the dataset for the model, because 75% of the dataset is still a huge amount of data and trying to narrow it down to the most data possible that can be used for the model would have been too time costly compared to its benefit. These steps were taken in Jupyter notebooks. The column with unencoded tokens was successfully removed and the data then ready to be further processed.

Furthermore, the labels each had to be transformed into integers, so they were comparable later.

Once the aforementioned steps had been taken, both the training and the target data were then transformed into numpy arrays, using the *np.array()* method. Then the arrays were further transformed into torch tensors, using *torch.Tensor()*. These tensors were then combined into a dataset, called *my\_dataset*, which was ready to be loaded into the data loader, called *train\_dataloader*.

After overcoming the problems and errors we ran into regarding memory space, we now have a tokenized and encoded data, ready to be fed into the model and to be used for training - or validation and testing, respectively. The training of the model will be explained in more detail in the next chapter.

1. Training the model

After cleaning the data, preprocessing it and splitting it into the three data sets, the actual training of the model was the next step that had to been taken.

Several steps were needed to train the neural network to be able to rate book titles in integer numbers.

5.1 Deciding on the model type

First, the data has to be loaded into the model. Therefore, as model type, a CNN - short for convolutional neural network - has been chosen. Even though CNNs are typically used for image recognition and processing, this type of neural network can also be used for texts. The use of a CNN was chosen over that of a RNN - short for recurrent neural network - for several reasons. A RNN has a short-term memory that helps with making a more accurate prediction. Said prediction is later compared with the true value using the loss function. Lastly, the gradients are being calculated through the use of error values in back-propagation. Because of its short-term memory, a RNN is often used for sequential data, for example in sentiment analysis, sequence labelling and speech tagging.[[6]](#footnote-7) In comparison, a CNN is a feed-forward, not a recurrent, neural network. That means that there are no loops or cycles. Instead, the information only flows in a forward direction. Every decision is always based on the current input, without any memory of any previous input. CNNs are therefore often used in classification and general regression problems.[[7]](#footnote-8) Another website also states, that CNNs can be used for text digitization and natural language processing, while also saying that CNNs are typically considered as the most powerful types of neural networks.[[8]](#footnote-9) In the case of this project, the goal is to classify data. The book titles that are inputted into the network are supposed to be classified into five six groups, the integer numbers from zero to five. Because of that, a CNN was chosen for the project, though at first glance a RNN might have been the obvious choice. In addition, since the encoding process was done without using n-grams, there is data connection between the various words in the titles. Therefore, the short-term memory of a RNN is not needed, since there is no use in remembering the previous inputs. Lastly, we felt more comfortable using a CNN, because we felt more secure in its use. In conclusion these are the arguments in favor of the CNN that ultimately resulted in us using this type of network for our project.

After that decision was made, the next question to be answered was which kind of CNN to use.

5.2 Using ResNet-50 and Keras

The first approached we tried to train our network was with keras, for wich we first imported some libraries like *keras.* There are lots of already built models present on the internet, so it is not always necessary to build one from scratch. While our first experience was with a resnet-18, which is used for image recognition, we decided to use a resnet-50 for our project as it was easier accessible with the libraries used. This is because we chose Keras, instead of PyTorch for our network and Keras is supporting resnet-50. Using resnet-18 with Keras has become available recently., but is still faulty, which is why we opted for resnet-50. It has a similar structure, with the numbers - 18 and 50 - representing the number of layers the data goes through. But before being able to actually apply the resnet-50 structure to our network, we had to add a dimension to our data. The original data is stored two-dimensional, but resent needs three dimensional data because it is primarily built for the use of images. To add another dimension, the *numpy* library was imported and with the *np.expand\_dims* another said dimension was added. For that, the axis was defined as -1. Then, another layer was added before the resnet-50 structure manually. It was assigned the name input\_layer, defining the shape of the input for the next layer.

This approach was mainly based on a tutorial found on kaggle, created by the user Tanmay Pandey, which we then altered for our use.[[9]](#footnote-10)

After all that, the training set kept erroring at various points. We tried lots of ways of fixing the problem, but in the end we came to the conclusion, that resnet-50 was simply too big for our purposes and was not fitting the problem.

Therefore, it was decided to build a new network from scratch, with less layers and a simpler architecture, meaning that we had to go several steps back and start anew.

5.3 Building our own network

When the Keras resnet-50 would not run, we decided to build our own network. It was decided to build this as a classification network with an output of six probabilities (corresponding to the numbers zero to five), which then compares the value with the highest probability with that of the given target label.

To achieve that, the self built network mainly consists of *linear layers, ReLUs* and a *softmax layer* before the output.

To build the network, a class that we called *BookClassification* was used. It consists of the two following functions. The *\_\_innit\_\_* function, that is used to instantiate all the layers and the model function, that calls the layers, and builds the network. The network we built consists of several *dense layers*, a *nn.ReLU()* activation function and a *softmax layer* at the end. Especially for the ReLU activation function, we did a lot of research on how to include it in our code. Very helpful for that was the website builtin.[[10]](#footnote-11)

As loss function *nn.CrossEntropyLoss()* has been used. And as optimizer *optim.Adam()*.

Lots of inspiration for this approach also came from the PyTorch website with its inbuilt forum, which especially helped us when researching some of the errors we ran into. User ptrblck had some very insightful answers.[[11]](#footnote-12) Another website, stating the most common errors occurring when using PyTorch, gave us a good approach to search for solutions after encountering various problems.[[12]](#footnote-13) Many of the errors we ran into had to do with the dimensions of the data that was being used. After a lot of Trial-and-Error and googling documentations, we were able to work out the remaining bugs and get the network to run smoothly. The solution for one of the problems for example, was to squeeze the label with the *squeeze()* method to reduce it to one dimension so it could be compared to the output value of the classification process.

With the built network, the training set then ran, but the training loss kept going up. Research provided us with various of possible reasons for that. One option, for example, was to change up the learning rate.[[13]](#footnote-14)

We then decided to experiment a little, with gradients and weights, but with no positive results. Sometimes the loss went up slower, sometimes faster, but unfortunately it never actually went down. We suspect the reason for this in the data. There are simply too little connections between the single words in a book title and rating of the book. Especially without using n-grams and a not so great data set, the network is simply not able to train well on the given data. Moreover, since the titles were very diverse but only consist of around five words maximum, we suspect that the encoded data is too sparse.

For the sake of completeness, we also tested the model with the testing set, resulting in the same bad performance and with an increasing loss as well.

1. Conclusion

All in all, while doing a bad performance on the given training set for the aforementioned reason, the model does run smoothly and the network does work. While that might not be a huge success generally speaking, it was one to us, since we accomplished to build a network from scratch and have it running in the end. In conclusion the project was a great learning opportunity for both of us. We broadened our knowledge on neural networks by trying out different approaches for solving our fictional problem. Furthermore, we learned a lot about data, how to handle and how to preprocess it.

In general, a huge problem in the whole process of doing the project was the fact, that whenever we googled a problem or error message we got, the tutorials or answers that could be found stopped where our problems started. Other times there were the same questions but no answers to them. Therefore, we had to overcome quite a few obstacles while working on the project. Especially steps like scrapping big parts of the code and starting anew, where very new and daunting.

1. Credits

While there are already various sources cited in the documentation, that have been very useful in the progress of this project, there are some more sources that need to be credited. First, the DataX course with its loads of useful material, which we used over and over again, especially when dealing with our data and the pandas library. Furthermore, the website Geeks for Geeks was very helpful during our research process. And last but not least the many consultation hours we had, which really helped us along in the process.

All in all, while the project in itself is our own, we are very grateful for all the resources we were able to use to help us along.

Bibliography

Beginner: Should ReLU/sigmoid be called in the \_\_init\_\_ method? (Last access: 14.03.2023) https://discuss.pytorch.org/t/beginner-should-relu-sigmoid-be-called-in-the-init-method/18689/2. [05.2018-01.2022]

Biswal, A.: Recurrent Neural Network (RNN) Tutorial: Types, Examples, LSTM and More (Last access: 13.03.2023) https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn#why\_recurrent\_neural\_networks. [14.02.2023].

Brownlee, J.: Why One-Hot Encode Data in Machine Learning? (Last access: 13.03.2023) https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/.[28.07.2017].

Editorial Team: Text Classification (Last access: 13.03.2023) https://towardsai.net/p/deep-learning/text-classification-with-rnn. [20.11.2020].

Goodreads Website (Last access: 13.03.2023) https://www.goodreads.com/.

Gupta, A.: Difference between ANN, CNN and RNN (Last access: 13.03.2023) https://www.geeksforgeeks.org/difference-between-ann-cnn-and-rnn/. [10.01.2023]

Krishnamurthy, B.: An Introduction to the ReLU Activation Function (Last access: 14.03.2023) https://builtin.com/machine-learning/relu-activation-function. [28.10.2022].

Lozano, L.C.; Planells, S.C.: Best Books Ever Dataset (Last access: 13.03.2023) https://zenodo.org/record/4265096#.Y\_TE5C1XYdU. [09.11.2020].

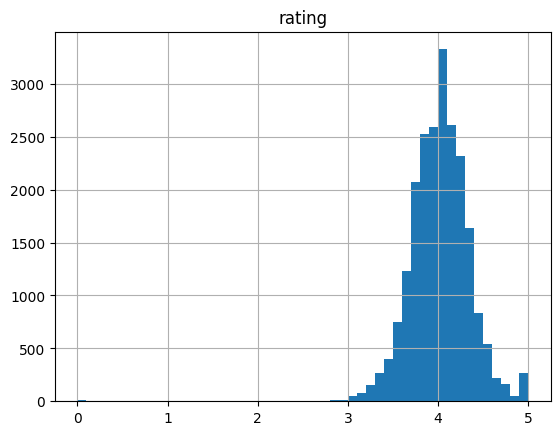
Pandey, T.: ResNet50- "notop" and normal model (Last access: 13.03.2023) https://www.kaggle.com/getting-started/63813. [2018].

Pramoditha, R.: How to Choose the Optimal Learning Rate for Neural Networks: Guidelines for tuning the most important neural network hyperparameter with examples (Last access: 14.03.2023) https://towardsdatascience.com/how-to-choose-the-optimal-learning-rate-for-neural-networks-362111c5c783. [21.09.2022].

The Three Most Common Errors in PyTorch (Last access: 14.03.2023) https://www.learnpytorch.io/pytorch\_most\_common\_errors/.[n.d.].

Zach: How to Perform One-Hot Encoding in Python (Last access: 13.03.2023) https://www.statology.org/one-hot-encoding-in-python/.[28.09.2021].

Appendix



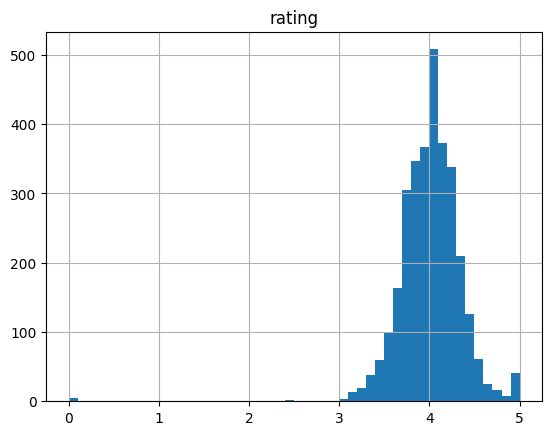
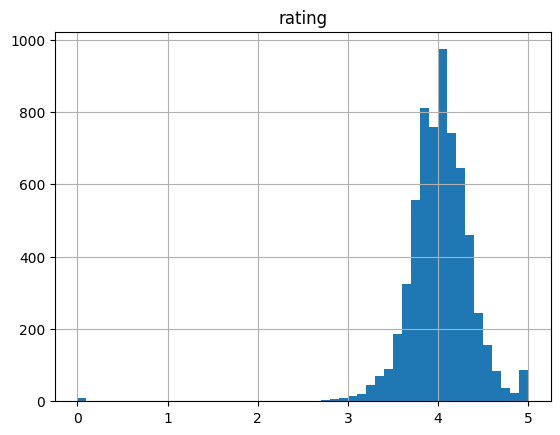


Figure 2: Histogram of the rating column in the training set

Figure 3: Histogram of the rating column in the validation set

Figure 4: Histogram of the rating column in the test set

Declaration of originality

We declare that the submitted academic paper has been written independently and no sources or aids other than those indicated have been used. All contents taken from the works of third parties which have been taken verbatim or in spirit from other sources have been marked as such appropriately and the respective source of the information has been clearly identified with precise bibliographical references. We have not submitted this paper before to an examining authority either in Germany or abroad.

1. cf. Lozano, L.C.; Planells, S.C.: Best Books Ever Dataset (Last access: 13.03.2023) https://zenodo.org/record/4265096#.Y\_TE5C1XYdU. [09.11.2020]. [↑](#footnote-ref-2)
2. cf.. Goodreads Website (Last access: 13.03.2023) https://www.goodreads.com/. [↑](#footnote-ref-3)
3. cf. Zach: How to Perform One-Hot Encoding in Python (Last access: 13.03.2023) https://www.statology.org/one-hot-encoding-in-python/.[28.09.2021]. [↑](#footnote-ref-4)
4. Zach: How to Perform One-Hot Encoding in Python (Last access: 13.03.2023) https://www.statology.org/one-hot-encoding-in-python/.[28.09.2021]. [↑](#footnote-ref-5)
5. cf. Brownlee, J.: Why One-Hot Encode Data in Machine Learning? (Last access: 13.03.2023) https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/.[28.07.2017]. [↑](#footnote-ref-6)
6. cf. Editorial Team: Text Classification (Last access: 13.03.2023) https://towardsai.net/p/deep-learning/text-classification-with-rnn. [20.11.2020]. [↑](#footnote-ref-7)
7. cf. Biswal, A.: Recurrent Neural Network (RNN) Tutorial: Types, Examples, LSTM and More (Last access: 13.03.2023) https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn#why\_recurrent\_neural\_networks. [14.02.2023]. [↑](#footnote-ref-8)
8. cf. Gupta, A.: Difference between ANN, CNN and RNN (Last access: 13.03.2023) https://www.geeksforgeeks.org/difference-between-ann-cnn-and-rnn/. [10.01.2023] [↑](#footnote-ref-9)
9. cf. Pandey, T.: ResNet50- "notop" and normal model (Last access: 13.03.2023) https://www.kaggle.com/getting-started/63813. [2018]. [↑](#footnote-ref-10)
10. cf. Krishnamurthy, B.: An Introduction to the ReLU Activation Function (Last access: 14.03.2023) https://builtin.com/machine-learning/relu-activation-function. [28.10.2022]. [↑](#footnote-ref-11)
11. cf. Beginner: Should ReLU/sigmoid be called in the \_\_init\_\_ method? (Last access: 14.03.2023) https://discuss.pytorch.org/t/beginner-should-relu-sigmoid-be-called-in-the-init-method/18689/2. [05.2018-01.2022] [↑](#footnote-ref-12)
12. cf. The Three Most Common Errors in PyTorch (Last access: 14.03.2023) https://www.learnpytorch.io/pytorch\_most\_common\_errors/.[n.d.]. [↑](#footnote-ref-13)
13. cf. Pramoditha, R.: How to Choose the Optimal Learning Rate for Neural Networks: Guidelines for tuning the most important neural network hyperparameter with examples (Last access: 14.03.2023) https://towardsdatascience.com/how-to-choose-the-optimal-learning-rate-for-neural-networks-362111c5c783. [21.09.2022]. [↑](#footnote-ref-14)