Project Machine Learning Nanodegree: **Social Media Comments Sentiment Analysis**

Kevin Klein

December 18, 2018

Contents

1 Definition

1.1 Project Overview

In recent years social media platforms have increasingly been abused to spread hatred, bigotry and racism as well as personal insults and threats against other social media users of those plattforms.

These comments have the potential to disrupt important discussions in a society and change the public opinion about important political decision making processes. In the worst case these toxic comments can even alter elections, because the public opinion could be changed by the content that most people see on social media. So if there is a lot of hate speech spread on social media, it might be a huge problem in the real world.

Facebook& Co. are working on solutions to get rid of this issues but are overwhelmed with the magnitude of the problem and have engaged masses of people to identify and delete those kind of comments. Unfortunately the amount of data that those people have to process is too big and in the most cases the employees might be too slow to delete harmful comments on social media.

NLP can be a huge source of efficiency gain and could mitigate the amount of hatred and bigotry happening in social media.

In my project I will investigate a NLP problem with a deep learning approach. I will concentrate on the classification of text data, namely I will be looking at social media / Twitter comments and try to classify whether this comments fall into different classes, i.e. good/bad, hate comments/not hate comments.

The purpose of this project is to identify and classify comments posted on social media or forums in the internet. This means we are dealing with a supervised learning problem, where we have a lot of data, that has been classified before in classes like comment is an insult / not in insult or potentially more classes (depending on the problem). The solution will be a neural network, where each comment of the test set will be classified to one of the corresponding classes. Since we will be working with neural networks, all the theory of optimization and loss operators will hold true and can be measured.

For my project I relied mostly on the Course course of Andrew NG "Deep Learning Specialization" in the part about Sequence models.

1.2 Problem Statement

In this exercise I will build a model to classify social media comments. The purpose is to train a NLP-classifier to detect whether given comments are hateful, racist or toxic insults.

In the end I want to have a classifier, which will take a social media comment as the input to a deep neural network and return the class, which the classifier believes is the right category of this comment.

In order to achieve this, I will use a six step approach, which will be explained more thoroughly in the next chapters. The following is an outline of the approach.

1. Load pacoages and data
2. Data preprocessing: Since we are working with text data there is a lot of preprocessing that needs to be done, because text data is mostly not standardised. This procedure will be given

in more detail in the section talking about data preprocessing.

3. Word embeddings & sentence indeces: Word embeddings are a concept used in natural language processing to make sense of text data and use it in a meaningful way. This will be thoroughly explained in the methodology chapter.
4. Model: Finally, after having prepared the data, I will investigate how to put the classifier into place and will discuss different model approaches such as recurrent neural networks (RNN). Of course, this will be discussed in more detail in the methodology chapter.

1.3 Metrics

Depending on the number of classes the loss function will either be "binary crossentropy" for $C = 2$, or "categorical crossentropy" for $C > 2$. The loss function is then given for the true target variable y and the predicted target variable $\hat{y}$ as $L(y, \hat{y}) = -\sum_{i=0}^{m} \hat{y}_i \log(y_i) + (1 - \hat{y}_i) \log(1 - y_i)$. This will also be the loss function we will try to minimize when we optimize the deep neural network.

For the evaluation metrics accuracy is the obvious candidate. However, since the data we are using is highly imbalanced it is not to unlikely that we will get accuracy of 80% or 90%, also if the model is not working good. Therefore when we compare the model with the benchmark model we will use the $F1$-score, since it gives a more comprehensive picture of the overall capability of the model.

2 Analysis

2.1 Data exploration

In the project we will use two datasets from Kaggle

1. Detecting Insults in Social Commentary

   (link: https://www.kaggle.com/c/detecting-insults-in-social-commentary/data)
2. Toxic Comment Classification Challenge (Wikipedia comments)

   (link:https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data)

The first datasets consists of 3947 comments and has a binary target variable called insult. That means, that this target variable decides whether a given comment is an insult. Therefore for the first dataset, the task will be to find an deep neural network that predicts the binary target variable. As mentioned before the data is highly imbalanced, i.e. the outcome of the target variable is not equally distributed. Roughly 73% of the comments are no insults. Intuitively this makes sense as we expect that an algorithm, when used in real world application, will mostly enocounter normal comments and try to classify the few comments that are insulting. But this also means that we need to be careful, when we are using accuracy as evaluation metrics. This will be also be adressed in the methodology section.

Thing get a lot more complicated in the second dataset. First of all there are considerably more data instances, namely 159571 examples. In contrast to the first dataset this one consists of 6 target

	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
1	"Hey man, I'm really not trying to edit war. It's just that this guy is constantly removing relevant information and talking to me through edits instead of my talk page. He seems to care more about the formatting than the actual info.".	0	0	0	0	0	0
2	'COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK'	1	1	1	0	1	0
3	Bye! \n\nDon't look, come or think of comming back! Tosser."	1	0	0	0	0	0
4	"You, sir, are my hero. Any chance you remember what page that's on?"	0	0	0	0	0	0

variables, where each target variable is binary. Therefore there are $2^6 = 64$ different combination of the target variables. In the following we will show a couple of examples.
However, the share of non sensitive comments is roughly 90% and so (as in the first dataset), we have to keep an eye on the evaluation metrics. The six categories we will be working with are given in the following list.

- toxic
- severe_toxic
- obscene
- threat
- insult
- identity_hate

For this dataset the goal is also to predict whether a comment has different characteristics. This can be done in different ways. I decided to build a new variable based on the 6 target variables and which will be used to classify the comments on.

We will build a model that is going to predict all six classes together but independently. This means, given a comment the output variable will consists of a 6×1 vector, which predicts for each entry whether this comment is part of this particular category.

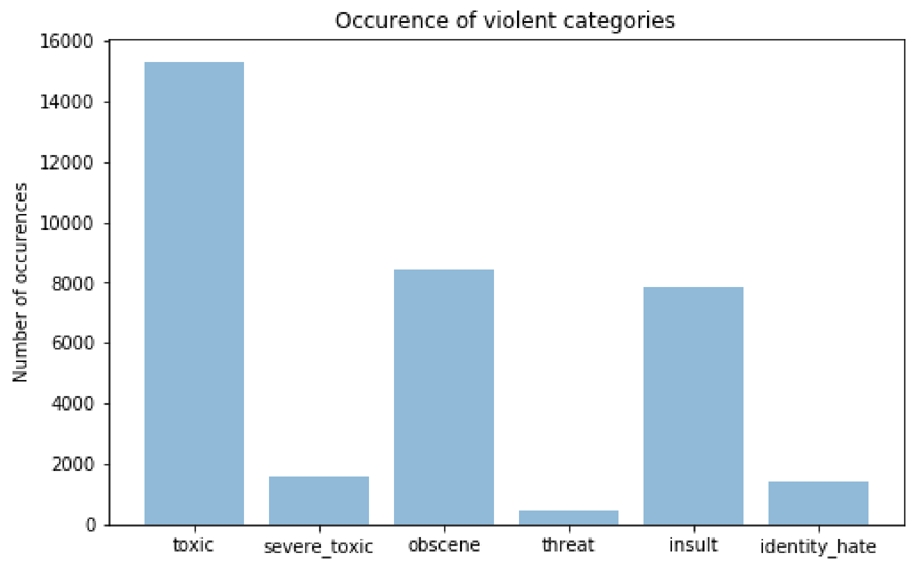


Figure 1: Frequency of the 6 categories

In the figure above we can observe the frequency of occurence of the 6 categories in the second dataset. First of all we obtain a better understanding of the sort of violations we are dealing with. Furthermore, it shows that it will be much harder to model and identify categories threat or identity hate than categories toxic or insult. Therefore it could be that the error in the algorithms comes from misclassification threat rather than the categories that are hit more often. The conclusion that one could set is that it could make more sense to penalize misclassification higher in the loss functions, that do not occur so often.

2.2 Algorithms and Techniques

2.2.1 Word embeddings

Since we are working with text data, we will be using one of the most important concepts in NLP the word embeddings.

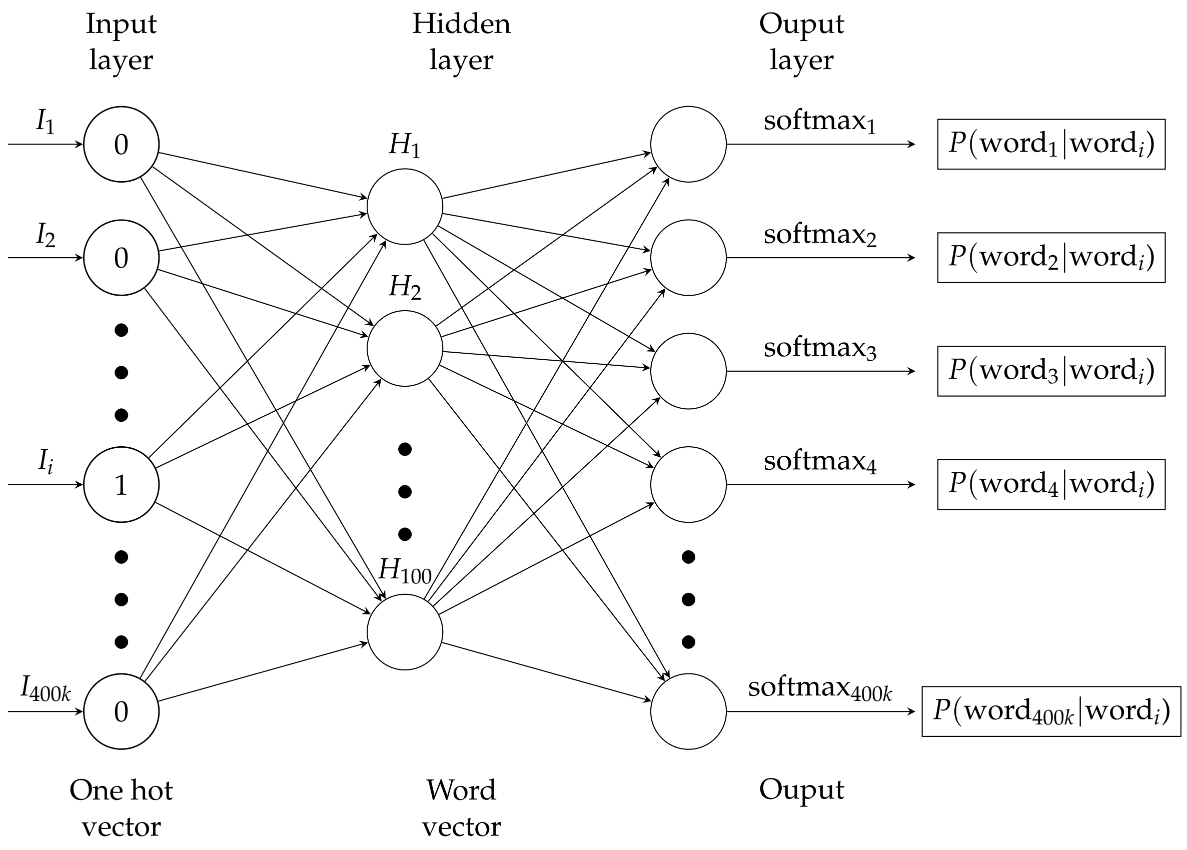
The idea is that in order to make use of words in NN applications we need to transform the words into vectors in order to input them into our neural network. The straightforward idea would be to take each word of a predefined vocabulary with V words and transform each word to a so called one hot vector. These are sparse vectors with dimension V. If we now take the i-th word in the vocabulary the one hot vector of this word would be a 1 on the i-th position of the vector and 0 everywhere else. This can then be used as input for a NN, where the input layer has V nodes and only the i-th node inputs a 1 and every other node inputs a 0. With that we could go on with the neural network.

Unfortunately this method has some drawbacks. First of all the first hidden layer of such a NN would have a tremendous amount of weights ($V * n$, where n is the number of hidden units). In modern vocabularies as used by tech companies $V \geq 100k$ and the hidden unit normally consists of more than 100 units. So we would have more than 1 million weights to update, just in the first layer, which is computationally quite expensive.

Secondly, certain words normally appear next to other words. For instance when I have the word orange I would expect that one probable next word in the sentence would be juice. So what we want to find is a vector representation of words, which would capture certain dependecies between words, because that would be easier for the algorithm to interpret in terms of meaning than just a bunch of words ordered next to each other. The problem with the one hot encoding is that there are no dependencies between words, because the inner product of two different words is always 0 and so under this representation all words are "independent". That is why we use word embeddings. And in this exercise we will use the Glove Vector Embedding with 100 dimensions developed by Stanford.

The idea is that we take a corpus of text (sentences) of some sort, i.e. from all Wikipedia articles and try to find the most probable so called "context" word, i.e. the word which will likely be going to come after a certain word.

So based on the word corpus the word vectors are being trained. The procedure can be observed in the figure below. Basically, the word embeddings is trained by a NN with one hidden layer. The inputs are the one hot vectors (for the Glove word embeddings 400k words were used, so the vector is of dimension (400k,1)), and are fed to the hidden layer with 100 units and those units are fed to the output layer, which predicts the most probable word given the input word.



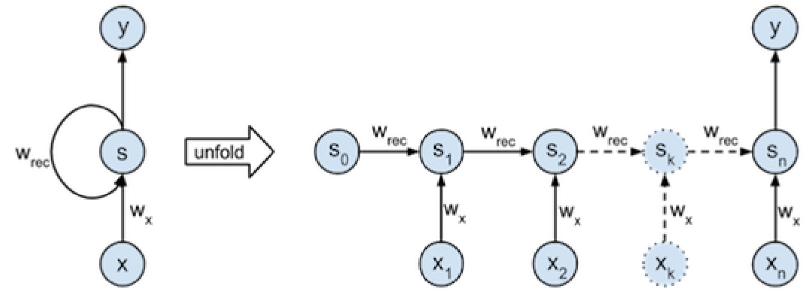


Figure 2: RNN framework

In this training model we actually do not care about evaluation, we only care that the training returns meaningful weights between the input layer and the hidden layer, because that is going to be the word embedding. Since in this example only the weights going from the i-th vector entry are trained, these weights (when the training is finished) are going to represent the word vector of the i-th word in the vocabulary. Of course the same holds true for every word in the vocabulary.

In the proposal I said that I will also build my own word embedding, but since those embeddings are already given this does not add any value to the project itself and therefore I decided not to go into this extra feature, and there are plenty of other characteristics to take care of.

2.2.2 RNN architecture

For our model we are going to use a recurrent neural network architecture. Since we are dealing with text data we need a sequence model to process the data. In other deep learning application the task is to classify one instance of a dataset, i.e. in a CNN each training example input is one picture with the same number of pixels. However, for text data this is different because know we are dealing with sentences, where each word consists of one input variable to the NN and the sentences themselves can have different length. So the only meaningful approach is a recurrent neural network.

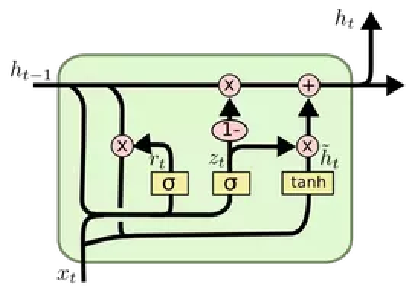
In the following graph an example of a RNN is given. As we can see the input variables are of the form $\{x_t\}_{t=1...T}$, where x_t are the word vectors of the corresponding words and T is the maximum length of the sentences. The hidden layer inbetween is denoted by s_t and the task of it is to propagate the saved information forward through time and use them to predict the next hidden layer and potentially the y_t. Our RNN model is a so called many-to-one model. This means that since we are only predicting one outcome variable at the end of the RNN architecture. In other NLP applications that might differ a lot. For instance in machine translation for each x_t there is a y_t, where in this case the y_t consist of the translated word in another language.

As we can see in the figure the model start with the first "hidden layer" at time $t = 0$ as s_0 and is then fed forward with the corresponding x_t. It is important to mention that the weights w_{rec}, w_x and w_y (the weights from s_n to y) are not changing through time. So the model just needs to train each weight vector once.

For longer sentences eventually it becomes tricky to predict the correct class. The reason is that it becomes harder to contain the meaning of the sentence and remember the characteristics of a sentence. For instance, if a sentence contains the word **good** a lot of times but refers to a sentence that describes that **nothing was good** then we expect the model to identify that structure.

For that we are going to use an updated version of a RNN model called GRU model. The model framework remains the same just the blocks that described the s_t are going to change in order to be able to save necessary information through time.

As we can see in the figure below is that more weight matrices are being used (which are also staying the same for all time steps). Those matrices are then usd to save dependencies through time. In the last equation $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$, the h_t is the hidden layer output for the next time step and z_t is a weighting factor calculated at every time step. This one determines which information should be taken from the last time steps hidden layer and which part should be updated with new information. The factor $\tilde{h}_t$ is calculated at every time step and basically decides if some information that was given about dependencies is still vaild or should be overwritten.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3: Description of Graded recurrent unit

2.3 Benchmark Model

The nearest Benchmark model is a random generator, which mimics the distribution of the target variables and draws random guesses from this distributions. For the datasets I use this implies the following

- For both datasets we will generate a benchmark model, which will sample numbers from a distribution, which takes as input the relative frequency of the classes in the target set. This will then be compared with evaluation metrics, namely with accuracy and F1-score.

Clearly this means that my task for the project will be to beat the random generator, since otherwise my algorithm would be worse than random guessing, which would not qualify as a machine learning algorithm whatsoever.

3 Methodology

3.1 Data Preprocessing

Since we are working with text data, the first and arguably most important step in this project is to preprocess the data to be used in a meaningful way. The main reason for that is because we are using comments from many different people, that all have a particular way to communicate and write so it is necessary to "standardize" the data. Also, since our data consists of insults there is a lot of nonsense being written, so another challenge is to cap how much "sense" a comment needs to contain in order to be useful for this project. I used a x - step procedure to clean and prepare the data, which is going to be explained in the following.

1. Decontract: The english languages phrases which are abbreviated, i.e. **don't** or **I'd**. For consistency the first step is therefore to decontract those phrases and change them to **do not** and **I would** respectively.
2. Preprocess: In this step we are splitting the data at certain special characters that appear often in the dataset, get rid of certain punctuations, which are useless for the analysis and lower case all the data, because for our specific task it does not matter whether we are working with lower or upper case text.
3. Read word embeddings: In the third step the word embeddings are downloaded and indexed, such that we have a comprehensive reprensation of the dictionary used for the glove word embeddind.

4. Assess meaning of words: Now that we have preprocessed our data the next step is to determine whether all the words in the comments are in the dictionary. So I built a function which inputs and a sentence and returns a vector with indices, which state on which positions the words are in the dictionary. At the same time the words which are not in the dictionary take index $= -1$. For each comment the words that are not in the dictionary are counted and if there are more unkown words in a comment than a certain threshold, this comments are not taken into account for training.

5. Transform sentences to indices: The next step is to find the index representation of the sentences. For the model all the sentences must have the same length, otherwise we cannot enter the comments as input. So the first step is to find the maximum length of words in the comments and the second step is to append the vectors with 0 of all comments that are less than maximum length.

6. Determine optimal maximum length: The maxim lengths of the dataset are

   - dataset 1: Maximum length is 733 words
   - dataset 2: Maximum length is 1403 words

   This lengths are quite long and pretty hard to work with. So I decided to shorten the maximum length such that a reasonable percentage of the data is still used (I used 90%). The new maximum lengths then are

   - dataset 1: Maximum length is 70 words
   - dataset 2: Maximum length is 120 words

3.2 Implementation

In the following I will take a complete other approach as I misunderstood the assigment for the second dataset. The goal was not to predict just one class but all six classes together.

For dataset 1 the implementation was straightforward. I build a RNN with LSTM blocks. I used 1 hidden layer with 64 units each and 30% for the layer from embedding to hidden layer and 60% dropout for hidden to output layer for regularization reasons. As in this dataset we were working with a binary category dataset, as last actviation I used a 1 unit layer with a sigmoid activation. For the learning process the adam optimizer was used with slowly decaying learning rate and we saved the weights of the epoch which had the best validation accuracy to account for early stopping regularization.

So in contrast to the previous submission of the project we are going to model the categories directly and are not splitting the problem into 2 subproblems.

The architecture used is a RNN with GRU blocks. The reason I used GRU over LSTM is because they are performing better and are computationally cheaper to use. We used 1 hidden layer with 64 units and the dropout was set to 60%. For the output layer we are using 6 nodes for each of the 6 categories and since we want to predict each category **independently** the activations are 6 sigmoid functions. Therefore for the loss function is binary crossentropy and optimizer is Adam with learning rate of 0.005. Furthermore, we used early stopping to save the weights of the best validation score to add more regularization.

3.3 Refinement

In contrast to the chosen model we used higher dropout to decrease the effects of overfitting.

Furthermore, the GRU is bidirectional (with 64 hidden units; so in total 128 units because bidirectional RNNs have units in both directions), meaning that the GRU unit is running in both direction and therefore more information is contained and processed.

Thirdly, after the GRU blocks we pass on a 128×120 matrix, where the information of the hidden unit is passed on for every timestep (in this case timestep refers to word until maximum length). Then we apply a so called Global Maximum pooling layer. The idea comes from CNN in image classifaction. The difference here is the task is to find the maximum value for each unit over time. The reason is that through this the model becomes position invariant, this means that it does not matter in which position a (for example)curse word is appearing and therefore the model becomes more stable and is more looking for certain words or features in general and not on the position where it is occuring [1]. This returns a vector of 128 units and is then fed to the final 6 unit output layer.

4 Results

4.1 Model evaluation & validation

For the first model I used 100 epochs to train the model and could observe that it converges and reached an training accuracy of roughly 94% after those 100 epochs. The validation accuracy peaked at 83.96% and stayed there with the early stopping approach. Afterwards the model failed to generalize up to the 86.86%. This means that there is still some source of overfitting which could be prevented by some of the techniques described above. However, as the training only used 2500 training examples the overfitting could also be explained by the inability of the model to generalize given the low number of training examples.

When applied on the test data we achieved an accuracy of 85.55 %, which means that more than 4 out of 5 comments are correctly classified. For the sake of this project I consider this satisfactory, however if it were to go into production more steps would need to be taken in order to improve overall accuracy.

In terms of sensitivity the model converged for every set of possible train-test split, as we have not fixed a random state for the split. This means that everytime I ran the code the model would always start to converging, proving the stability of the model.
Also the results can be trusted as I started to invent my own invented comments to see whether the model can generalize well on data, which is not drawn from the same distribution (as I created it and it does not come from the original dataset) and could mostly classify the comments correctly. So the algorithm can generalize not only on the test set but also on data it has never seen before.

For the second dataset I am going to describe the improved model. Since there were almost 100000 training examples the training was quite satisfactory after 10 epochs and attained a validation accuracy of 98.14%. At this point there is really not much that can be done to improve the model. Of course there is still some extent of overfitting and other smaller issues that could be adressed, but this would not change much of the outcome of the model itself.

[1]Source: https://stackoverflow.com/questions/48549670/pooling-vs-pooling-over-time

When applied on the test set we also obtain 98.14% and the same characteristics as for the first dataset also hold for the second one. This means that the model is invariant to input changes and works on every possible split of train-test data split. Also for my own comments I produced to test the model independently of the distribution given by the dataset the model classifies all comments correctly. So in my opinion, also if there is always room for improvement, the model is stable and appropiate in its ability to solve the assigment.

4.2 Justification

For both dataset the model easily performed much better for both accuracy and the other evaluation metrics that was considered F1-score. Obviously, this is mainly to the fact that the benchmark models are not completely meaningful as we used randomness as benchmark model. However, especially for the second dataset where 89 % are non-violent comments any other straightforward benchmark model (such as predicting everything as non violent) would not put the benchmark model in a better position than the calibrated model. Also, the model for the second dataset is so robust in terms of accuracy, that not many benchmark model would even come close to this performance.

5 Conclusion

5.1 Free-Form Visualization

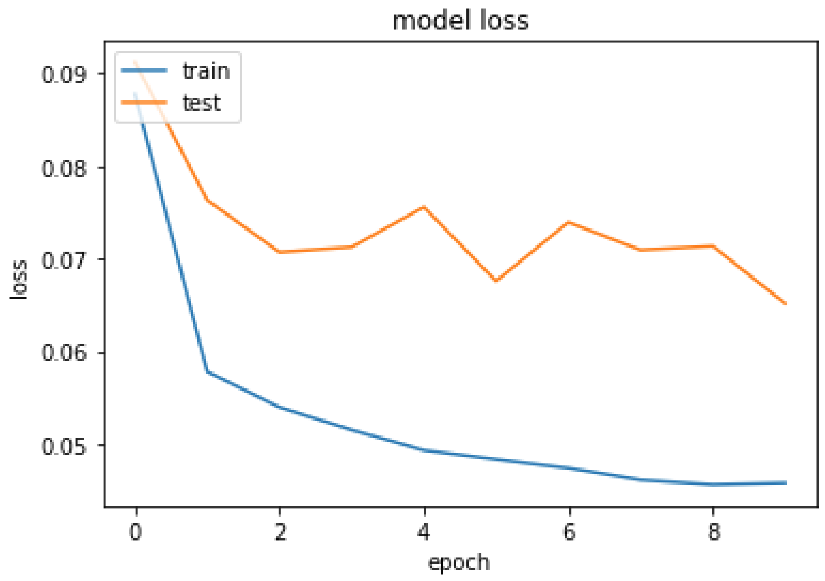


Figure 4: Training and validation loss of model

This figure represents the final calibration of the model of the second dataset and shows the converging of the training and validation set. The reason that this is not that smoothly visible for the test set is because in the first epoch most of the initial training was done, i.e. most of the loss decrease was done in the first epoch and the plot starts showing the training loss **after** the first epoch. But still we can observe that this converges. Just in the end the decrease of the training and validation loss smoothens out, which means either that the model reached its ability to model the training data and/or that we reached the point at which the model manages to generalizes best.

5.2 Reflection & Improvement

I considered this project quite interesting as it might have some real world applications which could be used probably quite straightforward to fight the problem with even rougher becoming conditions in social media platforms. I felt this project was difficult for two main reasons.

- Wrong architecture and understanding: As for most deepl learning problems the problem of the optimal architecture is one of the hardest and most time consuming. For me it was also a problem of understanding, since I misunderstood the assigment for the second dataset. The goal was not to predict just one class but all six classes together, i.e. the prediction should give a prediction for each of the six categories.

  So my problem was that I had closed my eyes for another approach and because I was focused/convinced that the approach I took was the right one, therefore I used a lot of time investigating the wrong approach instead of looking for other possible architectures.

Through this I learned that when stuck in an assigment it can be a good idea to take a step back and investigate whether the understanding is right or if there is a better way to deal with a certain problem

- Dealing with overfitting: When it comes to overfitting there are countless ways to tackle this particular problem. Whether you can play around with hyperparameters like learning rate, number of hidden layers, number of hidden units per layer or dropout rate, there are potentially countless different combinations of hyperparameters to be tested.

  What I learned is that when we have multiple options the easiest one is always the most favorable as this option is most suitable to generalize on the test and or validaton set.

For dataset 1 there is still some space for improvement. The important task here would be to find a tradeoff between training accuracy and overfitting since the dataset is probably to small to achieve both of them. This means if we are looking too much for generalization we might run the risk of having to high bias in the beginning.

For the second dataset we could argue that there is always room for improvements. As mentioned just above to improve accuracy it is always possible to play around with parameters and remember the combination which yielded the best outcome. However, at some point it is necessary to think whether it makes sense to add so much complexity in order to obtain possibly tiny results.

Another possible improvement for dataset 2 is to look at the categories as being dependent, meaning to consider also the possiblity of categories having correlation and taking this into account when building the model.

6 Pictures

- Figure 1: https://peterroelants.github.io/posts/rnn-implementation-part01/
- Figure 2: https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15