CSE 5525 Project Report
Deep Learning Experiment on Twitter Emoji Classification

Shijie Qu, Yifan Song, Wenbo Nan
Ohio State University
{qu.219, song.1221, nan.19}@osu.edu
Instructor: Prof. Alan Ritter

Abstract

Emojis are ideograms of smileys, symbols and other daily objects that can be sent through electronic messages and social media platforms. They have gained significant popularity in recent years, and the emoji "Face with tears of joy" appeared even in the Word of the Year announced by Oxford Dictionaries.[1] Since emojis have become an essential part of people' daily messages and are sometimes even interchangeable with texts as a way of expressing oneself, we think it should be worth to take a look at how emojis are used by people. To achieve this goal, our team randomly selected a number of tweets that consists of emojis as well as texts, and we aim to train a model to classify those emojis given the corresponding texts.

In this paper, we strive to conduct an experiment on Twitter emoji classification task using deep learning frameworks. We explored several deep learning architectures that we learned in class such as Convolutional Neural Networks, and Recurrent Neural Networks to build up distinct multi-label classification models, and in the end, compared their performance on our task of predicting emoticons.

1. Introduction

Less than twenty years ago, people still used emoticons as a supplement of electronic messages. And in the early 2010s when the major mobile phone platforms such as Apple integrated emojis into their systems[2], emojis became exponentially popular within the past several years, and by mid-2015, half of comments on Instagram contain emojis[3]. Such an important element in current-day communications should not be neglected when we perform natural language processing. However, there are only a limited amount of emojis, yet people use them in various ways and situations, which could lead to completely different meanings of the emojis used. Therefore, for this project, we would like to apply the deep learning algorithms we learned in this course to train models that could potentially find relations between texts and emojis, and be able to make predictions of emojis to use given pure text in English.

In the paper, we discussed our motivation to work on this project (Section 2), the source and preprocessing of our dataset (Section 3), the models we trained (Section 4), and the analysis of them (Section 5). Finally we talked about the conclusion and our vision for the future(Section 6). The link to our source code is provided in the foot node of this page.[4]

2. Motivation

The task of Natural Language Processing is comprehensive and difficult for computers to learn accurately because human languages can be very flexible and subtle. For example, uncommon acronyms, slang words and irony are all factors that contribute to the difficulty of NLP. Recently, with the widespread use of social media and electronic devices, emojis appear in our daily messages and become an inseparable part of our communications. Emojis can serve as an enhancement of one's sentiment or sometimes suggest irony. Sometimes emojis alone can be sent as messages with meanings.

Since the trend of emojis is relatively new in the recent years, there have been some studies and competitions on such topic. In the SemEval-2018 competition, one task was to complete a multilingual emoji prediction system.[5] More specifically, the training sets in the competition contains only one emoji in each tweet. And according to the report, most teams used SVM classifier or Bi-LSTM. Differentiated from the report, for our project, we trained our model with tweets that have no limit on number of emojis because we want to make our model as close to real life as possible. And we wanted our project to focus on deep learning algorithms. We built several deep learning algorithms

[1] https://languages.oup.com/word-of-the-year/2015/
[2] http://nymag.com/intelligencer/2014/11/emojis-rapid-evolution.html
[3] https://emojipedia.org/stats/
[4] https://github.com/howieha/Text2Emoji
[5] https://pdfs.semanticscholar.org/3e96/74a344db5e4e7aa02222d659e58e4307b084.pdf

and compared the results of each, in order to find a possibly optimal model that suits such task.

3. Data

3.1 Dataset

The raw dataset we adopted was from University of Amsterdam / Amsterdam University of Applied Science[6]. The original source contains 13 million tweets represented as their tweet status IDs and emoji annotations associated with them. The dataset were divided into training, test and validation sets. For the course of this project, we only used the training dataset from the source, which had more than 12 million entries.

Given the original dataset as a two column matrix with the first column being tweet ID and the second column being the list of IDs of emojis appearing in each tweet, we then used Tweepy[7] library to access the Twitter API and extract the texts of each tweet.

3.2 Preprocessing

After extracting texts using tweet ID from **3.1**, we got a total of 7,443,390 pairs of valid texts and emojis. To obtain a cleaner format of the texts, we further preprocessed them by removing any emojis, carriage return/line feed signs, urls and any word starting with '@' (which is used to tag a user) in the texts. Then, we trained a tokenizer on the training texts to tokenize each sentence. Here we adopted the SentencePiece[8] tokenizer, which is an unsupervised tokenizer that implements subword units and unigram language model. In this way, the tokenizer could successfully handle any out-of-vocab tokens in the validation or training set. Here, we manually set the total number of tokens to be 32000, which we believe is large enough for training. In the end, it is shown that on average, each sentence consists of 62 tokens, with the longest having 512 tokens, while the shortest having only 2 tokens. *Figure 1* below shows the distribution of the length of sentences in terms of number of tokens.

For the emoji part, we found that there is a total of 1,300 distinct emojis in our data. However, according to figure 2, we can see that most of the emojis occur only in a few times. Thus, to get rid of the sparsity of the data, we decided to use the most frequent 50 emojis for our multi-label classification tasks.

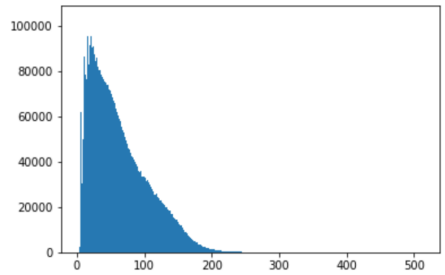

Figure 1. Distribution of the sentence length (length vs. frequency)

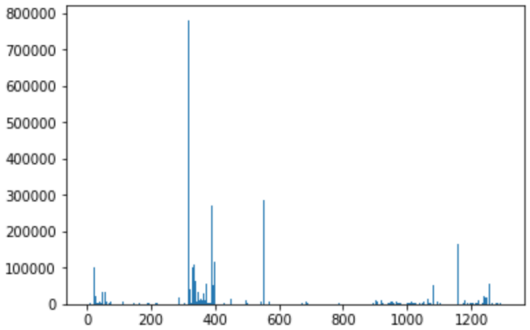

Figure 2. Distribution of the emoji frequency (emoji ID vs. frequency)

4. Method

Before start building the model, we first made our data to have the same length across samples, i.e., suppose the target length is *n*, then every sentence with a length smaller than *n* will be padded with zeros and every sentence with a length greater than *n* will be truncated. As the average number of tokens per sentence is 62, we choose $n = 64$.

4.1 Designated Models

The baseline model we chose is one of the simplest networks, Feed Forward Neural Network (FFNN). Based on what we have learned in class, we would like to explore how CNN and RNN (LSTM), along with their variations, perform in this problem. In addition, we would like to see how adding the attention mechanism will affect the result with both CNN and LSTM. To compare in a relative same level, all models follow a basic initialization: one embedding layer, one kernel layer (linear/convolution/lstm), one activation layer, one fully-connected layer and one dropout layer. The hyper-parameters will be shown in the code/appendix.

[6] https://uvaauas.figshare.com/articles/Twemoji_Dataset/5822100

[7] https://tweepy.readthedocs.io/en/latest/

[8] https://github.com/google/sentencepiece

4.2 Weight Penalty

We began with three models: FFNN, unigram CNN and basic LSTM. The performance is however, poor as precision, recall and f1 scores are all almost 0. This was due to the sparsity of each label. To solve this problem, we decided to put a weight on positive samples for the loss function so that a wrong predicting positive sample will provide a loss of *w* * *loss* while wrong predicting a negative sample is still the original *loss* (*loss* = Binary Cross Entropy Loss). By setting *w* > 1, the model will then tend to catch the positive samples. We start by setting *w* = *# of negative / # of positive* for each class. The average f1 score has significantly increased to 0.15 with an average recall of 0.58 but an average precision of 0.09. We would like to try increasing our precision while keeping a good recall value. Intuitively, we think this could be achieved by reducing *w*. Thus, we multiplied *w* with a coefficient *c* and tried different values of c. As displayed in Figure 3, we decided to use *c* = 0.5 in the following models., i.e. the loss of positive samples would now be *0.5 * w * loss*

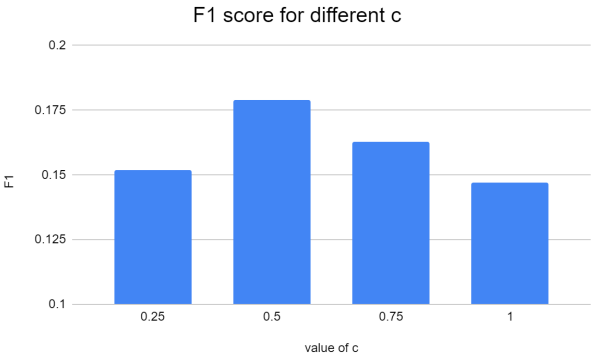

Figure 3. F1 score for coefficient of positive loss weight

4.3 CNN

For CNN models, we consider three variations: CNN model on unigram, bigram and trigram. In our setting, the unigram model contains only one convolution layer with kernel size equal to the embedding dimension. The bigram model consists of both unigram and bigram kernels. Similarly, the trigram model includes unigram, bigram and trigram filters.

4.4 LSTM

For LSTM model, we simply consider two variations: the normal LSTM model and the bi-directional one. For the bi-directional LSTM, we just concatenate the two outputs together and treat it the same way with the one direction LSTM afterwards.

4.5 With Attention

Adding self-attention mechanism to a deep learning model normally enables the model to learn which part of the sentence would be more significant for prediction, and then place more weight on such parts. Thus, we wonder that with this self-attentional layer on top of the CNN or LSTM model we introduced previously, the classification performance might be improved.

The architecture of both models after adding a self-attention layer is displayed in the two images below. For illustration purposes, both of the figures only show how the model will behave when predicting a specific emoji l, but in reality, we could use matrix form multiplication to predict all 50 classes simultaneously.

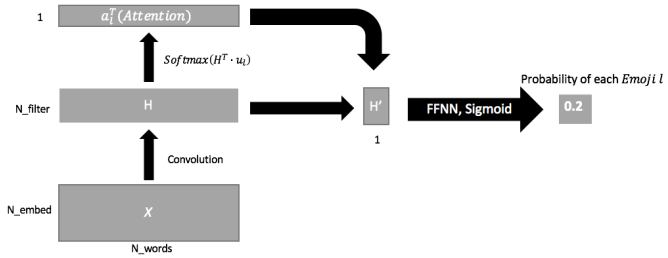

Figure 4. Architecture of Self-Attention CNN

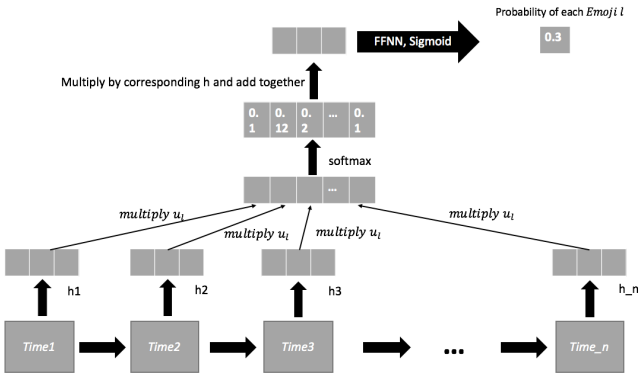

Figure 5. Architecture of Self-Attention LSTM

5. Results and Analysis

5.1 Model Performance

After running experiments for our eight models, we get the result based on five metrics: accuracy, precision, recall, F1 and ROC-AUC score. Each metric is calculated based on the average of all 50 labels with a relatively large variation. We'll show that in the next section. For each metric, accuracy is not a very useful one for our problem as it's meaningless for this extremely sparse dataset. The ROC-AUC score would be a good overall evaluation for the models while F1-score should also be focused as it balances precision and recall.

	Accuracy	Precision	Recall	F1	ROC-AUC
FFNN	0.889	0.018	0.157	0.059	0.577
CNN_uni	0.912	0.144	0.385	0.184	0.718
CNN_bi	0.922	0.147	0.415	0.196	0.728
CNN_tri	**0.923**	**0.146**	**0.418**	**0.198**	**0.731**
LSTM	0.942	0.138	0.319	0.176	0.684
LSTM_bi	0.941	0.139	0.321	0.179	0.688
CNN_att	0.921	0.109	0.375	0.167	0.72
LSTM_att	0.95	0.093	0.362	0.131	0.652

Figure 6. Model Performance

From the result, it's easy to determine that both CNN and LSTM models outperform significantly the basis model: our FFNN model is too simple to work with this huge sparse noisy data. Furthermore, the CNN models perform better than the LSTM models. An intuition behind would be that Tweets are often very short and different from normal language style, as well as lack logics between words and more wrong spelling or meaning. All these contradicts the pros of LSTM for carrying information across a long sentence. It's a little surprising for us that the attention based models perform worse. This could probably be explained with the same intuition above of lack logics and wrong noises. We also guess this would be due to incorrect parameter settings and initialization or the limit of computation power. For CNN models, the result is not surprising that the bigram and trigram models outperform the unigram model.

5.2 Per-label Analysis

Attached in the appendix is the table showing our model's prediction results of the most frequent 50 emojis in our dataset, ordered in descending order according to F1 score. We can see that the most accurate ones are the black heart, camera, smiley face and the famously popular Face with Tears of Joy. We think that there are two possible reasons. The first is that these emojis appear frequently, adding more weights to them whenever the model predicts true positive. The other possible explanation being that these emojis' meanings are straightforward, for example the camera emoji. People usually associates a small set of words with the camera emoji, making the predictions based on these words more accurate. We also observed that it is mostly smileys towards the bottom of the list, we think that it is due to the high coverage of these emojis, i.e. they can be used in various situations and does not really make a significant impact when they don't align with the original meaning of the sentences.

6. Conclusions and Future Work

To sum up, our group successfully conducted a set of experiments, where we used several neural-based models to perform emoji classification given tweets. Throughout the process, we tried different variations of vanilla CNN and LSTM models, and compared their performance against each other based on 5 of our evaluation metrics. At the end of the day, we found that using Convolutional Neural Network with a mix of trigram, bigram, and unigram filters may yield the best overall results.

However, due to the limit of time and computational units, we were not able to identify the reason why the two attention models inhibit the model predicting ability. This might be attributed to a couple of reasons such as, we did not run a large amount of epochs, or the hyperparameters are not tuned to their optimal values, etc.

In the future, if time and resources permitted, our team would like to further explore some more computationally-powerful deep learning frameworks such as BERT to see if we could obtain a significant performance gain. Also, due to the similarity between several clusters of emojis, we could also adopt approaches like hierarchical classification to capture the inner structure of emojis, and thus, to improve the model performance. Last but not least, since some tweets might have come with images, we believe that if we can come up with a way to utilize both text information and image information, we could potentially achieve a major breakthrough for our problem.

Appendix I. Per-label Results

Emoji	F1 Scores	Precision Scores	Recall Scores	Accuracy Scores	ROC-AUC Scores
❤	0.863	0.95	0.795	0.958	0.901
📷	0.52	0.431	0.656	0.971	0.847
☺	0.485	0.347	0.802	0.963	0.893
😂	0.395	0.506	0.324	0.816	0.692
😍	0.334	0.276	0.421	0.89	0.73
😘	0.319	0.238	0.483	0.925	0.763
🤔	0.292	0.227	0.41	0.911	0.732
🙄	0.284	0.203	0.474	0.857	0.725
💕	0.271	0.179	0.564	0.91	0.78
🙏	0.27	0.186	0.493	0.936	0.769
😩	0.252	0.183	0.404	0.889	0.717
😊	0.25	0.206	0.32	0.909	0.702
💯	0.246	0.164	0.493	0.908	0.754
😭	0.244	0.211	0.291	0.887	0.685
👍	0.22	0.146	0.45	0.925	0.748
🔥	0.218	0.144	0.455	0.942	0.758
🟫	0.215	0.147	0.407	0.904	0.723
💔	0.196	0.12	0.547	0.923	0.778
🟫	0.178	0.122	0.329	0.896	0.692
🙌	0.172	0.111	0.387	0.91	0.717
🏻	0.172	0.121	0.299	0.92	0.694
🏽	0.17	0.119	0.299	0.906	0.687
😔	0.159	0.099	0.403	0.933	0.733
🙃	0.157	0.1	0.375	0.897	0.705
💙	0.152	0.089	0.52	0.923	0.768
💀	0.151	0.092	0.424	0.903	0.724
👌	0.146	0.09	0.391	0.9	0.711
😎	0.145	0.091	0.355	0.917	0.708
👀	0.144	0.089	0.383	0.912	0.715
😒	0.143	0.083	0.508	0.872	0.734
✨	0.138	0.08	0.496	0.925	0.76
💗	0.136	0.077	0.592	0.926	0.794
😉	0.123	0.076	0.327	0.918	0.698

😢	0.123	0.075	0.344	0.938	0.714
💜	0.122	0.07	0.456	0.927	0.747
😅	0.121	0.074	0.328	0.907	0.692
😄	0.119	0.072	0.34	0.942	0.715
😁	0.118	0.073	0.311	0.909	0.688
😈	0.118	0.073	0.302	0.942	0.703
😊	0.099	0.056	0.41	0.887	0.706
😉	0.099	0.06	0.273	0.918	0.679
😴	0.098	0.056	0.39	0.924	0.721
🤘	0.098	0.056	0.393	0.916	0.718
💪	0.096	0.055	0.364	0.92	0.71
😕	0.092	0.051	0.448	0.891	0.722
👋	0.092	0.054	0.312	0.93	0.698
😳	0.088	0.052	0.292	0.932	0.692
😐	0.084	0.047	0.417	0.89	0.71
😑	0.079	0.044	0.386	0.908	0.71
😋	0.075	0.043	0.292	0.923	0.686

Table of the most frequent 50 emojis and their corresponding results