



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Martina Donadi

**A System for Sentiment Analysis of Online-Media with
TensorFlow**

*Fakultät Technik und Informatik
Studiendepartment Informatik*

*Faculty of Engineering and Computer Science
Department of Computer Science*

Martina Donadi

**A System for Sentiment Analysis of Online-Media with
TensorFlow**

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science European Computer Science
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. O. Zukunft
Zweitgutachter: Prof. Dr. W. Gerken

Eingereicht am: 31. July 2018

Martina Donadi

Thema der Arbeit

A System for Sentiment Analysis of Online-Media with TensorFlow

Stichworte

TensorFlow, Sentiment Analysis

Kurzzusammenfassung

Dieses Dokument stellt einen möglichen Ansatz für den Aufbau eines Sentiment-Analysesystems für die deutsche Sprache mittels TensorFlow- und menschenmarkierten Datensätzen vor. Diese Arbeit gibt eine Einführung in Konzepte des maschinellen Lernens und in TensorFlow und zeigt, wie man mit dem Werkzeug einen einfachen RNN erstellt. Der Schwerpunkt des Papiers liegt hauptsächlich auf den unterschiedlichen Ergebnissen, die bei der Verwendung unterschiedlicher Datensätze erzielt wurden.

Martina Donadi

Title of the paper

A System for Sentiment Analysis of Online-Media with TensorFlow

Keywords

TensorFlow, Sentiment Analysis

Abstract

This document presents a possible approach for the construction of a Sentiment Analysis system for the German language by means of TensorFlow and human-labeled data sets. This work gives an introduction to machine learning concepts and to TensorFlow and shows how to build a simple RNN with the tool. The paper's focus is mainly on the different results obtained from using different data sets.

Contents

1	Introduction	1
1.1	Goals	1
1.2	Overview	2
2	Basics	3
2.1	Machine Learning	4
2.2	TensorFlow	5
2.3	Requirements	9
3	System Architecture	10
4	Collecting Data	12
4.1	Scraping the Web	13
5	Analysis	15
6	Results	31
7	Conclusions and Future Work	35

1 Introduction

On top of each civil society holds the freedom of expression. According to the Universal Declaration of Human Rights "everyone shall have the right to freedom of expression; this right shall include freedom to seek, receive and impart information and ideas of all kinds, regardless of frontiers, either orally, in writing or in print, in the form of art, or through any other media of his choice". Although this right is not recognized as being absolute: "common limitations to freedom of speech relate to libel, slander, obscenity, pornography, sedition, incitement, fighting words, classified information, copyright violation, trade secrets, food labeling, non-disclosure agreements, the right to privacy, the right to be forgotten, public security, and perjury" ¹. On the other hand holds the current hyper-connected, hyper-social, hyper-informed society which produce everyday more content that one can possibly read in a life time. How can this content be analysed in a reasonable time if not with the help of computers? How can this content be used to make better business decisions? In my opinion, teaching a computer to understand the most human aspects of conversations, sentiment, is the key to understand and make a better use of large amounts of user content. I gladly invest my time on this topic because it is quite challenging and exciting at the same time. I hope to arouse the interest of this topic in others.

1.1 Goals

This paper aims to make some steps in the direction of building a sentiment analysis system for German Online Media. The main goal is to build a classifier for the analysis of users' comments of articles of an online newspaper with the help of TensorFlow and a human annotated dataset. Keeping in mind the difficulties in finding a complete and freely available annotated dataset for the German language, the accuracy of the analysis' results should be related not only to the method used but also to the dataset's size and content. A second purpose of this document is to give a brief introduction to Neural Networks with a special attention to Natural Language Processing (NLP) tasks. In addition, the paper presents a nice representation of the results by means of charts.

¹according to https://en.wikipedia.org/wiki/Freedom_of_speech

1.2 Overview

Let's give a little walk through the paper, pointing out what can be found. Chapter 2 ("Basics") offers an easy and straightforward introduction to machine learning related concepts and to the machine learning tool (TensorFlow) which is used to solve the sentiment analysis task. The chapter ends with a little discussion about the system requirements. A brief discussion over System Architecture follows. The following chapters give a description of the various steps to construct the whole system. Which data is analysed, how they has been collected and from where is pointed out in chapter 4 ("Collecting Data"). Chapter 5 ("Analysis") reports the discusses the measure of "goodness" of the results and gives an overview of the application of the analysis on the collected data. Finally chapter 7 ("Conclusions and Future Work") gives a summary of the goals achieved by the work and a critical discussion of what can be improved, suggesting possible further work to do.

2 Basics

Affect is the term by which linguistics defines the attitude or emotion that a speaker brings to an utterance. It can be manifested through facial expressions, gestures, written and spoken language. In written text, a central role of affect is played by metaphors such as "Time is a thief", "He is the apple of my eye" or "My wife's lawyer is a shark". However the expressions can not always be transformed, via comparison, to a literal meaning: "Time is like a thief", "He is like an apple for my eye" and "My wife's lawyer is like a shark". Sometimes discriminating between literal and non-literal meaning can be non trivial. For instance, the request "Can you pass me the salt" can be literally interpreted by answering "Yes I can", but it is self evident that the speaker does not need to question the interlocutor's ability to pass the salt. Nevertheless the ever increasing democratic access to media produces a cumbersome amount of text and speech which contains subjective content. This huge quantity of subjective content can not easily be analyzed by humans in a reasonable amount of time if at all. The need to gather and recognize users' attitude about a specific topic in huge volume of news has led to new computer-based systematic analysis approaches which go under the term "Sentiment Analysis". At this point at least one question, between many others, should possibly come to mind: How can sentiment be measured ?. Secondly: How can a computer system be trained to extract sentiment from text content? This paper's goal is to try to give a possible answer to these questions, providing a reasonable solution to the problem and a measure of the "goodness" of the analysis' results. The General Inquirer system Stone und Hunt [26] developed by Philip Stone in the mid-twentieth century is to be rewarded between one of the pioneeristic work in Sentiment Analysis. It is a content-analysis system which offers a dictionary of affect words, discriminating 82 categories of affect. It has been developed to analyse speeches by politicians and it is nowadays still in use. Harvard University made the system available online for academic research purposes ¹, merging the original dictionary with the Harvard-IV-4 and Lasswell dictionaries. Before the advent of many content-analysis tools, determining affect would have required the collaboration of many different professionalists. Currently research in the field is quite populated, offering different tools and models that one can use to perform

¹<http://www.wjh.harvard.edu/~inquirer/>

any task without having a deep understanding of the problem to be solved or knowledge in the specific area. Although there is no privileged model nor tool assuring granted quality of the results: it is important to measure the model's accuracy when constructing one. Before moving forward to the system's construction, let's try to have a reasonable understanding of what Machine Learning actually is.

2.1 Machine Learning

According to Tom Mitchell "A computer is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E ." This means that it is not sufficient to gather a big amount of data on a computer to make it possible for it to learn from them. An algorithm and a measure are needed to perform the learning process. Based on the way the learning algorithm works, different way of learning can be described:

- Supervised, Unsupervised or Semi-supervised learning
- On-line or Batch learning
- Instance-based or Model-based learning.

A learning algorithm generally combines some of the mentioned methods to better fit the learning problem. Supervised, Unsupervised or Semi-supervised learning are related to whether or not the learning process is carried on with human supervision. In Supervised learning the algorithm is fed with training data along with the desired result. A typical example is a classification task in which some input data have to be labeled with the name of the class they belong to. Some of the well known supervised learning algorithms are:

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines(SVMs)
- Decision Trees and Random Forests
- Neural Network

In Unsupervised learning the training data is passed to the learning algorithm without the desired results. A common example of an unsupervised task is association rule learning. This task consist of discovering relationships between large amounts of data. For example an online shop's owner looking at the logs of his customers may discover that customers who buy sheets of paper also tend to buy printer cartridges. Semi-supervised learning, as one can easily guess, is a combination of supervised and unsupervised learning, where in general the labeled data (training data along with their results) is a small part of the training dataset fed to the algorithm.

The main distinction between Batch and On-line learning holds in the fact that a Batch learning system is incapable of learning incrementally. This means that all the available data must be fed to a batch learning system, resulting in a very time and computing resources consuming process. Besides after the learning process is finished, the system is ready for production, but in case that new data arrives it is necessary to go through the whole learning process again. Thus if your system has limited resources and needs to learn incrementally, online learning would be a much better option. In online learning, data is fed in mini-batches, that is sequentially, either individually or by small groups. The learning process is thus divided into fast and cheap steps, allowing the system to learn on the fly. It is very important to set correctly the learning rate in online learning systems. The learning rate tells how fast the system should adapt to incoming new data and how fast to partially forget what it has already learnt. Finally what distinguishes Instance-based learning from Model-based learning is the way these systems generalize from the samples in the training set to predict new instances. In other words, it is the way the learning process is carried on: whether to compare new data to known data or to build a predictive model based on the training data. Instance-based systems learn all the examples in the training set by heart and use a similarity measure to generalize to new cases. Model-based system instead builds a model of the training examples and then use the model to make predictions. A great tool that allow to build Model-based learning systems is TensorFlow.

2.2 TensorFlow

TensorFlow [1] is a cross-platform software library for numerical computation with data-flow graphs. It has been developed by the Google Brain Team and released under the open source license². It runs on GPUs, CPUs and on some specialized hardware for tensor math, called Tensor Processing Units (TPUs). A distributed execution engine implemented in C++ provides

²TensorFlow was open-sourced in 2015.

a high performance core for the TensorFlow platform. On the top of the execution engine there are two frontends, in Python and C++ respectively. TensorFlow provides several high-level APIs, which make it possible to develop products for almost any kind of environment (desktop, web, mobile, cloud, etc.). Besides this, it is possible to set up different machine learning models to perform several different tasks, such as object detection, language translation, speech recognition , etc and to visualize the model through a suite of visualizing tools called Tensorboard.

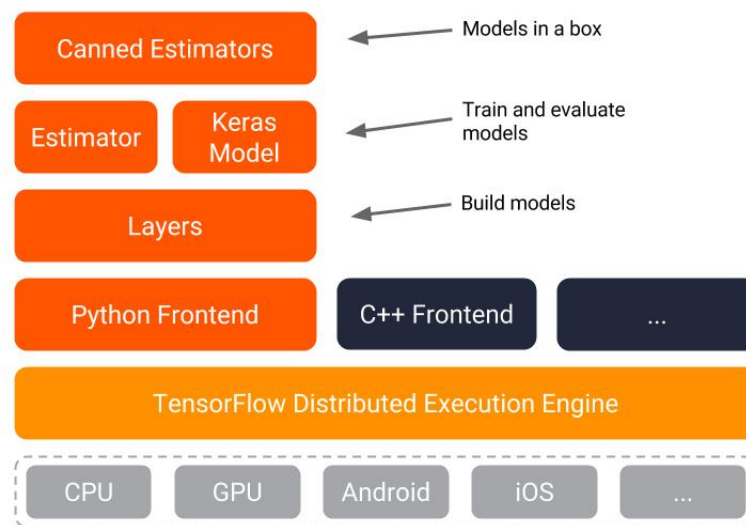


Figure 2.1: TensorFlow Architecture (Image by Google.com)

A popular high-level API is Keras, which provides building blocks to create and train deep learning models. Using an high-level API,like Keras, makes the production significantly easier because they are:

- *user friendly*: Keras gives clear feedback to the user, making error detection much easier during the development phase
- *modular and composable*: models are built by composing configurable blocks together
- *easy to extend*: it is always possible to modify a model by adding new layers or loss functions.

Data-flow graphs consists of nodes and edges, where each node represent an operation (implemented by kernels that run on GPUs,CPU or TPUs) and values flow along the edges. In the TensorFlow computational model [2] the values of interest are tensors: arrays specified at

graph-construction time. Tensors are handled through TensorFlow variables. A TensorFlow variable is essentially a reference to a tensor and it can be passed to operations that read or update the tensor it refers to. Edges are not only responsible for the flow of values, but there are also particular control edges that constrain the execution order. The execution of a data-flow graph, specified through a frontend language, usually ends with values on the output edges of the graph. A graph represents a training step for a machine learning model, where its parameters are stored in tensors and handled by variables. A model is typically a graph layers assembled together. The simplest way to assemble layers is to stack them. Such a model is called Sequential model. By means of a Sequential model it is possible to build a Recursive Neural Network (RNN). Before going into RNN details, let's take a step back and try to answer another question: what are Neural Networks? An Artificial Neural Network (ANN) is a computational data model that models linear and non-linear relationships between data and learn these relationships from the modeled data. An ANN is designed to simulate the way in which the human brain processes information: it learns, by means of training, from experience. ANNs consist of hundreds of single computational units (artificial neurons) connected with coefficients (weights) and organized in layers. Artificial neurons that are used are the so-called sigmoid neurons. Before sigmoid neurons, another type of artificial neurons were in use: perceptrons, which were first introduced by the scientist Frank Rosenblatt [18]. At its core, a perceptron is a computational unit for binary classification. In principle it seemed to be able to compute and execute any task, but Marvin Minsky and Seymour Papert [13] soon revealed its truly limited nature: a perceptron is incapable of producing XOR functionalities. The model of sigmoid neuron uses the following learning procedure: $z_i = \text{sigmoid}(\sum w_i x_i + b)$ where sigmoid is $\text{sigmoid}(z) = 1 / (1 + e^{-z})$. The behavior of a Neural Network is determined by its structure, its transfer function (which transfers processed data from one unit to another unit) and its learning rule. The weights are adjustable parameters: the sum (or a more complex activation function) of the weights of the inputs to a neuron constitutes the activation signal of a neuron which is passed through the transfer function to produce the neuron's output. In this sense, a neural network is a parameterized system.

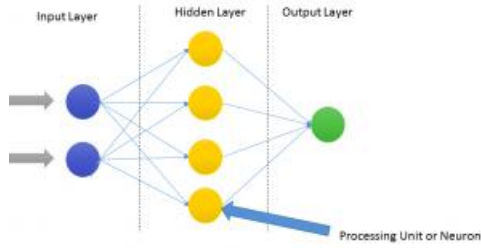


Figure 2.2: Simple Neural Network ^a

^a(Image by DnI Institute)

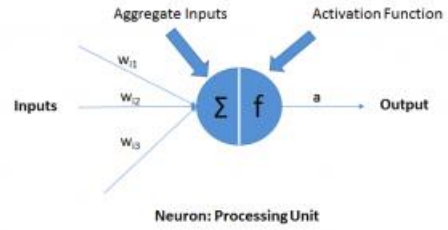


Figure 2.3: Neuron Processing Unit ^a

^a(Image by DnI Institute)

There are two largely used neural network models: Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). The former is typically a feed-forward network, mostly applied in signal and image processing field. To better understand what a convolution [4] is, one can think of a sliding function applied to a matrix. An image's filter works exactly this way: it slides over the pixels of an image generating a convolved value for each of the pixels. A Convolutional Neural Network consists of several layers of convolutions with non-linear activation functions (like ReLU or tanh) applied to the results. A CNN differs from a simple feed-forward network because it does not have just fully connected layers: not every input neuron is connected to each output neuron in the next layer. Each layer applies different convolutions and then combine the results. Recurrent Neural Networks differ from Convolutional Neural Network in that RNNs allow information to persist: they have memory. The way this memory is kept is by means of loops. An RNN can be seen as multiple copies of the same network, each passing an input to the next. In this sense, RNNs are related to sequences and lists. The building block of an RNN is a Recurrent Unit (RU). The simplest form of RU is a Gated

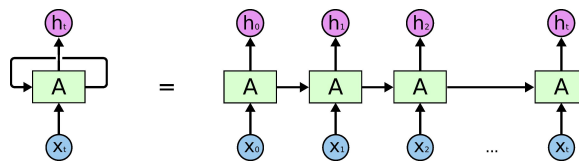


Figure 2.4: Unrolled RNN (Image by C. Olah [15])

Recurrent Unit (GRU), which was first introduced by Cho, et al. [5] in 2014. GRU aims to solve a common problem for RNNs: the problem of the vanishing gradient. When trying to train a Neural Network using gradient based optimization techniques, the model's accuracy degrades due to the long training time. In particular RNNs tend to use backpropagation algorithm in the learning process: the gradients of loss with respect to weights are calculated by moving

backwards in the network. The computed gradients tends to get smaller and smaller by moving backwards and the gradients vanish in the process. This means that neurons in earlier layers learn really slowly. Earlier layers in a RNN are very important because the earlier layers are the very foundation of the network. If the earlier layers give inaccurate results, they may affect the entire network, in the sense that the Prediction Accuracy of the model decreases due to the long training process. In the field of text classification, both CNNs and RNNs are largely applied, but the main advantage of RNNs is that they allow to model temporal sequences with variable length which is a better fit for the analysis of reviews of different lengths. RNNs have proved to be effective in many NLP tasks (see for example [5], [12],[9]) and according to [3], those based on GRUs seem to be the best choice in terms of accuracy.

2.3 Requirements

The system as a whole is completely implemented with Python 3.5.2 with the help of several libraries, easily installable via the pip command on the shell. The RNN is implemented using the Python frontend of TensorFlow and with the help of numpy and the Keras API. Numpy stands for Numerical Python. The numpy package provides high performance for scientific computing and data analysis, in particular for arrays operations, via a space-efficient multidimensional array representation module called ndarray. In addition, the package provides tools for integrating code written in C/C++. A nice introduction to Numpy's arrays can be found in chapter 4 of [29]. Keras, as already mentioned, is a high-level neural networks API that runs on top of TensorFlow and it has been designed with a focus on enabling fast experimentation. The official documentation can be found at <https://keras.io/>. For the retrieval of data from the Web the Python's libraries Requests, BeautifulSoup (bs4) and re are being used. These libraries are very user friendly and a quick reference can be found online. Once the libraries are installed it is possible to see the address of the site of their official documentation typing on the Python shell `import < libraryName >` and then `help(< libraryName >)`; for example, by typing `help(bs4)`, the following url is listed: <http://www.crummy.com/software/BeautifulSoup/bs4/doc/>. The data collected from the chosen online media is stored in MongoDB. MongoDB, version 2.6.10, is installed on localhost and accessed via the Python API pymongo (for a short reference see [14]). Finally Plotly is used for the visualization of the results.

3 System Architecture

For simplicity all the required software is installed on localhost. TensorFlow is accessed through the Python frontend. TensorFlow builds and trains the RNN, that is used for the text classification task. Python retrieves data from the Web with the help of Requests and BeautifulSoup, cleans them using re and stores them in mongoDB via pymongo. Subsequently Python retrieves data from mongoDB pass them to TensorFlow to classify them via the RNN and writes back the classification's result in the database. Once the classification task is completed mongoDB queries are passed to Plotly, which is connected to Phyton via the plotly module, to elaborate the visualization of the results. The scripts¹ are organized in packages as shown in the following components diagram :

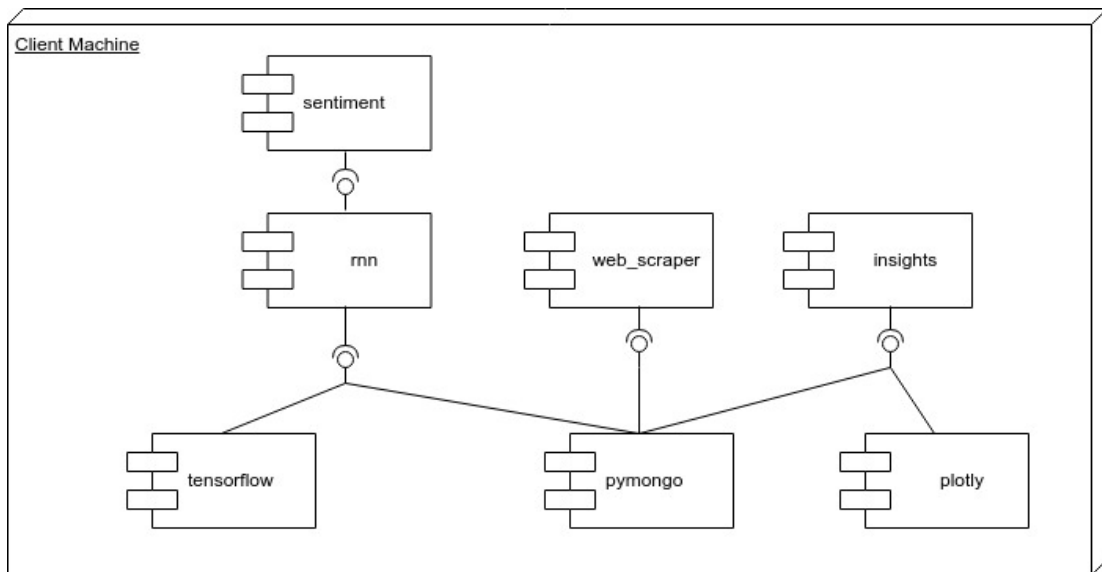


Figure 3.1: Component Diagram

The package rnn contains the script for building the RNN (train_and_save.py) and the script to run the classification task (classifier.py). The package on the top is the sentiment package

¹available at <https://github.com/gitEmme/webScrapper>

which contains the script to run the classification on the whole database (`rnn_classifier.py`). The scripts responsible for collecting data from the chosen website, `request.py` and `process_saved_links.py`, are contained in the `web_scraper` package with the script for loading the retrieved data to mongoDB (`load_to_db.py`). The RNN is trained with two different data set as later discussed. These different data set are cleaned and relabeled via the script `utilities.py` (in the `rnn` package) and are passed to the `save_and_train.py` script together with other parameters (such as activation and loss functions) described later on in the Analysis section. The technologies used in the system interact according to the diagram below.

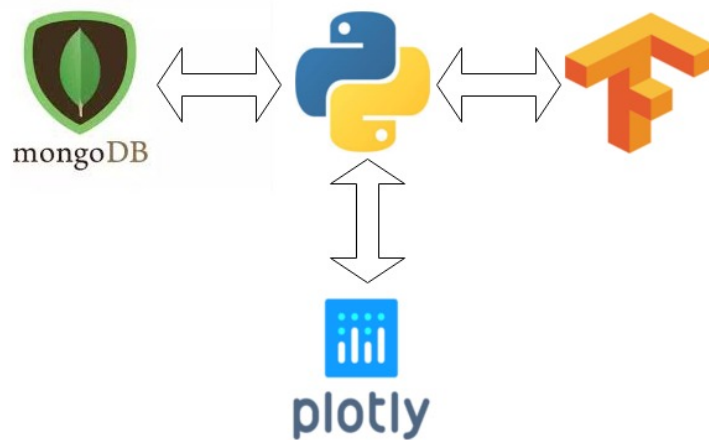


Figure 3.2: System Architecture

4 Collecting Data

The first step in building a system for Sentiment Analysis is to gather data. Without data, the system can not be tested or built at all. The data set to be collected has to be big enough to make sense for the system to exist: the system should overcome human speed limits in content analysis. According to Lucia Moses' article "*How The New York Times moderates 12,000 comments a day*" posted on DIGIDAYS UK on June 19th 2017¹ at the New York Times, a team of 14 part- and full-time moderators review 12000 comments per day. Thus millions of data can be considered a good order of magnitude for the system's purpose. Next establishing which data, from where and how to collect them are the required prerequisites to fulfill the data collection task. The Internet is a huge source of data of any kind, therefore the possibly huge dataset will be collected by scraping the Web. The chosen target site is *www.spiegel.de*². The content of interest is not the whole web site, but the user-produced content: the comments posted under articles in the related Forum sections³. Last important decision to be made is how to collect the data. The user content is obtained from the target site through a script that queries an internal database and paginates the content in the Forum sections in a reader-friendly manner. A "next page" button at the end of each page allows the reader to access the comments in the next page. Some Forum sections contain more than 100 pages of comments so that the retrieval of the content has to be automated somehow. Now some mathematicians could engagingly argue that according to the *Infinite Monkey Theorem*⁴ a monkey could absolve the task by reproducing all the wanted data, except for the fact that the process may never end. On the other hand, computer scientists may train a monkey to press the next button at the end of each page and copy-paste all the content, or in a more smarter way implement such a monkey.

¹<https://digiday.com/media/new-york-times-moderates-12000-comments-day/>

²The site's choice is a will to experiments further on a challenging and exciting project I have been taking part during my first exchange semester at the Haw Hamburg.

³<http://www.spiegel.de/forum/>

⁴The *Infinite Monkey Theorem* is a special formulation of the second Borel-Cantelli Lemma. It states that if you have a monkey hitting keys at random on a typewriter keyboard then, with probability 1, after an infinite time, he will type the complete works of William Shakespeare.

4.1 Scraping the Web

In this section the implementation of a Web scraper is described, that is, by analogy to a well-trained monkey. The Python programming language offers some very powerful tools for accessing and retrieving contents on the Web, such as the libraries Requests, BeautifulSoup and re. The Requests library make it easy to access web content over HTTP. The official online documentation presents the library through a witty remark: "Requests is the only Non-GMO HTTP library for Python, safe for human consumption"⁵. It is used to retrieve HTML pages within a given url. The obtained HTML pages are then parsed through BeautifulSoup. BeautifulSoup is a Python library for pulling content out of HTML and XML files. The library can be used to extract content enclosed between HTML tags. Using the inspection tool from Google Chrome browser it is possible to identify the opening and closing tags that wrap the comment sections and the tags wrapping the next page button link. In the case of <http://www.spiegel.de/forum/> the HTML element `<div id="threadlist">` contains the links to commented articles in each of the 12 news categories: Politik, Wirtschaft, Panorama, Sport, Kultur, Netzwelt, Wissenschaft, Gesundheit, Karriere, Leben und Lernen, Reise and Auto. All these links have been retrieved and saved into pickle files between March 19th and March 20th. One could wonder why the retrieval has been done in just two days. Well the answer has not to deal with speed, rather the opposite. Some of the mentioned sections contain more than 2000 pages of links to commented articles so that the retrieval process is quite slow. On average the time required to retrieve the links on each page is around 0.6 seconds. Now multiplying that for 2000 pages the time required to retrieve the links is more or less 20 minutes. Then those links have to be accessed again, one by one, to collect the comments which makes this process time consuming. With these considerations in mind and the fact that comments are also paginated under each of the retrieved commented article, the saved linking material should be sufficient for some weeks of work on a single laptop. Further, as mentioned before, some articles have more than 100 pages of comments, each containing on average 9 comments. For example, under the articles in the "Politik" section, while heavily underestimating the count of comments per article, more than 8 millions comments⁶ would be collected. For the purpose of the system, only the user-produced content in the first 6000 links to articles of each section have been collected, for a total amount of

⁵<http://docs.python-requests.org/en/master/>

⁶On March 20th the saved links to articles under the "Politik" section were 49680; considering around 20 pages of comments, each containing around 9 comments, the total amount of comments in "Politik" would be 8942400.

3124614 comments. The following map shows the amount of comments collected per topic:

```
{ 'auto': 423994,  
  'gesundheit': 189903,  
  'karriere': 159435,  
  'kultur': 239369,  
  'lebenundlernen': 433485,  
  'netzwelt': 214553,  
  'panorama': 167591,  
  'politik': 434068,  
  'reise': 111998,  
  'sport': 94445,  
  'wirtschaft': 411562,  
  'wissenschaft': 244211 }
```

With the help of regular expressions, it is possible to have a bit of insight on the collected data. For example, under the section "auto", 236 comments mention "Audi" (matching one of the forms "Audi", "audi" or "AUDI"), 307 comments mention "BMW" (matching one of the forms "BMW", "Bmw", "bmw"), 646 comments mention Volkswagen (matching one of the forms "Volkswagen", "Vw", "VW") and 38 comments mention "Ferrari" (matching one of the forms "Ferrari", "ferrari", "FERRARI"). Another interesting insight is that of the mentioned politicians under the "politik" section, 1213 comments mention "Merkel", 2106 mention "Trump", 2 mention "Berlusconi" and 313 mention "Macron". It is possible to play around with regular expressions to get some numbers, but this is more meaningful when considering the comments' sentiment. It is important to point out that these numbers are related to the small amount of data (about 3 million spread across 12 sections), that is retrieved from the website.

5 Analysis

The RNN implementation is based on the implementation provided by Magnus Erik Hvas Pedersen on github [16]. The data set "One Million Posts: A Data Set of German Online Discussion" [21] is used as training set to build the RNN. First the data set has been downloaded and filtered only for data that contains a sentiment annotation. Even though the title seems promising, once the data set is opened and cleaned, the actual amount of posts labeled with a sentiment annotation are 3589. Of these 3589 comments, only 43 are labeled as "positive", 1691 are labeled as "negative" and 1855 are labeled as "neutral". The title and body of each post are put together because the title of a post can be reasonably considered to be part of the post as well. After removing all of the extra spaces and break lines in the text, the posts are ready to be transformed into values to be fed to TensorFlow. These values are tensors. The generation of these tensors requires several steps. First, all the words present in the data set are collected to create a dictionary. This is done via Keras' Tokenizer module, which breaks down raw-text into the so-called tokens and maps every word in the set of tokens to an integer value. In this way, each word in the data set is uniquely identifiable through an integer number. The map allows to transform each post into an array of integers. Now the RNN can take as input sequences of arbitrary length. However, the sequences need to have the same length in order to use the whole batch of data. For this reason, the average number of tokens in a post is computed and each post's token sequence is padded or truncated to have the same length. Padding is made at the beginning of the sequence. For example, if the sequence [1, 33, 567, 5769, 6, 7, 29, 4, 67, 4, 73, 5] represents a post and the length of each post, based on the computed average number of 100, a certain number of 0s is added at the beginning of the sequence to make it 100 numbers long. In the case of the training data set in use the average length of the training set is 103. The maximum number of tokens that a sequence can have is actually computed as the average plus 2 standard deviations. This way, the computed value covers 95% of the data set ($0.95 * 0.80 * 3589 = 2727.64$ samples). These words to integers vector mappings are technically called word2vec representations. Subsequently, it is necessary to transform these integer-tokens sequences into embeddings: real-valued vector. Embeddings can be fed to the Neural Network during the training phase. Posts in the data set have been

re-labeled with integers to reflect the class that the post belongs to. 0 stays for Positive class, 1 stays for Neutral class and 2 stays for Negative class. These class numbers are passed to Keras' `to_categorical()` function that transform the class numbers into a vector shape, which is easily interpretable by TensorFlow for a classification task. The RNN consists of Gated Recurrent Units as provided by the Keras implementation. A GRU has an internal state that is updated with every new incoming input. The GRU saves floating-points values (typically between -1.0 and 1.0) in its internal state, which are read and written via matrix operations. In this sense the internal state represents some kind of memory in the recurrent unit. New state-values depend on the old state-values and the current inputs. The way of producing new state-values is by means of a gate which is a type of matrix operation. A GRU has normally two gates: one to update the internal state and one to produce an output value. TensorFlow trains the recurrent units by gradually changing the weight-matrices of the gates. The internal state of a GRU

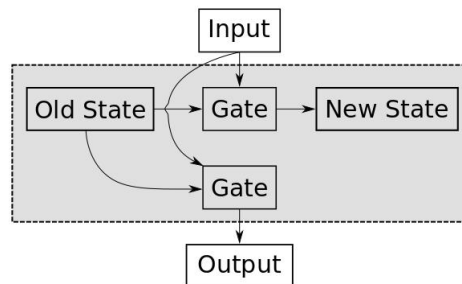


Figure 5.1: GRU (Image by Hvass-Labs)

is set to 0 every time a new sequence of input words begins. Each word in a sequence (the word's numerical mapped value indeed) is passed through the GRU in a series of time-steps. When the entire sequence has been processed, the GRU outputs a vector of values. This vector summarizes what is contained in the input sequence. A fully connected layer with a softmax activation function is then used to get a three dimensional array with values between 0.0 and 1.0, one for each possible sentiment class. The classification of a comment which can be either positive, negative or neutral, is then determined by the maximum score in the array. The RNN consists of:

- an Embedding-layer, which converts each integer-token into a vector of floating-point values
- three layers of GRU: the first outputs 16 units, the second outputs 8 and the third outputs 4 units.

- a fully-connected Softmax layer that outputs a three dimensional array with values in the range 0.0 and 1.0.

The Embedding-layer is trained as part of the network in order to learn how to map words with similar semantic meanings to similar embedding-vectors. The size of the embedding vectors is set to 8 because it seems to work well with small values for Sentiment Analysis tasks (according to [16]). To train the weights inside the recurrent units while avoiding the vanishing gradient problem, it is necessary to minimize some loss function. A loss function measures the difference between the actual output and the desired output. The used loss function for our model is the categorical cross-entropy function. The Adam Optimizer is added to the model to minimize the loss function and to see the model's learning rate. The Adam algorithm (presented by Diederik Kingma and Jimmy Ba [11] in 2014) is an extension of the stochastic gradient descent algorithm that can be used to iteratively update the network's weights. The main difference between the Adam algorithm and the stochastic gradient descent is that Adam maintains a learning rate for each network weight (parameter) and not a single learning rate for all weight updates. Sebastian Ruder, who developed a comprehensive review of modern optimization methods [19] in 2016, recommends using the Adam optimization method. The data set is split into training and test sets according to a 80-20 percentage split. After compiling the defined model with Keras, it is possible to see the model summary though the summary() method:

```
-----
Layer (type)                Output Shape                Param #
-----
layer_embedding (Embedding) (None, 103, 8)             155000
-----
gru_1 (GRU)                 (None, 103, 16)           1200
-----
gru_2 (GRU)                 (None, 103, 8)            600
-----
gru_3 (GRU)                 (None, 4)                  156
-----
dense_1 (Dense)             (None, 3)                  15
-----
Total params: 156,971
Trainable params: 156,971
Non-trainable params: 0
```

The training process has been repeated for 1000 epochs and produced an accuracy of 51.81%. For brevity a short verbose output of a training for just three epochs is reported below:

Train on 2727 samples, validate on 144 samples

Epoch 1/3

```
64/2727 [.....]
- ETA: 12:07 - loss: 1.0828 - acc: 0.4062
128/2727 [>.....]
- ETA: 6:15 - loss: 1.0571 - acc: 0.4766
192/2727 [=>.....]
- ETA: 4:19 - loss: 1.0561 - acc: 0.4427
256/2727 [=>.....]
- ETA: 3:19 - loss: 1.0486 - acc: 0.4453
320/2727 [==>.....]
- ETA: 2:43 - loss: 1.0400 - acc: 0.4500
384/2727 [===>.....]
- ETA: 2:20 - loss: 1.0347 - acc: 0.4401
448/2727 [===>.....]
- ETA: 2:02 - loss: 1.0273 - acc: 0.4420
512/2727 [====>.....]
- ETA: 1:48 - loss: 1.0206 - acc: 0.4551
576/2727 [====>.....]
- ETA: 1:37 - loss: 1.0135 - acc: 0.4635
640/2727 [=====>.....]
- ETA: 1:28 - loss: 1.0063 - acc: 0.4734
704/2727 [=====>.....]
- ETA: 1:20 - loss: 0.9994 - acc: 0.4744
768/2727 [=====>.....]
- ETA: 1:14 - loss: 0.9935 - acc: 0.4779
832/2727 [=====>.....]
- ETA: 1:08 - loss: 0.9873 - acc: 0.4772
896/2727 [=====>.....]
- ETA: 1:02 - loss: 0.9813 - acc: 0.4810
960/2727 [=====>.....]
- ETA: 56s - loss: 0.9755 - acc: 0.4875
1024/2727 [=====>.....]
- ETA: 52s - loss: 0.9715 - acc: 0.4912
1088/2727 [=====>.....]
- ETA: 49s - loss: 0.9675 - acc: 0.4899
1152/2727 [=====>.....]
```

- ETA: 45s - loss: 0.9630 - acc: 0.4852
1216/2727 [=====>.....]
- ETA: 42s - loss: 0.9585 - acc: 0.4901
1280/2727 [=====>.....]
- ETA: 40s - loss: 0.9543 - acc: 0.4891
1344/2727 [=====>.....]
- ETA: 37s - loss: 0.9494 - acc: 0.4918
1408/2727 [=====>.....]
- ETA: 35s - loss: 0.9468 - acc: 0.4943
1472/2727 [=====>.....]
- ETA: 32s - loss: 0.9421 - acc: 0.4966
1536/2727 [=====>.....]
- ETA: 30s - loss: 0.9389 - acc: 0.4974
1600/2727 [=====>.....]
- ETA: 28s - loss: 0.9356 - acc: 0.4975
1664/2727 [=====>.....]
- ETA: 26s - loss: 0.9311 - acc: 0.5030
1728/2727 [=====>.....]
- ETA: 24s - loss: 0.9286 - acc: 0.5029
1792/2727 [=====>.....]
- ETA: 22s - loss: 0.9257 - acc: 0.5017
1856/2727 [=====>.....]
- ETA: 20s - loss: 0.9219 - acc: 0.5022
1920/2727 [=====>.....]
- ETA: 19s - loss: 0.9179 - acc: 0.5047
1984/2727 [=====>.....]
- ETA: 17s - loss: 0.9150 - acc: 0.5050
2048/2727 [=====>.....]
- ETA: 15s - loss: 0.9112 - acc: 0.5112
2112/2727 [=====>.....]
- ETA: 14s - loss: 0.9073 - acc: 0.5142
2176/2727 [=====>.....]
- ETA: 12s - loss: 0.9051 - acc: 0.5106
2240/2727 [=====>.....]
- ETA: 11s - loss: 0.9029 - acc: 0.5103
2304/2727 [=====>.....]
- ETA: 9s - loss: 0.9017 - acc: 0.5082
2368/2727 [=====>.....]
- ETA: 8s - loss: 0.8992 - acc: 0.5059
2432/2727 [=====>.....]

```
- ETA: 6s - loss: 0.8977 - acc: 0.5053
2496/2727 [=====>...]
- ETA: 5s - loss: 0.8951 - acc: 0.5072
2560/2727 [=====>..]
- ETA: 3s - loss: 0.8923 - acc: 0.5078
2624/2727 [=====>..]
- ETA: 2s - loss: 0.8898 - acc: 0.5107
2688/2727 [=====>.]
- ETA: 0s - loss: 0.8877 - acc: 0.5112
2727/2727 [=====]
- 64s 24ms/step - loss: 0.8864 - acc: 0.5101
- val_loss: 0.8071 - val_acc: 0.5278
Epoch 2/10
 64/2727 [.....]
  - ETA: 41s - loss: 0.8031 - acc: 0.3906
128/2727 [>.....]
  - ETA: 40s - loss: 0.7931 - acc: 0.4297
192/2727 [=>.....]
  - ETA: 39s - loss: 0.7881 - acc: 0.4583
256/2727 [=>.....]
  - ETA: 38s - loss: 0.7973 - acc: 0.4805
320/2727 [==>.....]
  - ETA: 36s - loss: 0.7930 - acc: 0.4906
384/2727 [===>.....]
  - ETA: 35s - loss: 0.7931 - acc: 0.5026
448/2727 [===>.....]
  - ETA: 34s - loss: 0.7893 - acc: 0.5067
512/2727 [====>.....]
  - ETA: 33s - loss: 0.7917 - acc: 0.5254
576/2727 [====>.....]
  - ETA: 32s - loss: 0.7924 - acc: 0.5226
640/2727 [=====>.....]
  - ETA: 31s - loss: 0.7898 - acc: 0.5219
704/2727 [=====>.....]
  - ETA: 30s - loss: 0.7873 - acc: 0.5241
768/2727 [=====>.....]
  - ETA: 29s - loss: 0.7869 - acc: 0.5312
832/2727 [=====>.....]
  - ETA: 28s - loss: 0.7896 - acc: 0.5312
896/2727 [=====>.....]
```


- ETA: 27s - loss: 0.7889 - acc: 0.5379
960/2727 [=====>.....]
- ETA: 26s - loss: 0.7915 - acc: 0.5354
1024/2727 [=====>.....]
- ETA: 25s - loss: 0.7898 - acc: 0.5322
1088/2727 [=====>.....]
- ETA: 24s - loss: 0.7924 - acc: 0.5276
1152/2727 [=====>.....]
- ETA: 22s - loss: 0.7902 - acc: 0.5304
1216/2727 [=====>.....]
- ETA: 21s - loss: 0.7888 - acc: 0.5304
1280/2727 [=====>.....]
- ETA: 21s - loss: 0.7870 - acc: 0.5320
1344/2727 [=====>.....]
- ETA: 20s - loss: 0.7875 - acc: 0.5298
1408/2727 [=====>.....]
- ETA: 19s - loss: 0.7874 - acc: 0.5305
1472/2727 [=====>.....]
- ETA: 18s - loss: 0.7872 - acc: 0.5299
1536/2727 [=====>.....]
- ETA: 17s - loss: 0.7868 - acc: 0.5260
1600/2727 [=====>.....]
- ETA: 16s - loss: 0.7869 - acc: 0.5256
1664/2727 [=====>.....]
- ETA: 15s - loss: 0.7896 - acc: 0.5234
1728/2727 [=====>.....]
- ETA: 15s - loss: 0.7880 - acc: 0.5243
1792/2727 [=====>.....]
- ETA: 14s - loss: 0.7866 - acc: 0.5257
1856/2727 [=====>.....]
- ETA: 13s - loss: 0.7857 - acc: 0.5242
1920/2727 [=====>.....]
- ETA: 12s - loss: 0.7859 - acc: 0.5234
1984/2727 [=====>.....]
- ETA: 11s - loss: 0.7869 - acc: 0.5227
2048/2727 [=====>.....]
- ETA: 10s - loss: 0.7864 - acc: 0.5254
2112/2727 [=====>.....]
- ETA: 9s - loss: 0.7859 - acc: 0.5223
2176/2727 [=====>.....]

```
- ETA: 8s - loss: 0.7850 - acc: 0.5207
2240/2727 [=====>.....]
- ETA: 7s - loss: 0.7838 - acc: 0.5223
2304/2727 [=====>.....]
- ETA: 6s - loss: 0.7835 - acc: 0.5260
2368/2727 [=====>....]
- ETA: 5s - loss: 0.7845 - acc: 0.5241
2432/2727 [=====>....]
- ETA: 4s - loss: 0.7837 - acc: 0.5230
2496/2727 [=====>...]
- ETA: 3s - loss: 0.7827 - acc: 0.5244
2560/2727 [=====>..]
- ETA: 2s - loss: 0.7814 - acc: 0.5277
2624/2727 [=====>..]
- ETA: 1s - loss: 0.7805 - acc: 0.5297
2688/2727 [=====>.]
- ETA: 0s - loss: 0.7810 - acc: 0.5264
2727/2727 [=====]
- 43s 16ms/step - loss: 0.7804 - acc: 0.5273
- val_loss: 0.7820 - val_acc: 0.5278
Epoch 3/10
 64/2727 [.....]
  - ETA: 38s - loss: 0.7494 - acc: 0.6875
128/2727 [>.....]
  - ETA: 38s - loss: 0.7425 - acc: 0.6250
192/2727 [=>.....]
  - ETA: 39s - loss: 0.7573 - acc: 0.5833
256/2727 [=>.....]
  - ETA: 32s - loss: 0.7926 - acc: 0.5430
320/2727 [==>.....]
  - ETA: 34s - loss: 0.7864 - acc: 0.5531
384/2727 [===>.....]
  - ETA: 32s - loss: 0.7893 - acc: 0.5521
448/2727 [===>.....]
  - ETA: 33s - loss: 0.7808 - acc: 0.5558
512/2727 [====>.....]
  - ETA: 32s - loss: 0.7828 - acc: 0.5469
576/2727 [====>.....]
  - ETA: 32s - loss: 0.7799 - acc: 0.5417
640/2727 [=====>.....]
```

```

- ETA: 32s - loss: 0.7762 - acc: 0.5391
704/2727 [=====>.....]
- ETA: 31s - loss: 0.7716 - acc: 0.5440
768/2727 [=====>.....]
- ETA: 30s - loss: 0.7706 - acc: 0.5469
832/2727 [=====>.....]
- ETA: 29s - loss: 0.7675 - acc: 0.5493
896/2727 [=====>.....]
- ETA: 28s - loss: 0.7664 - acc: 0.5469
960/2727 [=====>.....]
- ETA: 27s - loss: 0.7640 - acc: 0.5469
1024/2727 [=====>.....]
- ETA: 26s - loss: 0.7635 - acc: 0.5430
1088/2727 [=====>.....]
- ETA: 26s - loss: 0.7645 - acc: 0.5423
1152/2727 [=====>.....]
- ETA: 24s - loss: 0.7618 - acc: 0.5460
1216/2727 [=====>.....]
- ETA: 22s - loss: 0.7611 - acc: 0.5444
1280/2727 [=====>.....]
- ETA: 21s - loss: 0.7606 - acc: 0.5414
1344/2727 [=====>.....]
- ETA: 20s - loss: 0.7622 - acc: 0.5379
1408/2727 [=====>.....]
- ETA: 19s - loss: 0.7646 - acc: 0.5362
1472/2727 [=====>.....]
- ETA: 18s - loss: 0.7639 - acc: 0.5346
1536/2727 [=====>.....]
- ETA: 18s - loss: 0.7665 - acc: 0.5312
1600/2727 [=====>.....]
- ETA: 17s - loss: 0.7662 - acc: 0.5356
1664/2727 [=====>.....]
- ETA: 16s - loss: 0.7655 - acc: 0.5331
1728/2727 [=====>.....]
- ETA: 15s - loss: 0.7671 - acc: 0.5330
1792/2727 [=====>.....]
- ETA: 14s - loss: 0.7673 - acc: 0.5340
1856/2727 [=====>.....]
- ETA: 13s - loss: 0.7662 - acc: 0.5361
1920/2727 [=====>.....]

```

```

- ETA: 12s - loss: 0.7652 - acc: 0.5365
1984/2727 [=====>.....]
- ETA: 11s - loss: 0.7652 - acc: 0.5373
2048/2727 [=====>.....]
- ETA: 11s - loss: 0.7662 - acc: 0.5308
2112/2727 [=====>.....]
- ETA: 10s - loss: 0.7652 - acc: 0.5312
2176/2727 [=====>.....]
- ETA: 9s - loss: 0.7656 - acc: 0.5299
2240/2727 [=====>.....]
- ETA: 7s - loss: 0.7668 - acc: 0.5295
2304/2727 [=====>.....]
- ETA: 6s - loss: 0.7664 - acc: 0.5260
2368/2727 [=====>....]
- ETA: 5s - loss: 0.7666 - acc: 0.5279
2432/2727 [=====>....]
- ETA: 4s - loss: 0.7667 - acc: 0.5288
2496/2727 [=====>...]
- ETA: 3s - loss: 0.7668 - acc: 0.5284
2560/2727 [=====>..]
- ETA: 2s - loss: 0.7662 - acc: 0.5277
2624/2727 [=====>..]
- ETA: 1s - loss: 0.7654 - acc: 0.5282
2688/2727 [=====>.]
- ETA: 0s - loss: 0.7648 - acc: 0.5283
2727/2727 [=====]
- 46s 17ms/step - loss: 0.7644 - acc: 0.5266
- val_loss: 0.7748 - val_acc: 0.5278
  32/718 [>.....] - ETA: 3s
  64/718 [=>.....] - ETA: 4s
  96/718 [===>.....] - ETA: 4s
 128/718 [====>.....] - ETA: 4s
 160/718 [=====>.....] - ETA: 4s
 192/718 [=====>.....] - ETA: 3s
 224/718 [=====>.....] - ETA: 3s
 256/718 [=====>.....] - ETA: 3s
 288/718 [=====>.....] - ETA: 3s
 320/718 [=====>.....] - ETA: 2s
 352/718 [=====>.....] - ETA: 2s
 384/718 [=====>.....] - ETA: 2s

```

```
416/718 [=====>.....] - ETA: 2s
448/718 [=====>.....] - ETA: 1s
480/718 [=====>.....] - ETA: 1s
512/718 [=====>.....] - ETA: 1s
544/718 [=====>.....] - ETA: 1s
576/718 [=====>.....] - ETA: 1s
608/718 [=====>.....] - ETA: 0s
640/718 [=====>.....] - ETA: 0s
672/718 [=====>..] - ETA: 0s
704/718 [=====>.] - ETA: 0s
718/718 [=====] - 6s 8ms/step
Accuracy: 51.81%
```

By starting TensorBoard ¹ on the system's shell via the command `tensorboard --logdir <dir> --host <host>` it is possible to plot the learning process. TensorBoard allows to visualize the constructed model and plots some quantitative metrics about the execution of the graph. Since a network may contained millions of nodes the visualization of the graph is simplified. Nodes are grouped by name scopes and the dependencies are represented though arrows. Dotted lines represent control dependencies. Solid arrows represent the flow of tensors between operations (op). Another simplification is made by series collapsing: sequential nodes, which have isomorphic structure, are collapsed into a single stack of nodes. A short legend of the icons used in the graph model, the model itself, and the execution metrics are reported below:

¹see https://www.tensorflow.org/guide/summaries_and_tensorboard for a better understanding of how TensorBoard works










Symbol	Meaning
	High-level node representing a name scope. Double-click to expand a high-level node.
	Sequence of numbered nodes that are not connected to each other.
	Sequence of numbered nodes that are connected to each other.
	An individual operation node.
	A constant.
	A summary node.
	Edge showing the data flow between operations.
	Edge showing the control dependency between operations.
	A reference edge showing that the outgoing operation node can mutate the incoming tensor.

Figure 5.2: TensorBoard Icons (Image by TensorBoard)

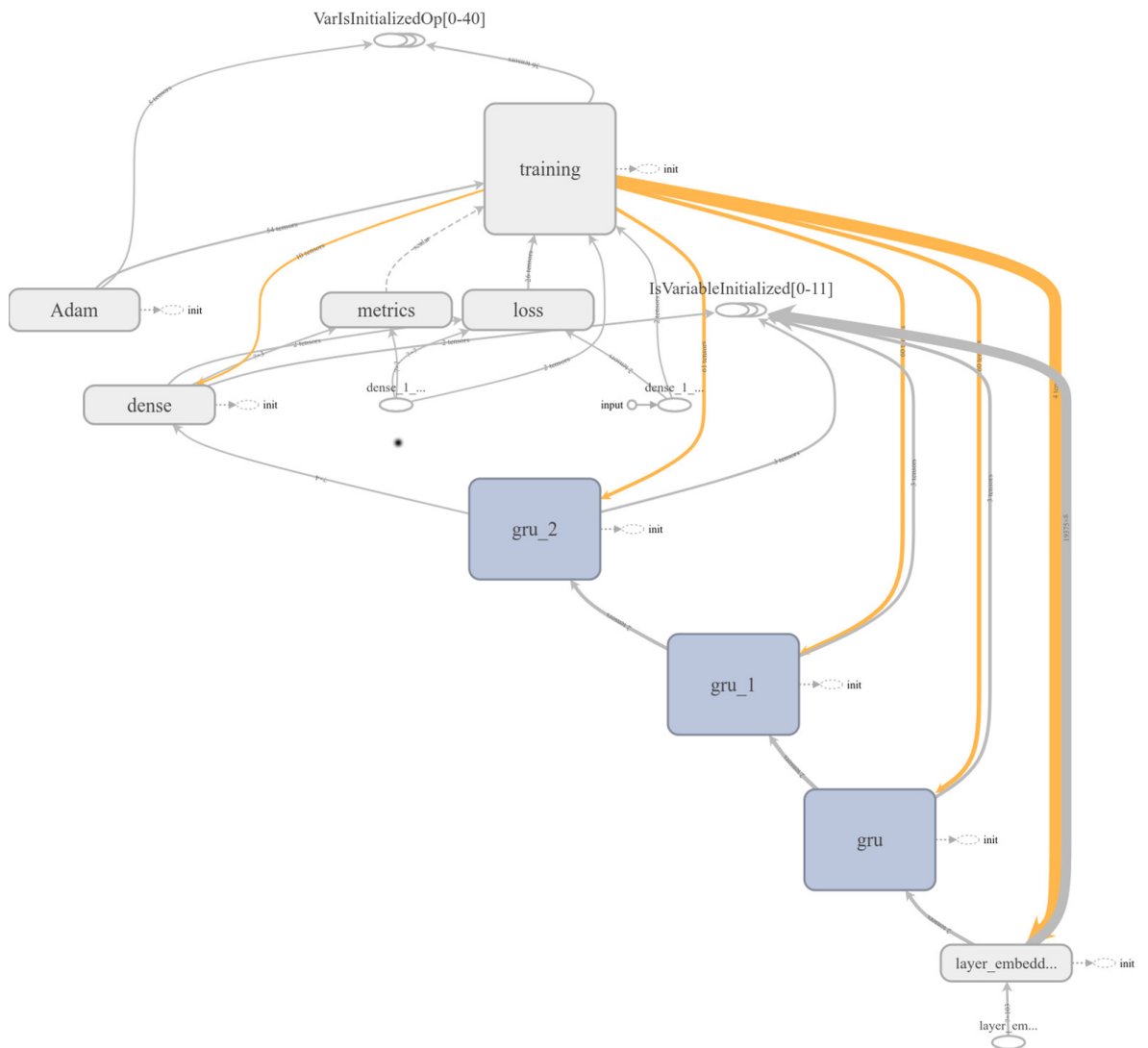


Figure 5.3: RNN Model(Image by TensorBoard): The model represented is used also for the training of a second RNN described later in the paper

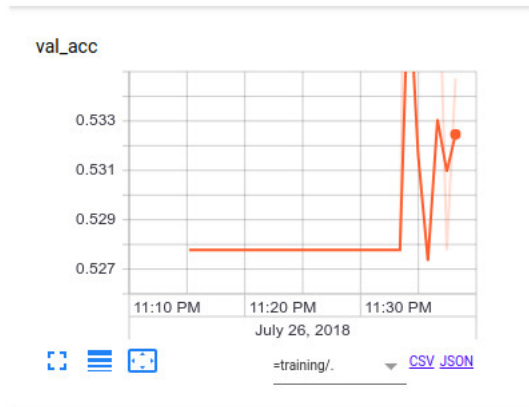


Figure 5.4: Validation Accuracy

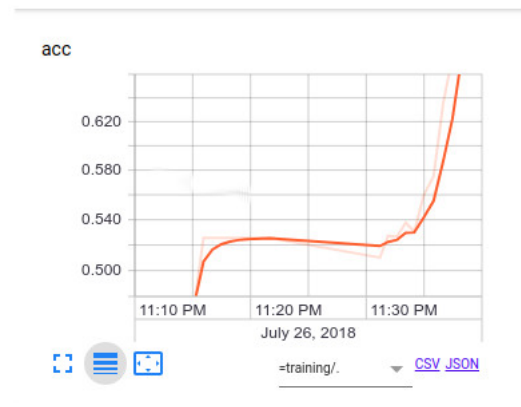


Figure 5.5: Accuracy

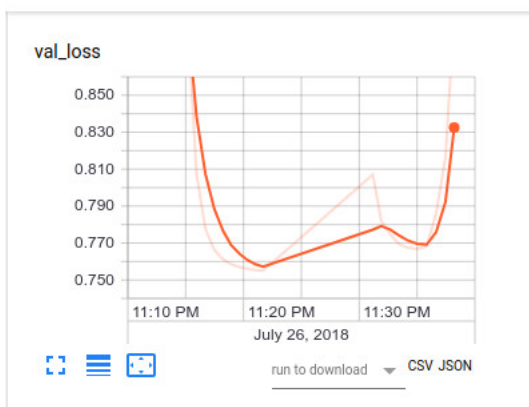


Figure 5.6: Validation Loss

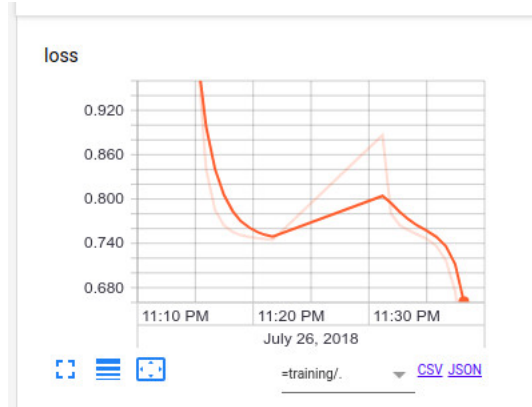


Figure 5.7: Loss

The accuracy of the trained RNN, even for a longer training process, is not satisfactory at all. The expression "almost better guessing" would be the perfect match to express the quality of the result. It has to be said that such a result was quite expected indeed. This is quite surely caused by the poor training data set used. With this in mind, a second RNN has been trained using a larger data set containing annotated customers reviews on the German public train operator Deutsche Bahn. The data set comes from the Germeval Task 2017 ("Shared Task on Aspect-based Sentiment in Social Media Customer Feedback") and can be download at <https://sites.google.com/view/germeval2017-absa/data?authuser=0>. The data set contains 22000 messages from various social media and web sources. The sum-

5 Analysis

mary of the model, which was constructed using the Germaneval Task 2017 data set, that is provided already split in train and test sets, is the following:

Layer (type)	Output Shape	Param #
layer_embedding (Embedding)	(None, 85, 8)	515816
gru_1 (GRU)	(None, 85, 16)	1200
gru_2 (GRU)	(None, 85, 8)	600
gru_3 (GRU)	(None, 4)	156
dense_1 (Dense)	(None, 3)	15

Total params: 517,787
Trainable params: 517,787
Non-trainable params: 0

This second model gives an accuracy of 71.21% after three training epochs, confirming the fact that the low accuracy obtained with the "One Million Posts" data set is largely dependent on the quality and the size of the data set.

Train on 18460 samples, validate on 972 samples

Epoch 1/3

2018-07-25 12:46:55.515057

18460/18460 [=====]

- 264s 14ms/step - loss: 0.8032 - acc: 0.6788

- val_loss: 0.7582 - val_acc: 0.6965

Epoch 2/3

18460/18460 [=====]

- 269s 15ms/step - loss: 0.7809 - acc: 0.6788

- val_loss: 0.7532 - val_acc: 0.6965

Epoch 3/3

18460/18460 [=====]

- 247s 13ms/step - loss: 0.7286 - acc: 0.6982

- val_loss: 0.6856 - val_acc: 0.7150

2369/2369 [=====] - 4s 2ms/step

Accuracy: 71.21%

Finally, the RNN can be used to classify the collected data from "Spiegel Online" by means of applying the Keras predict() function to the trained model. In short, we can view the whole classification process in the picture below.

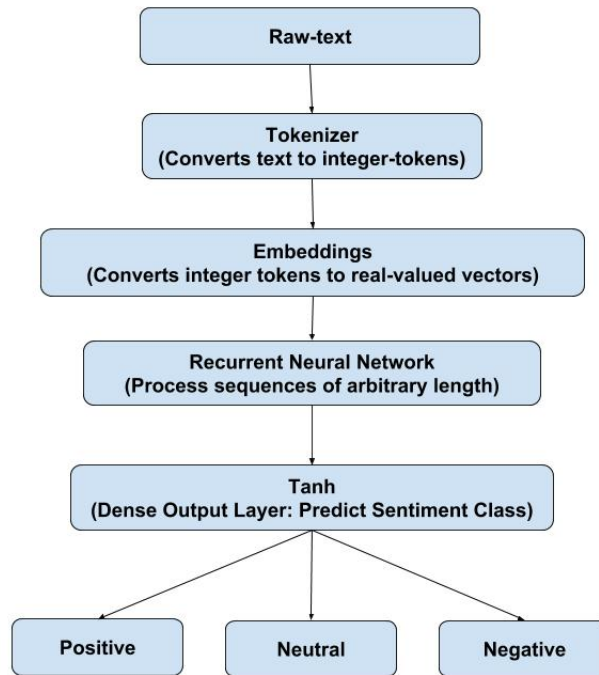


Figure 5.8: Flowchart of the classification process

6 Results

The accuracy results obtained by means of the two trained RNNs have to be strictly related to the data sets on which the two networks are trained. In the following discussion RNN_1 indicates the network trained on the data set "One Million Posts: A Data Set of German Online Discussion" [21] and RNN_2 indicates the network trained on the Germeval Task 2017 (Shared Task on Aspect-based Sentiment in Social Media Customer Feedback) data set¹. The dataset used for RNN_1 contains only 43 positive samples, 1691 negative samples and 1855 neutral samples for a total of 3589 samples. This data set is too little and the number of samples per class is badly balanced: there are too many negatives and neutral samples and too little positive samples (just 1% of the total). The accuracy on training is thus very little, with an accuracy of only 51.81%. When training the network for a longer time, the validation accuracy gets smaller than the accuracy on the training set causing overfitting in the model. The overfitting problem can be solved by adding some dropout layers to the network or validating on a bigger data set. RNN_2 gives a better accuracy result (71.21%), even if built in the exact same way of RNN_1 . This is related to the data set on which the model is trained. The train data set from the Germeval Task 2017 contains 19432 samples of which 1179 are positive, 5045 are negative and 13208 are neutral. Again, the number of positive samples is very little (6% of the total). Also for this network, longer training time leads to overfitting.

The vocabulary generated from the RNN_1 's data set is quite poor, even if the content News-related (the data set contains comments from Austrian news papers). In the process of converting raw text into tensors most of the words are lost because the words are not present in the data set's dictionary and thus they are not mapped to any value. The mapping of text to tensors is a bit better with the RNN_2 ' data set, even though this data set is not very News-related. Most of the samples are comments about traveling by train. The classification results obtained on the collected data from Spiegel Online for both the constructed networks (RNN_1 and RNN_2) are plotted below with the help of the plot tool Plotly.

¹<https://sites.google.com/view/germeval2017-absa/data?authuser=0>

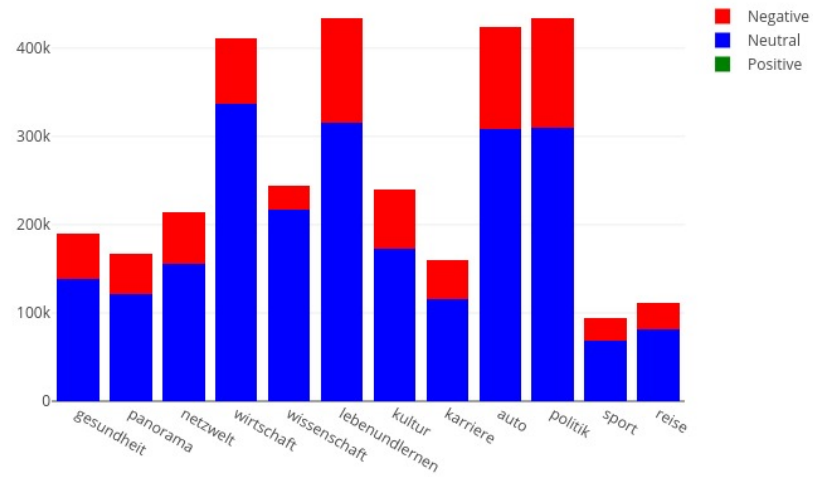


Figure 6.1: Sentiment per section with RNN₁

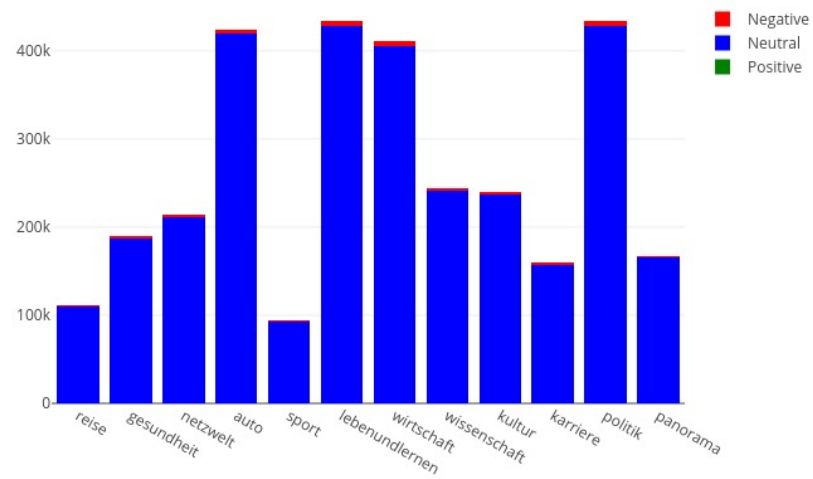
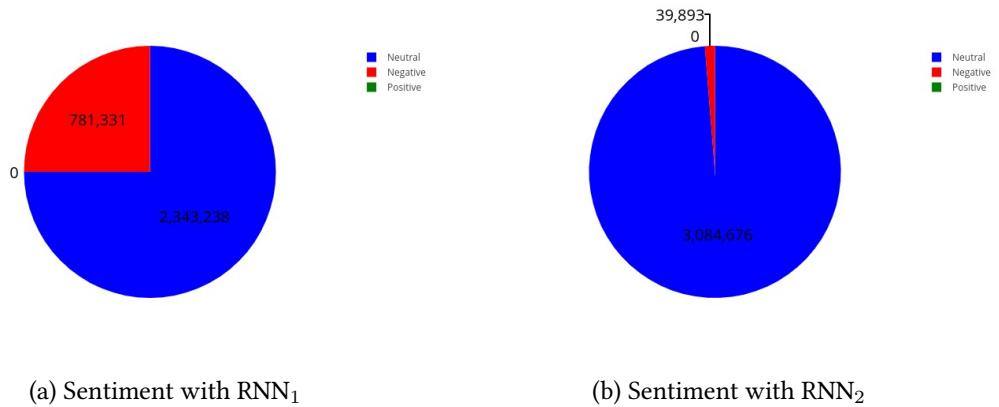
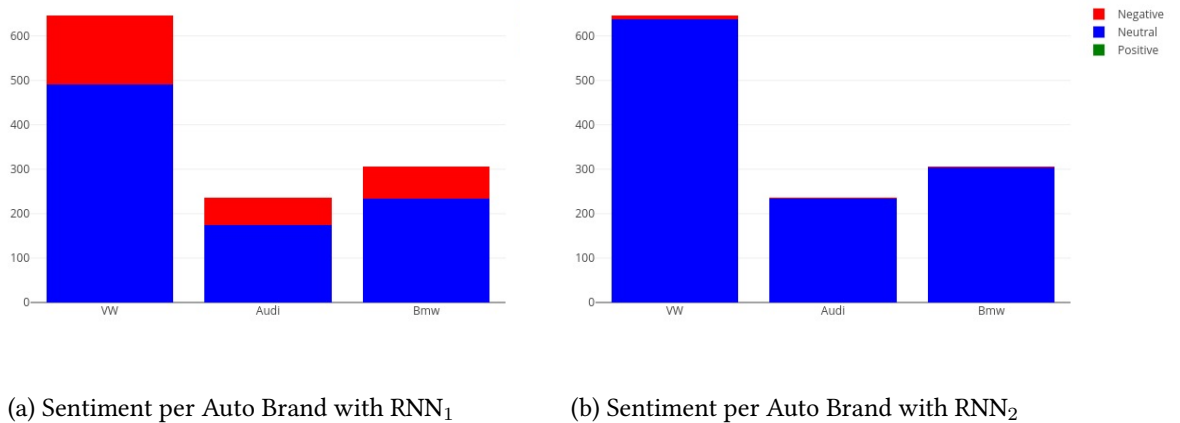


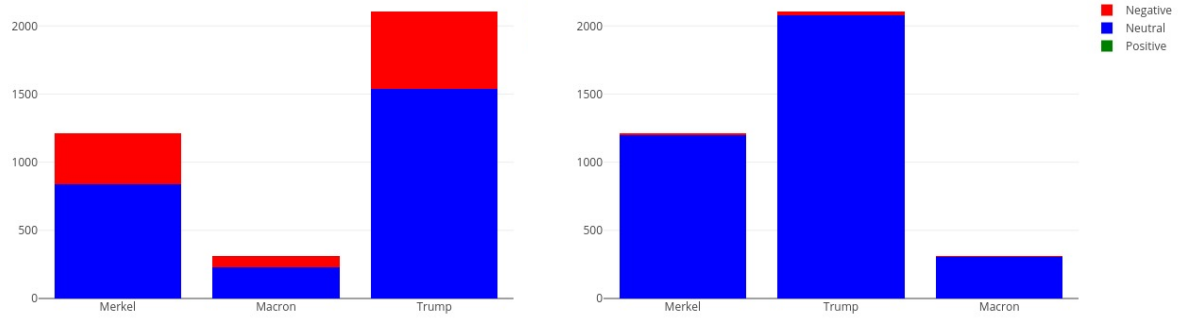
Figure 6.2: Sentiment per section with RNN₂



It is important to keep in mind that the classification is built over the concept of pattern matching and thus it is not related to any text comprehension: the network classifies text based on the samples it has seen and has no understanding of the semantics. None of the trained networks detected positives comments. This could be related to the very little amount of positive samples in the training set (1% for RNN_1 and 6% for RNN_2): the networks did not have enough positive examples to be able to learn to detect them.



6 Results



(a) Sentiment per Politician with RNN₁

(b) Sentiment per Politician with RNN₂

7 Conclusions and Future Work

The achieved results are quite promising in the field of Sentiment Analysis for the German language. What must be taken into account in the developed system is the importance of data sets used for the system's training. It transpires that bigger data sets can really improve the system's accuracy (20% better accuracy in the case of the trained RNNs). Even with a simple RNN model, better data can really improve the performance. In addition, the Neural Network construction can be improved, by adding dropout layers and testing different activation function and other parameters. What is also to be kept in mind is that the training data sets should reflect the data to be analysed, at least in the sense of containing most of the words in the vocabulary of the target data. This is important in order to achieve a good word2vec mapping, without losing information, that could turn into a loss of accuracy. At the current time, is quite hard to find freely available data sets for the German language so that also other approaches should be considered, such as character-level analysis. I strongly believe that it is important to investigate different approaches to sentiment analysis, especially in the direction of multilingual analysis in as our world is more global. Also it is important to give more details about the used data and results by comparing the results to the current State of the Art for Sentiment Analysis as described in [7].

Bibliography

- [1] ABADI, Martín ; BARHAM, Paul ; CHEN, Jianmin ; CHEN, Zhifeng ; DAVIS, Andy ; DEAN, Jeffrey ; DEVIN, Matthieu ; GHEMAWAT, Sanjay ; IRVING, Geoffrey ; ISARD, Michael ; KUDLUR, Manjunath ; LEVENBERG, Josh ; MONGA, Rajat ; MOORE, Sherry ; MURRAY, Derek G. ; STEINER, Benoit ; TUCKER, Paul ; VASUDEVAN, Vijay ; WARDEN, Pete ; WICKE, Martin ; YU, Yuan ; ZHENG, Xiaoqiang: TensorFlow: A System for Large-scale Machine Learning. In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. Berkeley, CA, USA : USENIX Association, 2016 (OSDI'16), S. 265–283. – URL <http://dl.acm.org/citation.cfm?id=3026877.3026899>. – ISBN 978-1-931971-33-1
- [2] ABADI, Martín ; ISARD, Michael ; MURRAY, Derek G.: A Computational Model for TensorFlow: An Introduction. In: *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. New York, NY, USA : ACM, 2017 (MAPL 2017), S. 1–7. – URL <http://doi.acm.org/10.1145/3088525.3088527>. – ISBN 978-1-4503-5071-6
- [3] BAKTHA, K. ; TRIPATHY, B. K.: Investigation of recurrent neural networks in the field of sentiment analysis. In: *2017 International Conference on Communication and Signal Processing (ICCSP)*, April 2017, S. 2047–2050
- [4] BRITZ, Denny: *Understanding Convolutional Neural Networks for NLP*. 2015. – URL <http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>
- [5] CHO, Kyunghyun ; MERRIËNBOER, Bart van ; GÜLÇEHRE, ÇaÄlar ; BAHDANAU, Dzmitry ; BOUGARES, Fethi ; SCHWENK, Holger ; BENGIO, Yoshua: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar : Association for Computational Linguistics, Oktober 2014, S. 1724–1734. – URL <http://www.aclweb.org/anthology/D14-1179>

- [6] CHOI, Yoonjung ; KIM, Youngho ; MYAENG, Sung-Hyon: Domain-specific Sentiment Analysis Using Contextual Feature Generation. In: *Proceedings of the 1st International CIKM Workshop on Topic-sentiment Analysis for Mass Opinion*. New York, NY, USA : ACM, 2009 (TSA '09), S. 37–44. – URL <http://doi.acm.org/10.1145/1651461.1651469>. – ISBN 978-1-60558-805-6
- [7] DASHTIPOUR, Kia ; PORIA, Soujanya ; HUSSAIN, Amir ; CAMBRIA, Erik ; HAWALAH, Ahmad Y. A. ; GELBUKH, Alexander ; ZHOU, Qiang: Multilingual Sentiment Analysis: State of the Art and Independent Comparison of Techniques. In: *Cognitive Computation* 8 (2016), Aug, Nr. 4, S. 757–771. – URL <https://doi.org/10.1007/s12559-016-9415-7>. – ISSN 1866-9964
- [8] GRON, Aurlien: *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O'Reilly Media, Inc., 2017. – ISBN 1491962291, 9781491962299
- [9] HOCHREITER, Sepp ; SCHMIDHUBER, Jürgen: Long Short-Term Memory. In: *Neural Comput.* 9 (1997), November, Nr. 8, S. 1735–1780. – URL <http://dx.doi.org/10.1162/neco.1997.9.8.1735>. – ISSN 0899-7667
- [10] HOGENBOOM, Alexander ; BAL, Daniella ; FRASINCAR, Flavius ; BAL, Malissa ; JONG, Franciska de ; KAYMAK, Uzay: Exploiting Emoticons in Sentiment Analysis. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. New York, NY, USA : ACM, 2013 (SAC '13), S. 703–710. – URL <http://doi.acm.org/10.1145/2480362.2480498>. – ISBN 978-1-4503-1656-9
- [11] KINGMA, Diederik P. ; BA, Jimmy: Adam: A Method for Stochastic Optimization. In: *CoRR* abs/1412.6980 (2014). – URL <http://arxiv.org/abs/1412.6980>
- [12] MIKOLOV, Tomas ; KARAFIÁT, Martin ; BURGET, Lukáš ; CERNOCKÝ, Jan ; KHUNDANPUR, Sanjeev: Recurrent neural network based language model. In: KOBAYASHI, Takao (Hrsg.) ; HIROSE, Keikichi (Hrsg.) ; NAKAMURA, Satoshi (Hrsg.): *INTERSPEECH*, ISCA, 2010, S. 1045–1048. – URL <http://dblp.uni-trier.de/db/conf/interspeech/interspeech2010.html#MikolovKBCK10>
- [13] MINSKY, Marvin ; PAPERT, Seymour: *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA : MIT Press, 1969

- [14] O'HIGGINS, N.: *MongoDB and Python: Patterns and Processes for the Popular Document-oriented Database*. O'Reilly Media, 2011. – URL <https://books.google.de/books?id=ZW1Tu8oU3mcC>. – ISBN 9781449310370
- [15] OLAH, Christopher: *Understanding LSTM Networks*. 2015. – URL <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [16] PEDERSEN, Magnus Erik H.: *TensorFlow Tutorial 20. Natural Language Processing*. 2018. – URL https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/20_Natural_Language_Processing.ipynb
- [17] RANE, A. ; KUMAR, A.: Sentiment Classification System of Twitter Data for US Airline Service Analysis. In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* Bd. 01, July 2018, S. 769–773. – ISSN 0730-3157
- [18] ROSENBLATT, F.: The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. In: *Psychological Review* (1958), S. 65–386
- [19] RUDER, Sebastian: *An overview of gradient descent optimization algorithms*. 2016. – URL <http://arxiv.org/abs/1609.04747>. – cite arxiv:1609.04747Comment: Added derivations of AdaMax and Nadam
- [20] SANKAR, H. ; SUBRAMANIASWAMY, V.: Investigating sentiment analysis using machine learning approach. In: *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, Dec 2017, S. 87–92
- [21] SCHABUS, Dietmar ; SKOWRON, Marcin ; TRAPP, Martin: One Million Posts: A Data Set of German Online Discussions. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Tokyo, Japan, August 2017, S. 1241–1244
- [22] SCHOUTEN, K. ; FRASINCAR, F.: Survey on Aspect-Level Sentiment Analysis. In: *IEEE Transactions on Knowledge and Data Engineering* 28 (2016), March, Nr. 3, S. 813–830. – ISSN 1041-4347
- [23] SHARMA, Anuj ; DEY, Shubhamoy: An Artificial Neural Network Based Approach for Sentiment Analysis of Opinionated Text. In: *Proceedings of the 2012 ACM Research in Applied Computation Symposium*. New York, NY, USA : ACM, 2012 (RACS '12), S. 37–42. – URL <http://doi.acm.org/10.1145/2401603.2401611>. – ISBN 978-1-4503-1492-3

- [24] SHAYAA, S. ; JAAFAR, N. I. ; BAHRI, S. ; SULAIMAN, A. ; WAI, P. S. ; CHUNG, Y. W. ; PIPRANI, A. Z. ; AL-GARADI, M. A.: Sentiment Analysis of Big Data: Methods, Applications, and Open Challenges. In: *IEEE Access* (2018), S. 1–1
- [25] SIDARENKA, Uladzimir: PotTS at SemEval-2016 Task 4: Sentiment Analysis of Twitter Using Character-level Convolutional Neural Networks. In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, California : Association for Computational Linguistics, June 2016, S. 235–242
- [26] STONE, Philip J. ; HUNT, Earl B.: A Computer Approach to Content Analysis: Studies Using the General Inquirer System. In: *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*. New York, NY, USA : ACM, 1963 (AFIPS '63 (Spring)), S. 241–256. – URL <http://doi.acm.org/10.1145/1461551.1461583>
- [27] TAI, Yen-Jen ; KAO, Hung-Yu: Automatic Domain-Specific Sentiment Lexicon Generation with Label Propagation. In: *Proceedings of International Conference on Information Integration and Web-based Applications & Services*. New York, NY, USA : ACM, 2013 (IIWAS '13), S. 53:53–53:62. – URL <http://doi.acm.org/10.1145/2539150.2539190>. – ISBN 978-1-4503-2113-6
- [28] TANG, Tiffany Y. ; WINOTO, Pinata ; GUAN, Aonan ; CHEN, Guanxing: "The Foreign Language Effect" and Movie Recommendation: A Comparative Study of Sentiment Analysis of Movie Reviews in Chinese and English. In: *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*. New York, NY, USA : ACM, 2018 (ICMLC 2018), S. 79–84. – URL <http://doi.acm.org/10.1145/3195106.3195130>. – ISBN 978-1-4503-6353-2
- [29] WES, McKinney: *Python for Data Analysis*. 1. O'Reilly Media, Inc., 2012
- [30] YOU, Quanzeng: Sentiment and Emotion Analysis for Social Multimedia: Methodologies and Applications. In: *Proceedings of the 2016 ACM on Multimedia Conference*. New York, NY, USA : ACM, 2016 (MM '16), S. 1445–1449. – URL <http://doi.acm.org/10.1145/2964284.2971475>. – ISBN 978-1-4503-3603-1
- [31] ZHANG, Shiwei ; ZHANG, Xiuzhen ; CHAN, Jeffrey: A Word-Character Convolutional Neural Network for Language-Agnostic Twitter Sentiment Analysis. In: *Proceedings of the 22Nd Australasian Document Computing Symposium*. New York, NY, USA : ACM, 2017 (ADCS 2017), S. 12:1–12:4. – URL <http://doi.acm.org/10.1145/3166072.3166082>. – ISBN 978-1-4503-6391-4

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 31. July 2018

Martina Donadi