CS577 Final Project (2022 FALL)
Twitter emoji prediction analysis

Hao Shen, Yuan Tian, Bonan Kou, Siyu Yao

I. Introduction

Emoji has become an indispensable part of language that is frequently used in cyberspace i.e., instant chatting apps and online forums. There has been a lot of research effort made to predict emoji usage in a given context. In this project, we propose and try to solve three research questions: 1) How do today's popular ML models perform in terms of emoji prediction? 2) Will additional features generated by extra models increase the performance? 3) How do the user feel when he or she can dynamically acquire a emoji feedback when typing a message.

To answer the first question, We experimented with a bunch of classifiers (transformers, multi-layer perceptrons, RNN, GRU, LSTM with their variations) on the task of emoji prediction. More specifically, we define our task to be selecting the best emoji that captures tone and sentiment of a given twitter. We evaluated performance of each classifier with different input representation on a dataset from Github with their corresponding emojis. We experimentally prove the advantage of transformer models on the task of emoji prediction over traditional neural networks. To answer the second question, we leveraged a pre-trained sentiment analysis model and tried to enhance our model by modifying the training data. However, we found this method doesn't affect the result a lot for our task. Third, limited by time and resources, we cannot conduct a formal user study, so we only developed a simple user interface, where the user could type in the command line and the corresponding emoji picture will be shown (close the previous picture window at the same time).

II. Related Work

A. Multi-resolution Annotations for Emoji Prediction

This paper talks about how emojis become more and more important in our daily life [5]. A data annotation method based on the self-attention mechanism in Transformer networks is used in the process of predicting emojis. We are inspired to use transformer after reading this paper.

B. Emoji prediction: Extensions and benchmarking

We gain the idea of what data we should use and how to evaluate our data from this paper [4]. It talks about annotate large corpus from Twitter posts for the prediction task. Twitter is probably one of the most popular plat form and each twitter is limited not to use too much words. In other words, the emoji that can represent a single twitter is easier to find and is more practical to use in real life. The evaluation in this paper focused on the change of accuracy with or without BERT-based model. We also compared the same model runs with Bag of Words and Word Embedding to see the difference.

C. Emoji Prediction from Twitter Data using Deep Learning Approach

This paper inspired us about the relationship between sentiment analysis and emoji prediction [3]. It treats emoji prediction as a similar type of task compared with sentiment analysis. In other words, instead of just categorizing each tweet with POS, NRU and NEG, it uses multi-class classification to classify the tweets. We are inspired that sentiment analysis might be helpful in this project. The result is not as expected and it's mentioned in the later parts.

D. Multimodal Emoji Prediction

This paper focused more on how to apply the result of the emoji prediction into the real life and what advantages and disadvantages it may bring [1]. Unlike the previous one, this paper focused on how emojis would be helpful in sentiment prediction, and thus analysis of emojis is of great significance.

E. Classifying the Informative Behaviour of Emoji in Microblogs

The authors of this paper focuses more on emoji analysis, including the redundancy of emojis [2]. With the help of this paper, we can take a deeper look at the content beneath each emoji. There is a useful approach mentioned in this one that also help us in our work, which is the use of thesaurus metrics combined with partial syntactic analysis was the most effective in classifying redundancy.

III. Model architecture

We experimented with 8 different ML models with different architectures (e.g. forward, recurrsive, attention-based). And we applied different embedding methods (e.g. bag of words, word embedding, sentence embedding). Furthermore, we enhanced the models by taking the sentiment as the additional feature when training them. To be more specific, we used a pre-trained sentiment analysis model to generate positive/negative tag for each sentence in our dataset, and feed train our model with the modified dataset. However, we found using sentiment as additional feature doesn't affect the result a lot (almost not change or even worse), so we won't show them here.

A. Fully connected NN (Bag of words)

For this approach, a neural network with three hidden layers is built. The size of the neural network is $972*600*300*100*20$. The learning rate is 0.02 and the number of epoch is 500. The input layer is determined by Bag of Words and contains vectors that represent each twitter inside the train set. Basically

we construct a vocabulary based on the train set and filter the words that appears less than a certain number. For this project specifically, the thresh hold we set is 50, and the size of input layer would become 972. we would also sort the words inside the vocabulary based on the number of times that they appear. Though with the use of it, the sequential information would be lost. The output layer is 20 since we have 20 emoji options for prediction. The output of the neural network would be a vector of size 20, and each entry represents the probability of a certain emoji.

The hyper parameter that we mainly tuned is the size of hidden layers, the learning rate, and the number of epochs. The size of hidden layers should contain as much information as possible, while not too slow to run. $600 * 300 * 100$ is the most efficient set that we found out. The learning rate is set accordingly. The number of epochs is set to make sure the model fit the data while not underfit nor overfit.

B. Fully connected NN (BERT sentence Embeddings)

Instead of using bag of word as the baseline, we also tried to replace the bag of words embedding with BERT-base embedding, which is hugely popular in NLP community, and pass it to the forward neural network mentioned in the previous subsection. To be more specific, we embedded each whole sentence from the twitter dataset to a vector with 768 dimension, and feed the embedding to a 3-layers neural network. The size of this neural network is $768 * 600 * 300 * 100 * 20$ with a learning rate of 0.001.

C. RNN

We also tried recurrent models such as RNN, GRU, LSTM, and we take RNN as the baseline of them. Instead of using sentence embeddings, we leveraged the pretrained word embedding and represented the whole sentence using the last output hidden state of the network since it theoretically incorporates all the previous information.

In this approach, for efficiency consideration, only the final hidden state will be captured as the representation and be passed to a single layer neural network for label decision. There are 2 hidden layers with dimension of 128 inside the RNN.

Since the computational efficiency of recurrent models are usually limited to its non-parallelism, to accelerate the training process of these recurrent neural networks (s, we leveraged the mini-batch method to separate our dataset. In this approach, the batch size is 16 and the learning rate is 0.007.

D. GRU

In order to go beyond the baseline, we replaced RNN (recurrent neural network) with GRU (Gated recurrent unit) and applied similar hyperparameters due to they having similar architectures. The learning is set to 0.009 and the batch size is set to 64.

E. LSTM

Further, we replaced GRU with LSTM (Long short-term memory). With similar architectures, the learning rate is set to 0.001 and the batch size is set to 64.

F. LSTM + NN (capturing all hidden states)

In the previous recurrent models, we only capture the last hidden state as the representation of the whole sentence. However, this method naturally reduces information since later words are more focused. To explore if the previous hidden states are helpful to improving the performance, we designed a variation of LSTM, which captures all the hidden states, and then warps them together and feed into a single layer forward neural network. To be more specific, there are hidden layers inside LSTM with dimension to be 256. Also, the learning rate is set to 0.001 and the batch size is set to 64.

G. BiLSTM

In addition to LSTM and the varied LSTM mentioned above, we also explored the performance of Bi-LSTM (Bidirectional Long short-term memory). Most hyper-parameters are remained similar to LSTM but the size of long term memory $c0$ and the short term memory $h0$ are doubled, so the input size of decision neural network is 512 x 20 rather than 256 x 20. The learning rate is set to 0.003 and the batch size is set to 80.

H. Transformer

Transformer is a neural network that learns context and thus meaning by tracking relationships in sequential data like the words in this sentence. We try a PyTorch implementation of transformer model with 2 hidden layers with 200 bits, 4 heads, and a dropout constant of 0.5 to avoid overfit. We use BERT to encode twitters as 768-bit vectors and use the BERT embeddings as input for the transformer model.

IV. Experiment Result

Note: The blue curve represents the training set, and the green curve represents the test set.

A. Fully connected NN (Bag of Words)

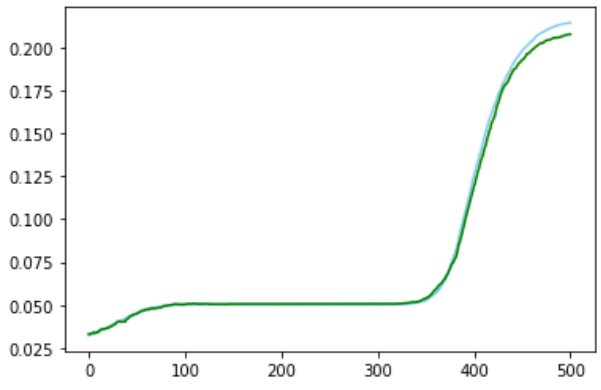

Fig. 1: Neural network result

The result of fully connected neural network with the use of Bag of Words is not great, but it is what we expected since it's used as a base line of other models in our project. The

highest accuracy that we achieved with this model is 20.069% on our test data set. It's good enough for a model that is not complex. Also the use of Bag of Words means that we choose to lose the sequential information, which would have a negative effect on the result based on what we found later. The existence of overfit is another possible cause for the low accuracy. We managed to achieve 21.668% of accuracy on the train data set. I also found that because I sort my vocabulary based on the frequency of each word, the words that appear most actually does not contain much information for emoji prediction. The words include "the", "I", "you", "this" and so on. The thresh hold we set for the word frequency can also be a factor to affect the result, since we need to seek the balance between noise or losing information.

B. Fully connected NN (BERT sentence Embeddings)

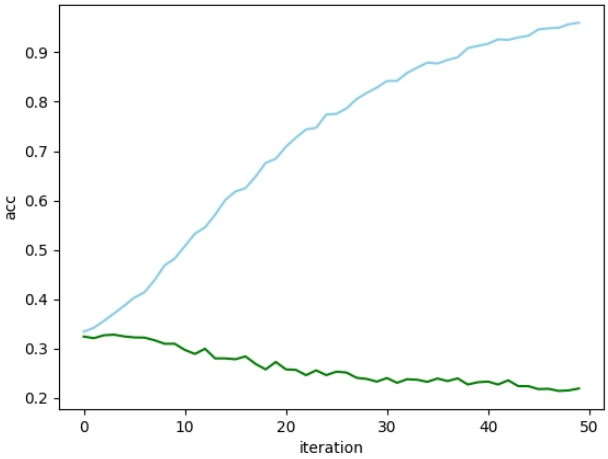

Fig. 2: NN + BERT result

In the experiment, we observed using BERT embedding can hugely speed up the convergence speed compared with all other models. In practice, we found an interesting behavior that the BERT embedding empower the model already perform well at very beginning, this may benefit directly from the good word embedding which well captures the sentiment of sentences. While training accuracy keeps increasing, the test accuracy starts decreasing due to overfitting. Another advantage is that BERT outperform other models in speed since the NN can effectively leverage parallelism of GPU compared to recurrent models, though it takes some time to load the pre-trained embedding..

C. RNN

As the baseline of recurrent models, as expected, RNN performs the worst compared to other recurrent models. As you can see in Fig. 3, it's even hard to converge for the training dataset.

We also observed that it sometimes converge very well due to the magic random initial weights.

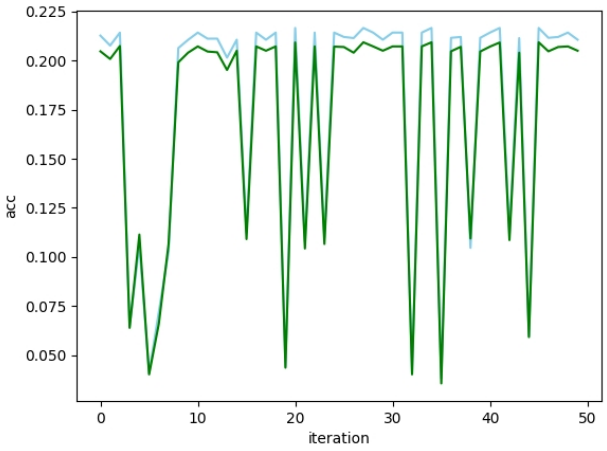

Fig. 3: RNN result

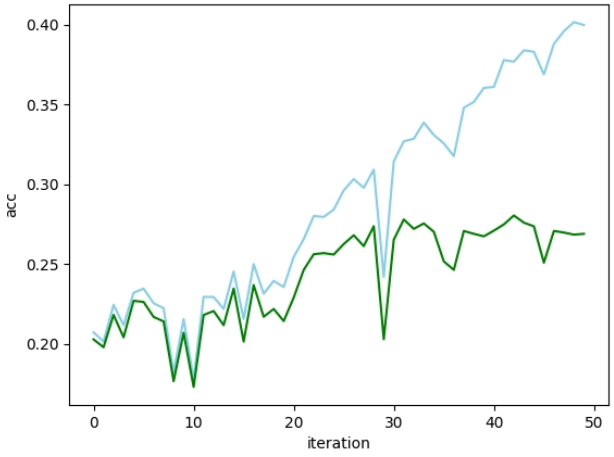

Fig. 4: GRU result

D. GRU

Compared with RNN, GRU performs much better, as shown in Fig. 4. The training accuracy continuously increases while the test accuracy fluctuates and slowly increases.

E. LSTM

As you can see in Fig. 5, with long term memory, LSTM performs even better than GRU. The training accuracy increase much faster and more stable than GRU. Within 50 epoches, the training accuracy quickly reaches 97%. The test accuracy first increases and then decreases due to overfitting, but the changing is also more stable.

F. LSTM + NN (capturing all hidden states)

As shown in Fig. 6, we can conclude that the model converges much faster for both the training dataset and test dataset. Compared to LSTM capturing only the last hidden state, it only takes 1/4 time to reach almost 99% accuracy for training dataset, and the accuracy for test dataset also changes relatively faster. Actually, if we take the learning speed as the criterion, this is the best one among all of our models.

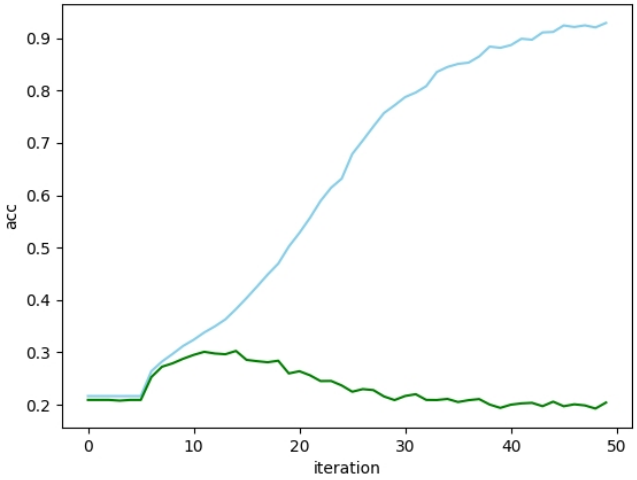

Fig. 5: LSTM result

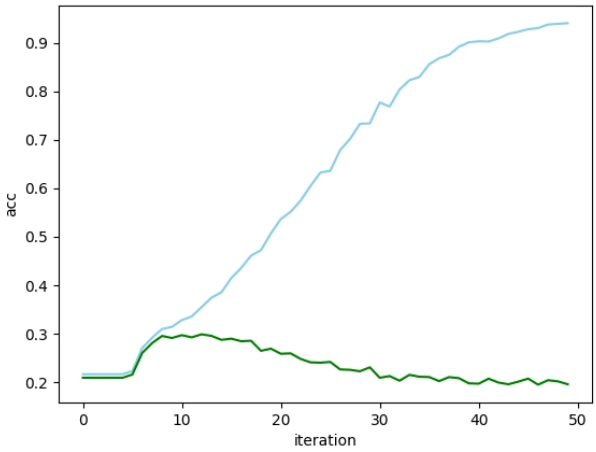

Fig. 7: Bi-LSTM result

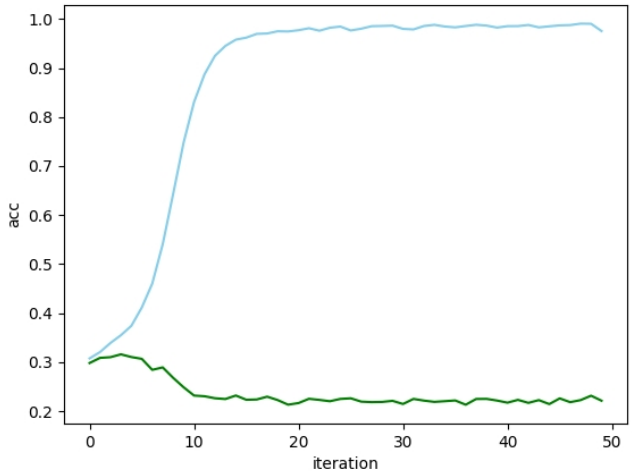

Fig. 6: LSTM result (capturing all hidden states)

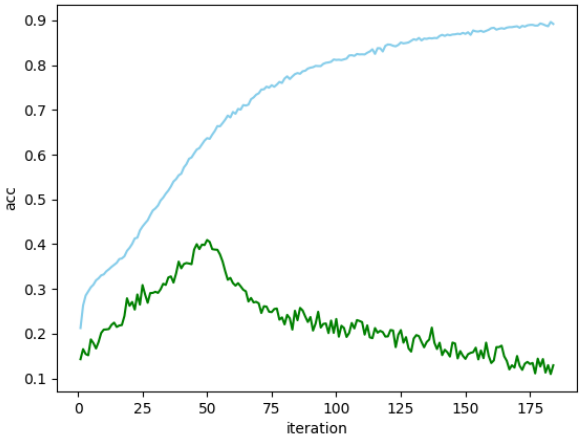

Fig. 8: Transformer result

G. BiLSTM

As you can see in Fig. 7, it's interesting that the performance of Bi-LSTM is very similar to LSTM, just with a slight distortion. We can conclude Bi-LSTM is not neccessary at all for this task because it doubles the number of parameters and greatly increase the training time when getting an almost same result.

H. Transformer

As you can see in Fig. 8, it's note-worthy that transformer model achieves the highest accuracy on the test set. The peak of accuracy of test set comes at epoch 49. A possible reason for the superior peak performance of transformer models and the fact that peak is achieved slower than other models is due to the complexity of transformer model itself. However, as more powerful

V. Dicussion & Results

In general, we found the accuracy of transformer model surpasses all other alternatives. Besides transformer model, the accuracy of LSTM and NN enhanced by BERT embedding comes after, while the vanilla NN (bag of words) renders the worst performance. According to the result, while the training accuracy keeps going up, none of the models achieved more than 40

Another interesting finding we have noticed is that transformer model has the slowest speed of convergence. It is possible due to the inherent complexity of transformer models. In the future, we hope to investigate more on the influence of input representation on the accuracy.

As for results, due to the reasons mentioned above, we have more focused on the first research question and will elaborate it in the following sections with charts as well as our analysis. In short, first, we assume the upper bound of performance for twitter emoji prediction is about 40% due to the shortage of information. Second, using sentiment as additional features cannot effectively enhance the performance for twitter emoji prediction. Third, based on our experience, dynamically getting emoji feedback will increase the fun of typing.

VI. Work partition and contribution

In this project, the following submodules are accomplished:

1) Conducted experiments using Neural networks, user interface, writing report (Hao Shen)
2) Conducted experiments using RNN, GRU, LSTM with their variations, and writing report (Yuan Tian)
3) Conducted experiments using Transformer and writing report (Bonan Kou)
4) prepared presentation slides and writing report(Siyu Yao)

References

[1] Francesco Barbieri, Miguel Ballesteros, Francesco Ronzano, and Horacio Saggion. Multimodal emoji prediction. *arXiv preprint arXiv:1803.02392*, 2018.

[2] Giulia Donato and Patrizia Paggio. Classifying the informative behaviour of emoji in microblogs. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[3] VN Durga Pavithra Kollipara, VN Hemanth Kollipara, and M Durga Prakash. Emoji prediction from twitter data using deep learning approach. In *2021 Asian Conference on Innovation in Technology (ASIANCON)*, pages 1–6. IEEE, 2021.

[4] Weicheng Ma, Ruibo Liu, Lili Wang, and Soroush Vosoughi. Emoji prediction: Extensions and benchmarking. *arXiv preprint arXiv:2007.07389*, 2020.

[5] Weicheng Ma, Ruibo Liu, Lili Wang, and Soroush Vosoughi. Multi-resolution annotations for emoji prediction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6684–6694, 2020.