Emoji Prediction via Word Embedding and Deep Learning

Yongyi Zhao
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213, U.S.
yongyizh@andrew.cmu.edu

Kerou Zhang
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213, U.S.
kerouz@andrew.cmu.edu

Jiarun Wei
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213, U.S.
jiarunw@andrew.cmu.edu

Tianyu Ren
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213, U.S.
tianyur@andrew.cmu.edu

Abstract

This project offers a sample procedure for predicting an emoji based on a natural sentence. It can be roughly divided into 2 parts: word embedding and deep learning part. The word2vec and GloVe methods are chosen for word embedding; The CNN and LSTM models are chosen for the deep learning model. It compares different combinations of embedding models and deep learning model from the perspective of computational efficiency, test accuracy. It also analyses different factors that influence the accuracy and computational efficiency of the whole procedure. It is shown from the experiment that the combination of LSTM+GloVe has the highest accuracy; the CNN model has higher computational efficiency than LSTM; the embedding vector size has little influence on the accuracy, and the external embedding source will lower the test accuracy.

1. Introduction

Emojis have become a visual texting language for online communication widely used in all kinds of social media like Twitter, Facebook, and Instagram. They help with getting our partner's feelings quickly, killing the vapidity of plain text, shortening the emotional distance between the two sides of the conversation, and have become more and more popular and indispensable in daily texting for their enjoyment, vividness and expressiveness. However, searching for appropriate emojis while text chatting can be time-consuming and affect user experience to some degree. Having an appropriate emoji bump out of the keyboard automatically can be joyful and efficient. Currently, the iOS keyboard and some other input methods can associate an emoji automatically with some highly emotional words like happy, sad, cry and etc. Apparently, predicting emoji based on a single word is less comprehensive and sometimes confusing than predicting emojis out of a whole sentence. To solve this problem, our project aims to build a text classifier to assign a sentence with a single emoji using ML algorithms. We used sentences from Twitter as the training and test set and applied two different word embedding methods, Word2Vec and GloVe, specifically with different embedding dimensions, and two deep learning algorithms, LSTM and CNN specifically and compared the different combination of embed- ding methods, embedding dimensions and algorithms from the perspective of training time, test accuracy and so on. According to our results, the less dimension embedded, the less training time is. And the combination of GloVe+LSTM gives the highest accuracy of 57%. Our results can be further applied to text reply automatically.

2. Related Work

2.1. Sentiment Analysis

Sentiment analysis (also known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, and computational linguistics to systematically identify and extract affective states and subjective information. It is a technique that is increasingly important in the area of monitoring customer satisfaction, detecting sentiment in emails, survey responses, social media data, and beyond. And emoji prediction is a fun branch of Sentiment analysis. There are basically five specific problems within the field of sentiment analysis [1]: Document-level sentiment analysis; Sentence-level sentiment analysis; Aspect-based sentiment analysis; Comparative sentiment analysis; and, Sentiment lexicon acquisition. Sentence-level sentiment analysis is what we are concentrated on in order to predict emojis out of a sentence. Sentence-level sentiment analysis usually analyzes subjective sentences, and most methods use supervised approaches to classify the sentences into two classes [4]. Other forms of sentiment analysis use unique, categorical labels similar to the emoji tags to predict those distinct emotions, such as happy, confused, sad, etc.

2.2. Twitter Emoji Prediction

S. Singh [3] has done twitter emoji prediction using GloVe+LSTM. However, it only uses one embedding algorithm and training model.

In this project, word2vec and CNN are added to the list of comparison. An optimal combination of embedding and training algorithms and analysis of factors which influence the accuracy of these algorithms are come up with.

3. Data

3.1. Dataset

As is shown in figure 1, we used the Twitter emoji prediction dataset, which contains about 25 thousand tweets with the corresponding emoji IDs of 20 categories. We also got a mapping file that maps the emoji ID back to the emoji photo. Before we do further steps, we merged the 20 classes of emojis into seven as many of them are either too similar to be distinguished or are duplicated to some degree. For example, as shown in figure2, No.1 and No.6 emojis are exactly the same camera emoji, and No.9, No.14 and No.18 are black, blue and purple heart respectively, which are very similar from the perspective of emotion.

3.2. Preprocessing

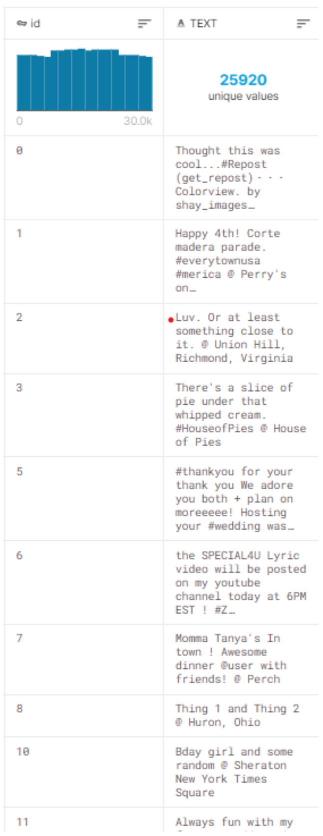
In order to make the corpus more understandable by the training model, we preprocessed the dataset before training. First, we removed special characters often used in Twitter, like ”” and ”@” and the word behind. In Twitter culture, ”” usually leads a subject or an event that people are discussing about, which is subjective like ”HeforShe” and is regarded as irrelevant to our emotion analysis. Also, as the word following ”” is often an abbreviation or compound word which can't be split by our model, and thus cannot be analyzed, we removed the word following ”” together with ””. Similarly, as ”@” is often used to indicate the location or the person one is tweeting at or to, and that information makes no contribution to the emotion, we removed the word following ”@” together with ”@” as well. Besides, we set all words to lowercase to eliminate the confusion caused by lowercase and uppercase spelling to the model and make the words unique.

4. Methods

4.1. Word embedding model

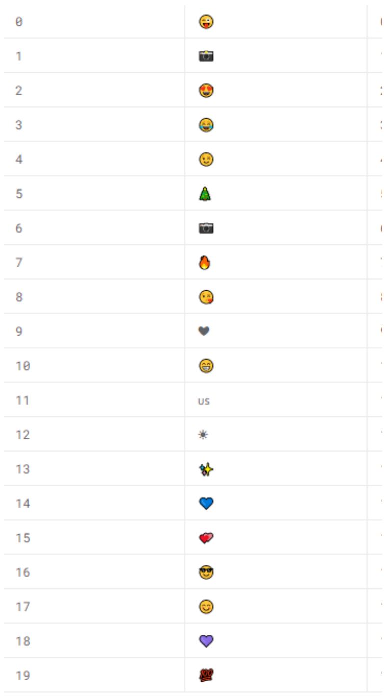
Once we are done with the preprocessing, we embedded our training dataset using 2 kinds of embedding algorithms-GloVe model and the word2vec model, respectively. Both models map each individual word in the training set to a vector and finally forms a weight matrix. This vector acts as the digital version feature of each word. The dimension of each vector can be customized, and it is really a tradeoff between insufficient information and overfitting. The major difference between GloVe and word2vec is that GloVe takes word frequency into consideration while word2vec does not. However, the word2vec model itself is a trained model, and it considers the relationship between context within a certain window size, while the GloVe model is not a trained embedding model and is frequency-based.

The flow chart shown in figure 3 shows a brief summary of the embedding process. The green boxes are the procedures, and the yellow boxes are examples illustrating the process: First, a unique integer is allocated as an identification number of each word in the corpus. For example, $'grape' \leftarrow 1$, $'you' \leftarrow 2$, $'eat' \leftarrow 72600$, etc. Then



id	TEXT
0 – 30.0k	25920 unique values
0	Thought this was cool...#Repost (get_repost) · · · Colorview. by shay_images…
1	Happy 4th! Corte madera parade. #everytownusa #merica @ Perry's on…
2	Luv. Or at least something close to it. @ Union Hill, Richmond, Virginia
3	There's a slice of pie under that whipped cream. #HouseofPies @ House of Pies
5	#thankyou for your thank you We adore you both + plan on moreeeee! Hosting your #wedding was…
6	the SPECIAL4U Lyric video will be posted on my youtube channel today at 6PM EST ! #Z…
7	Momma Tanya's In town ! Awesome dinner @user with friends! @ Perch
8	Thing 1 and Thing 2 @ Huron, Ohio
10	Bday girl and some random @ Sheraton New York Times Square
11	Always fun with my

Figure 1. twitter emoji prediction dataset



0	😜
1	📸
2	😍
3	😂
4	😉
5	🎄
6	📷
7	🔥
8	😘
9	❤
10	😁
11	us
12	☀
13	✨
14	💙
15	💕
16	😎
17	😊
18	💜
19	💯

Figure 2. Emoji mapping

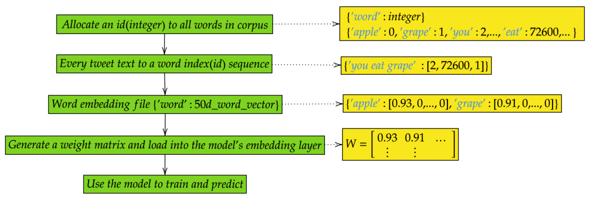

Allocate an id(integer) to all words in corpus
{'word' : integer}
{'apple' : 0, 'grape' : 1, 'you' : 2,..., 'eat' : 72600,... }
Every tweet text to a word index(id) sequence
{'you eat grape' : [2, 72600, 1]}
Word embedding file {'word' : 50d_word_vector}
{'apple' : [0.93, 0,..., 0], 'grape' : [0.91, 0,..., 0]}
Generate a weight matrix and load into the model's embedding layer
W = [0.93 0.91 ...]
Use the model to train and predict

Figure 3. Flow chart of the embedding process

once the words are concatenated to sentences, they will become vectors composed of integers. So the sentence 'you eat grape' will correspond to the vector [2,72600,1]. Next, the embedding vectors for each word in the corpus will be extracted and concatenated into a matrix and arranged according to the word's ID. For example, if apple's embedding is 0.93 etc. and grape's embedding is 0.91 etc., the weight matrix will look like this:

$$W = \begin{bmatrix} 0.93 & 0.91 & \dots \\ \vdots & \vdots & \end{bmatrix} \tag{1}$$

Finally, such a matrix will be used to train.

4.2. Training Model

For the training model part, we use CNN and LSTM, both deep learning models.

LSTM Classifier RNN (Recurrent Neural Network) is being extensively used on sequence data processing. Compared with traditional neural networks, RNN can analyze changes in sequence data. For example, the meaning of the same word may vary to different contextual contents. LSTM (Long Short-term memory), as a special kind of RNN, further addresses the issue of gradient disappearance and gradient explosion during long sequence training.

In our training model, the first layer is the embedding layer. The embedding layer serves as a converter that converts each word in a sentence into a meaningful vector. Specifically, the embedding layer loads the pre-trained GloVe or Word2vec embedding weights matrix. The input is a sequence of integers that represent individual words. Then the embedding layer transforms each integer i into the ith column vector of the embedding weights matrix.

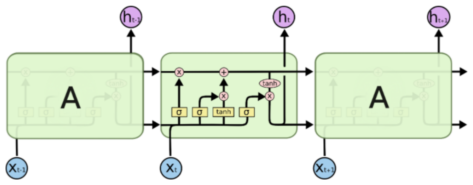


Figure 4. The repeating module of LSTM cell blocks [2]

Then we add the LSTM layer to the training model. In the LSTM network, traditional neural network layers are replaced by LSTM cell blocks. These cells consist of an input gate, forget gate, and output gate, as shown in figure 4. In each cell block, the new word value x_t is concatenated to the previously hidden state output h_{t-1} as the current cell block's input. First, the input goes through a sigmoid layer, also known as the forget gate layer. It outputs f_t, a real number ranging from 0 to 1, representing the degree to which the cell state C_t should switch off this input.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2}$$

Next, a sigmoid layer, known as the input gate layer, outputs i_t, which chooses which value to update. Also, a tanh layer generates a candidate cell state $\tilde{C}_t$, which will be added to the cell state. These two layers will determine which input information will be store in the cell state.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \end{aligned} \tag{3}$$

The former two parts become the remembering and forgetting components in the LSTM cell and cooperatively outputs the new cell state C_t.

$$C_t = f_t * C_t + i_t * \tilde{C}_t \tag{4}$$

Finally, the output gate layer, a tanh layer, controls the output of each cell.

$$
\begin{aligned}
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
\tag{5}
$$

Our proposed LSTM architecture looks like figure 5. The text data input has a shape of (batch size, number of time steps=word numbers in each sentence, hidden size=embedding vector length). This means that when we truncate each of our sentences into the same length, such as 15 words, choose a batch size of 128, and each embedding vector is of 50 length size. The input shape would be (128,15,50). We feed the input data into two layers of LSTM cells. The output of these cells is of the same size as the input data and then passed to a Dense layer. Finally, we apply a softmax activation, which outputs a vector representing the probability distribution on potential emojis.

When compiling the Keras LSTM model, we use 'categorical cross-entropy' for our multi-category classification task with only one truth for each data sample. The optimizer we use is Adam optimizer, and the metric is 'accuracy.'

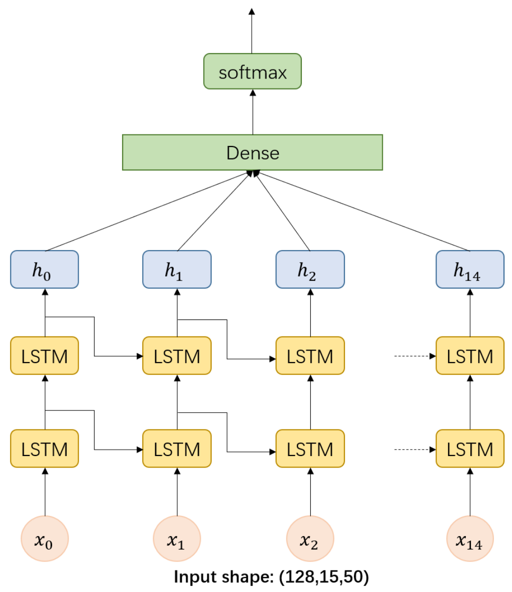


Figure 5. LSTM architecture

CNN Classifier CNN (Convolutional neural networks) is the backbone of Computer vision and is responsible for many significant breakthroughs in Image Classification. However, CNN also performs exciting results when applied on NLP tasks.

Convolutions are sliding window functions applied to a matrix, and we refer sliding window to a kernel or a filter. CNN are layers of convolutions with activation functions like ReLU so that it can model non-linear relationships. Rather than flatten all of the input data like traditional neural networks, CNN can extract spatial relationships. For NLP tasks, we usually use kernels that slide over word embeddings. Therefore, the kernels have the same length as the input data. However, the width of kernels, how many words are covered in that sliding window, generally range from 2 to 5. By analyzing neighboring words together, the model can better understand the meaning of the sentence. Also, CNN is capturing spatial relationships in sentences, which helps analyze sentiment and output corresponding emojis.

After the embedding layer, our proposed CNN architecture looks like figure 6.

We feed the input data into three convolution layers with a ReLU activation function, followed by max-polling layers. Then we stick two fully-connected layers. Finally, we output the model to the softmax activation function. When compiling, we still use 'binary cross-entropy' loss to train our model and 'accuracy' as the metric. We use the 'rmsprop' optimizer.

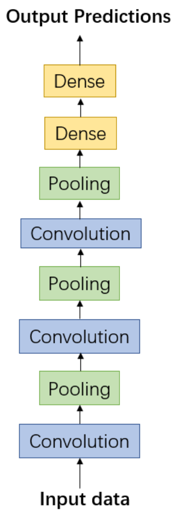


Figure 6. CNN architecture

5. Experiment

5.1. Algorithm Comparison

As is shown in figure 7, 4 combinations of word embedding and training algorithms are tested. The embedding vector size and training epoch are enforced to be 50 and 5 respectively, which are exactly the same among 4 combinations.

It is clearly shown that the combination of GloVe+LSTM has the best accuracy of approximately 57%.

One interesting thing to find is that LSTM does not always perform better than CNN. For example, under

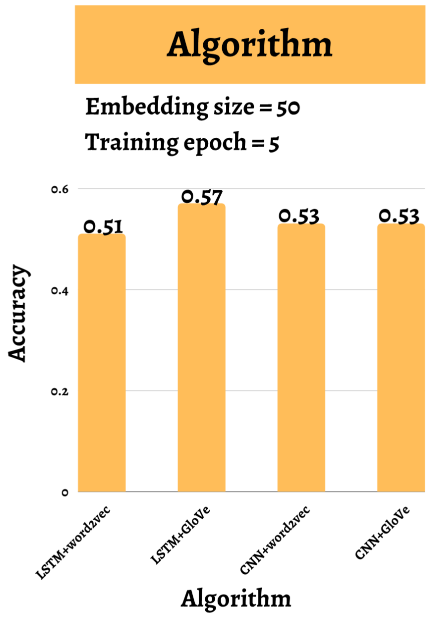


Figure 7. Algorithm comparison

word2vec embedding, the accuracy of CNN is 0.53 while that of LSTM is only 0.51.

A reasonable explanation would be that the advantage of LSTM, which considers a wider range of sentences, is weakened because the tone of Twitter is more like a keyword style, and few of them are complete sentences. Consequently, it makes CNN has the capability to achieve higher accuracy by only considering a few words within its window size.

Moreover, as shown in table1, LSTM also requires much more computing time. For embedding size of 50, it requires 6 times more computing time than CNN, and 9 times for embedding size of 300. It is because, for each word, LSTM considers much more context. Therefore, CNN might be more suitable for large dataset.

Computing time	dim = 50	dim = 300
CNN	19s	39s
LSTM	134s	386s

Table 1. CNN and LSTM computing time comparison with GloVe embedding and 5 epochs

5.2. Embedding Size Comparison

As is shown in figure 8, 4 kinds of word embedding vector sizes are tested. The algorithm and training epoch are enforced to be word2vec+CNN and 5 respectively, which are exactly the same among 4 embedding sizes.

It can be concluded that the word embedding vector size does not significantly affect accuracy. However, the accuracy will still have a discount when the embedding size is too large or too small, which is around 0.5%.

This is because when the embedding size is too small, there may be a lack of features. On the contrary, when the embedding size is too large, overfitting may occur.

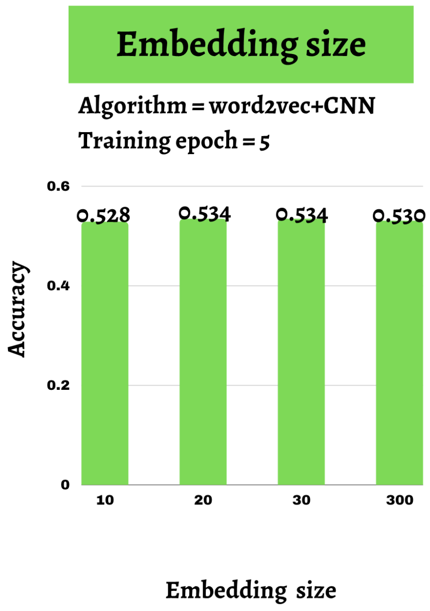


Figure 8. Embedding size comparison

5.3. The Influence of External Embedding

As is shown in figure 9, 4 combinations of embedding source and training algorithms are tested. The embedding vector size embedding, algorithm and training epoch are enforced to be 50, word2vec and 5 respectively, which remains unchanged among 4 combinations.

The external embedding source didn't help with the final accuracy. It even reduces accuracy.

This is possibly caused by too much useless information contained in the external embedding, which confuses the training model.

5.4. Error Analysis

Unlike complete articles, such as news reports or literature, tweets possess higher randomness and unpredictability. Sentences may not have a clear, logical relationship with each other. Also, some sentences are merely composed of a few words whose meaning can be somewhat confusing. For example, in the data set, a tweet says ”Born N Red Nuff Said” which is only understandable to the people who are familiar with the subject discussed.

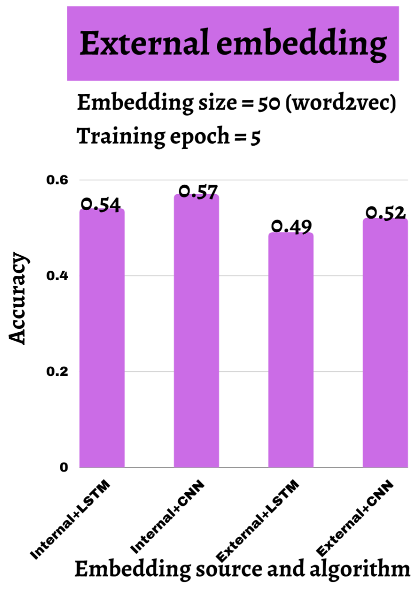


Figure 9. External embedding model comparison

Also, a single tweet may be represented by several different emojis at the same time. For example, if one says, ”Today’s traveling was so happy, check these awesome photos!” this tweet can be represented either by a ”happy face” emoji or a ”camera” emoji. In these cases, it is tough to determine which is the most suitable emoji and may result in misjudges.

Moreover, though the random seed for train-test-splitting is set to be the same for all cases, a single train-test-splitting may not be convincing enough. It is possible that the train set happens to be more understandable for LSTM than CNN, such as a train set containing many long tweets and long sentences, and vice versa. Validation methods such as KFold cross-validation may be used to derive more stable out-comings and more reliable results. Also, such practices may help to reduce the risk of overfitting.

6. Conclusion

In conclusion, two embedding methods and two deep learning algorithms present acceptable results, namely around 50% to 57% accuracy on the test set. And when we input some sample sentences, the system gives reasonable emoji feedback. LSTM generally has slightly better performance than CNN, and with GloVe embedding, it presents the best accuracy around 57%. However, LSTM requires 6 to 9 times higher training time than CNN algorithm and could have a lower accuracy when combined with word2vec. Also, increasing embedding size does not have a significant influence on accuracy. Using external embedding resources will reduce accuracy.

The randomness and unpredictability of tweet sentences may contribute to the error. And it is difficult to determine the most dominant emoji for a tweet that may possess different emotions. Moreover, some cross-validation methods may be implemented in future work to generate more reliable results and help avoid overfitting.

Contribution

All team members have equally contributed to the project. Team members are arranged by last name alphabetically.

Tianyu Ren Mainly responsible for implementing GloVe model.

Jiarun Wei Mainly responsible for implementing word2vec model.

Kerou Zhang Mainly responsible for implementing CNN model.

Yongyi Zhao Mainly responsible for implementing LSTM model.

Everybody Everybody in the team have contributed to the literature review, algorithm design, coding, presentation preparation and report writing. The project will not exist without each member in the team.

References

[1] Ronen Feldman. Techniques and applications for sentiment analysis. *Communications of the ACM*, 56(4):82–89, 2013.

[2] Christopher Olah. Understanding LSTM networks, Aug. 2015.

[3] Sukhwinder Singh. EmojiPredictionfromTweet, 2020.

[4] Hong Yu and Vasileios Hatzivassiloglou. Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 129–136, 2003.