# Deep Learning: Project Report
## Sentiment Analysis

Thora Daneyko
Matrikelnr.: 3822667

September 2, 2017

# 1 Program description

## 1.1 Preprocessing

Before we can feed it to the model, the raw data needs to be preprocessed and converted into numeric matrix representations. Overall, the model needs information about the correct labels, the index of each word in the embedding lookup table, additional features for each word (e.g. capitalization features), the characters in each word, and the real length of each word and Tweet.

First of all, each of the Tweets is tokenized using NLTK's `TweetTokenizer` class. All words except hash tags are converted to lower case, while an all-caps feature stores whether the word originally was in all upper case (suggesting intense emotion). This feature is concatenated with the word's feature vector in the EmoLex or NRC Emotion Lexicon (Mohammad 2010) to give a discrete feature vector for each word. Then, the characters of each word are converted to a numeric vector to later get a character-level representation of each word. Afterwards, hash tags, user names, numbers and URLs are collapsed into `<hashtag>`, `<user>`, `<number>` and `<url>` tokens, respectively.

## 1.2 Model

Figure 1 shows an overview over the components of my model. I implemented almost everything envisioned in the project preparation report, with the exception of the sentiment-specific word embeddings.

The vector representations for the individual words in a Tweet consist of three parts: 1) A pre-trained word embedding, 2) a character-level representation, and 3) a discrete feature vector containing the EmoLex and all-caps features.

The word embeddings are taken from the GloVe project website (Pennington, Socher, and Manning 2015). They have been trained on 2 billion Tweets, resulting in a dictionary size of 1.2 million words. In my program, however, I only use the 100,000 most frequent words, which accounts for 94 % of the tokens in the training and testing data. The GloVe Twitter embeddings are released in

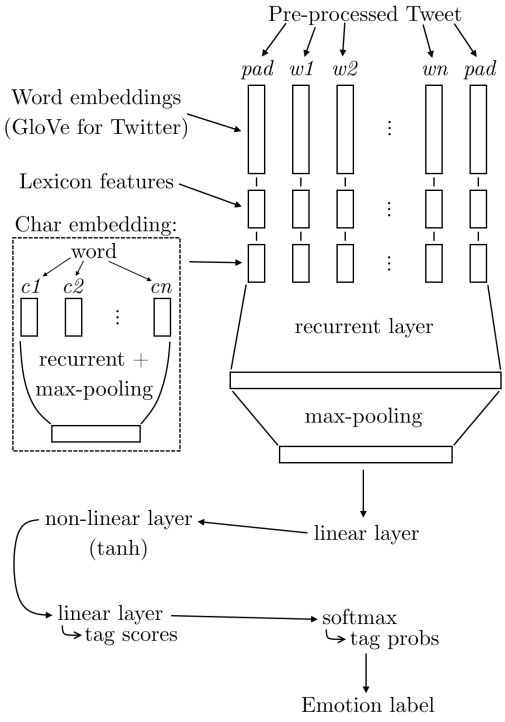


Figure 1: The final architecture of my model.

different sizes, but the largest embeddings with 200 dimensions have proven to perform best. The vector of unknown tokens is initialized to the mean vector of the rarest 10 % of words in the dictionary.

The character-level representation is obtained by first creating a lookup table for the characters in the data using `tf.nn.embedding_lookup`. All words are then fed to a bidirectional recurrent neural network (biRNN) to get a single vector for each word. The size of this vector is further reduced using max-pooling.

These three word vectors are concatenated into a single one for each word. To get a single representation of the complete Tweet then, these vectors are fed to another biRNN with max-pooling, just as with the characters. The resulting Tweet vector is then passed through a linear layer, a hidden layer with tanh activation, and finally another linear layer which produces the tag scores. These are converted into probabilities using softmax. The label with the highest probability is the one predicted by the model.

## 2 Results

The parameters set in the `config.py` provide the best results from a range of parameters I experimented with. Unfortunately, the model overfits quickly: The validation loss starts increasing again after about 10 epochs. However, the

| | micro | macro |
|---|---|---|
| Precision | 62.81 | 56.55 |
| Recall | 62.81 | 53.05 |
| F1 | 62.81 | 52.91 |

Table 1: Precision, recall and F1 score with micro and macro averaging at epoch 13, mean value of three runs.

F1 score still improves for some more epochs. To reduce overfitting, I have implemented a high dropout of 0.5, which only slightly contains the problem. Exponential decay of the learning rate helps against overfitting, but decreases the overall results, so it is disabled.

Table 1 shows the precision, recall and F1 scores of the final model. The results vary on each run, usually producing macro F1 scores between 52 and 54, but sometimes even reaching more than 55 or dropping below 52. The micro F1 scores are quite stable at 62 to 63.

# 3 Usage

## 3.1 Submitted files

The code is structured similarly to the one used in the assignments (which served as a basis for my program). `train.py` is the executable file controlling the training and validation. The model itself is defined in `model.py`. Model parameters are stored in `config.py`. `numberer.py` is the id-token mapping helper class that was provided for the homework assignments. All preprocessing of the input data, embeddings and emotion lexicon is defined in `preprocessing.py`.

The embeddings have already been preprocessed, since the original file from GloVe containing the complete 1.2 million word vectors used almost 2 GB of space. The id-to-word mapping and lookup matrix needed by `train.py` are stored in binary format (using `pickle`) in `embeddings`, with a size of only 154 MB. `emolex.txt` contains the EmoLex data.

## 3.2 How to run

To run the model, use the following command:

```
python train.py TRAIN_DATA TEST_DATA EMBEDDINGS EMOLEX
```

where `TRAIN_DATA` is the data the model should be trained on and `TEST_DATA` is the data to evaluate on. `EMBEDDINGS` should be the file containing the preprocessed word embeddings and `EMOLEX` the one with the NRC Emotion Lexicon data.

When doing several runs on the same data, it can save some time to only do the preprocessing of the raw data once. To do this, you can add an optional argument `-write`, which will save the preprocessed data in binary files called

`traindata` and `testdata`. To read from these files directly and skip the preprocessing step in subsequent runs, pass them as `TRAIN_DATA` and `TEST_DATA` and add the option `-read`. The option must be the first argument given to the program:

```
python train.py [option] TRAIN_DATA TEST_DATA EMBEDDINGS EMOLEX
```

By default, `train.py` will do all the preprocessing and not create any additional files.

For each epoch, the program will output the current results, similar as in the assignments, e.g.:

```
epoch 0 - train loss:  93.93, validation loss:  80.35; prec:  37.05,
recall:  33.46, f1:  31.23 (micro:  52.88)
```

The precision, recall and F1 values reported are the macro-averaged scores. Finally, the micro-averaged score is also printed, but not split into precision, recall and F1 because they are equal with micro-averaging.

# References


Mohammad, Saif (2010). *NRC Word-Emotion Association Lexicon*. URL: `http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm` (visited on 07/23/2016).

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2015). *GloVe: Global Vectors for Word Representation*. URL: `https://nlp.stanford.edu/projects/glove/` (visited on 08/15/2017).