# Natural Language Processing Midterm Project : Emoticon Prediction using Recurrent Neural Networks

*Yu-Hsuan Wang*[1], *Sheng-syun Shen*[2], and *Chia-hsing Hsu*[2]

[1]Graduate Institute of Computer Science and Information Engineering
[2]Graduate Institute of Communication Engineering
**Team Name:** 半夜大師敲門

## I. INTRODUCTION

When people write articles on the Internet, they might possess some emotions. For example, some people will write down the things that make them happy. In the case, their emotions during writing are likely to be happy, since they are thinking about the events which make them happy in order to write it down. Thus, the words they use would tend to be positive, like "excited", "cool", etc. In other words, humans' usages of words might depend on their emotion during writing and possess some sort of features. As a result, readers could predict authors' emotion when they read the authors' sentence. To be more specific, when there is a sentence "John scolded me this morning... Today sucks!", we are very confident that the author must be in a bad mood when he/she is writing down the sentence.

The project is about predicting authors' emotion according to the sentences they wrote. Each sentence contains text and an emoticon. Emoticon is a feature introduced by Yahoo! that allows user to express their emotion through a small icon in addition to text. Thus, emoticons can be viewed as authors' emotion when they wrote the sentence.

In recent years, deep learning has achieved great performance in a variety of tasks, including speech processing, machine translation, etc. In the project, deep learning is utilized to handle the emotion prediction problem. We encode sentences into a sequence of vectors and formulate the project as a multi-class classification problem.

## II. PROPOSED APPROACHES

### A. System Overview

First, we tokenize the input sentence into meaningful tokens. After tokenization, we pre-process the tokens with different pre-processing techniques. Lastly, we encode each pre-processed token into a one-hot vector and send it into our deep learning models.

In the respect of deep learning models, we try three types of recurrent neural networks, including simple recurrent neural network (RNN), long short-term memory (LSTM) and gated recurrent units (GRU). In addition to recurrent neural networks, we also try to add attention mechanism on the models and see if the mechanism helps our models achieve better performance.

### B. Pre-processing

We observe that some of the given sentences contain some meaningless tokens, such as punctuation marks, which might not be helpful for emotion prediction. In addition, the tokens increase the sentences' length and thus increase the computation costs during our training. What's worse, the tokens might be noisy and thus worsen our emotion prediction performance. Therefore, we believe that pre-processing is needed before we train our deep learning model.

We pre-process our data based on two assumptions: 1. An emoticon is determined by the text before it. 2. If an emoticon is the first token of a sentence, then it must be determined by the text after it. Our first assumption is based on our own experience of using emoticons. We use emoticons when we finish a sentence and we want to express our emotion to others clearly. Thus, emoticons can be viewed as indicators of our emotion about the text we wrote before it. The second assumption trivially holds true. If there is no text before emoticons, the emoticons must be determined by the text after it.

To be more specific, our pre-processing methods involve with three different aspects: meaning of token itself, term frequency and scopes. For each aspect, we introduce a pre-processing method. In other words, there are three pre-processing methods in total.

For meaning of token itself, we observed that there are four kinds of tokens in corpus: Chinese characters, English letters, punctuation marks and others. We only accept Chinese characters, English letters and two kinds of punctuation marks, and ignore the rest. We believe that Chinese character is the most important element to identify emoticons since the authors of the sentences are all native Madarin speakers. However, we have also observed that there are sentences containing only English, such as the one with rid 2858.("I'm so happy EMOTICON .hahaha") In addition to this example, there are also sentences that Chinese characters are useless while English letters contain important information during emotion prediction, such as the one with rid 4064.(超..... happy .... 勹啦 ！！EMOTICON) As a result, we consider English letters as well. Last but not least, although most of punctuation marks serve only as grammar purpose, some punctuation marks also provide clear emotion information about authors' emotion whereas Chinese characters seem ambiguous, such as the one with rid 5076.(怎？？？？EMOTICON) Therefore, we regard exclamation mark and question mark as such punctuation marks, and take them into our consideration during our emotion prediction.

In the respect of term frequency, we accept only the most frequent 60000 terms. (Except for function words) The pre-processing method assumes that the importance of a token is based on its term frequency. If a token's

term frequency is low, its importance is also low since it does not appear in other sentences and thus we cannot use it to predict emoticon. In the view of information retrieval, according to Zipf's law, the most important terms are those having high-level term frequency. (However, the most frequent terms are not important since they are very likely to be function words) In addition, by considering only high frequency terms, we can significantly decrease the size of our vocabulary to reduce the memory usage during our training.

Finally, our last pre-processing techique is the consideration of scopes. The method is based on an assumption that authors used emoticons depend only on the text near the emoticons, instead of the whole sentences. For example, the sentence with rid 2955 is long and contains different emotions in different parts of the sentence. However, the emoticon is determined by the tokens just right before it. (!!氣.....) If we consider the whole sentence, our prediction would be ambiguous. On the other hand, if we consider only the three tokens before the emoticon, we can clearly predict the correct answer. As a result, we should limit the scope of our consideration when we predict emotion.

### C. Recurrent Neural Networks

In this section, we will give a simple introduction of different RNN architectures we implemented in this project.

*1) Simple Recurrent Neural Network:* Recurrent Neural Networks (RNN) have gained attention in NLP field. A RNN consists of an input layer, a hidden layer with a recurrent connection and an output layer. The recurrent connection allows the propagation through time of information about the state of the hidden layer. Given a sequence of tokens, a RNN takes as input the one-hot encoding $x_t$ of the current token and predicts the probability $y_t$ of emoticon. Between the current token representation and the prediction, there is a hidden layer with $m$ units which store additional information about the previous tokens seen in the sequence. More precisely, at each time $t$, the state of the hidden layer $h_t$is updated based on its previous state $h_{t-1}$ and the encoding $x_t$ of the current token, according to the following equation:

$$h_t = \sigma(Ax_t + Rh_{t-1}), \tag{1}$$

where $\sigma(x) = 1/(1 + exp(x))$ is the sigmodi function applied coordinate wise, $A$ is the $d \times m$ token embedding matrix and $R$ is the $m \times m$ matrix of recurrent weights. Given the state of these hidden units, the network then outputs the probability vector $y_t$ of the next token, according to the following equation:

$$y_t = f(Uh_t), \tag{2}$$

where $f$ is the softmax function and $U$ is the $m \times d$ output matrix.

The model is trained by using stochastic gradient descent method with back-propagation through time. We use gradient re-normalization to avoid gradient explosion. In practice, this strategy is equivalent to gradient clipping since gradient explosions happen very rarely when reasonable hyper-parameters are used. The details of the implementation are given in the experiment section.

*2) Long Short-Term Memory:* The simple RNN has the ability to capture sentence information. However, the length of reachable sentence is often limited. The gradient tends to vanish or blow up during the back propatagion(Bengio et al., 1994; Pascanu et al., 2013). An effective solution for these problems is the Long Short-Term Memory (LSTM) architecture(Hochreiter and Schmidhuber, 1997; Gers,2001). Such architecture consists of a set of recurrently connected subnets, known as memory blocks. Each block contains one or more self-connected memory cells and the input, output and forget gates. Once an error signal arrives Constant Error Carousel (CEC), it remains constant, neither growing nor decaying unless the forget gate squashes it. In this way, it solves the vanishing gradient problem and learns more appropriate parameters during training.

Moreover, based on this structure, the input, output and stored information can be partial adjusted by the gates, which enhances the flexibility of the model. The activation of hidden layer rely on the current/previous state, previous hidden activation and current input. These activation interact to make up the final hidden outputs through not only additive but also element-wise multiplicative functions. Such structures are more capable to learn a complex composition of word vector than simple RNNs.

The LSTM networks are described by the following composition function:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i), \tag{3}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f), \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \tag{5}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \tag{6}$$

$$h_t = o_t \odot tanh(c_t), \tag{7}$$

where $\sigma$ is the logistic sigmoid function. $i$, $f$, $o$ and $c$ are respectively for the input gate, forget gate, output gate, and memory cell activation vectors, all of which have the same size as the hidden vector $h$. The symbol $\odot$ denoted the element-wise product of the vectors. The weight matrices from the cell to gate vectors are diagonal, so element $m$ in each gate vector only receives input from element $m$ of the cell vector. The weight matrices from input, hidden, and outputs are not diagonal.

*3) Gated Recurrent Units:* A gated recurrent unit (GRU) was proposed by Cho et al. to make each recurrent unit to adaptively capture dependencies of different time scales. Similarity to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cells.

The activation $h_t^j$ of the GRU at time $t$ is a linear interpolation between the previous activation $h_{t-1}^j$ and candidate activation $\bar{h}_t^j$:

$$h_t^j = (1 - z_t^j)h_{t-1}^j + Z_t^j \bar{h}_t^j, \tag{8}$$

where an update gate $z_t^j$ decides how much the unit updates its activation, or content. The update gate is computed by

$$z_t^j = \sigma(W_z x_t + U_z h_{t-1})^j \tag{9}$$

This procedure of taking a linear sum between the existing state and the newly computed state is similar to

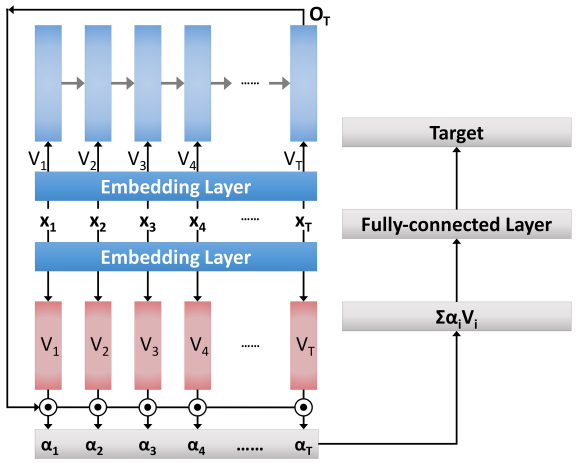


Fig. 1. Architecture of the proposed Neural Attention Model.

the LSTM unit. The GRU, however, does not have any mechanism to control the degree to which its state is exposed, but exposes the whole state each time.
The candidate activation $\bar{h}_t^j$ is computed similarly to that of the traditional recurrent unit,

$$\bar{h}_t^j = \tanh(Wx_t + U(r_t \odot h_{t-1}))^j \tag{10}$$

where $r_t$ is a set of reset gate and $\odot$ is an element-wise multiplication.

### D. Attention Mechanism

Recently, attention-mechanism has been incorporated with recurrent neural networks, and has shown significant improvement on a great variety of tasks. It's able to take care about the position of input elements, and some previous analysis also shows the potential of attenuating the unimportant parts and detecting the points in the entire input data. We thus explore the use of attention mechanism for solving the emoticon prediction task. We duplicated a recently proposed model which is suitable for every sequence classification problems, and the details are shown in this paper[1].

In the upper part of Figure 1, we demonstrate the encoding procedure to transform input sequences into fixed-length vector representation $O_T$. The set $x = (x_1, x_2, \ldots, x_T)$ denotes the input sequence, where $T$ is the sequence length. Each element in $x$ represents a fixed-length feature vector. In order to reduce the model complexity, we set an embedding layer, a linear transformation matrix, to turn the inputs into low dimensional dense vectors $V = (V_1, V_2, \ldots, V_T)$, and then they will be sent to the recurrent encoder. In each time step, the recurrent network takes one element $V_i$ from feature vector set, and after processing the last element, it then generates an output vector $O_T$, which can be regarded as the summaries of the preceding feature vectors.

When input sequence $x$ is long, the summaries vector $O_T$ is likely to contain noisy information from many irrelevant feature vectors $V_i$, we thus apply attention mechanism to select only relevant frames among the entire sequence. The procedures are shown in the lower part of Figure 1. There is also an embedding layer to transform input sequences into dense vectors, and all the parameters in the embedding layer are shared with the previous one. We then calculate the cosine similarity between the sequence vector $O_T$ and word embedding set $V$:

$$e_i = O_T \odot V_i, \tag{11}$$

where $\odot$ denotes cosine similarity between two vectors. As a result, we have a list of score $e = (e_1, e_2, \ldots, e_T)$. The attention weights $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_T)$ come from the normalized score list $e$. The normalization function is shown below:

$$\alpha_i = \frac{\sigma(e_i)}{\sum_{i=1}^{T} \sigma(e_i)}, \tag{12}$$

where $\sigma$ denotes the sigmoid activation function. We weighted sum all the feature vectors as $\sum \alpha_i V_i$, and sending it to a fully connected layer. Therefore, we can predict our target.

## III. EXPERIMENTS and DISCUSSIONS

### A. Experimental Setup

We are given a corpus of Yahoo Blog containing 300,000 sentences, and every sentence are labeled with one emoticon label. There are 40 emoticon classes in total. We use the famous tokenization toolkit, jieba, for tokenizing the corpus. The first part of our deep learning model is an embedding layer which encodes a one-hot vector, whose dimension is the same as the size of vocabulary, into a vector with dimension = 256. Then the embedded sequence of vectors is passed into a recurrent layer with dimension = 128. In the end, the output of the recurrent layer is passed into an 40-dim softmax activation layer to classify the input sentence. We use a famous deep learning toolkit, Keras, to implement our deep learning models. We split our original training set to our training set and development set, according to ratio 9:1.

Due to our limit of computation capability, we limit the sentence length to 1,500. In other words, if the input sentence consists more than 1,500 tokens, we will only consider the first 1,500 tokens and ignore the rest.

When we use our third pre-processing method, we set our scope to 50. That is to say, we only consider the nearest 50 tokens during our predictions.

| | Raw Data | Chinese+ English+!? | Chinese (Scope=50) | Top 60000 Frequency Terms |
|---|---|---|---|---|
| RNN | 0.18460 | 0.26156 | **0.26216** | 0.25519 |
| LSTM | **0.29144** | 0.28756 | 0.28786 | 0.28254 |
| GRU | 0.29315 | 0.29312 | **0.29375** | 0.29122 |
| Attention LSTM | **0.28689** | 0.28206 | 0.28317 | 0.28441 |
| Attention GRU | **0.28631** | 0.28276 | 0.28616 | 0.28318 |

TABLE I
EXPERIMENT RESULTS

### B. Discussion: Pre-processing

From Table 1, we can see that the most useful pre-processing method is limiting the scope of consideration. However, some models' best performances are achieved by using raw corpus without any pre-processing methods.

The potential explanation for raw corpus is powerful is our computation limitation. We limit the scope of our

[1] https://arxiv.org/pdf/1604.00077v1.pdf

consideration to 1,500 in order to train our model in practical time. (Except for the one with scope = 50, of course) Therefore, one of the raw corpus major defects is eliminated since we would not see too many noisy tokens. (The sentence length of raw corpus can be as long as 4,937 tokens...)

Of course, there are other reasons that could explain such differences. Some tokens which are not considered by our pre-processing methods are useful during prediction, such as "@@." On the other hand, there are some useless tokens which have been taken into consideration by our pre-processing methods such as URL. Since the raw corpus takes everything into consideration, it has full information and thus balances the effects of considering noisy tokens such as URL, and thus outperforms our pre-processing methods. In the respect of top 60,000 frequency term pre-processing method, although it reduce the vocabulary size from 129,690 to 60,000, the method might ignore too many tokens. Some of the ignored tokens might be crucial during emotion prediction.

Nonetheless, we can view the performances of raw corpus and our pre-processing methods in a different angle. Although our pre-processing methods' performances are inferior to the performance of raw corpus, they are only slightly lower. As a result, we can claim that the pre-processed corpus is the core element of the original corpus, or the performances of our pre-processing methods will be significantly inferior to the one of raw corpus. The contribution of our pre-processing methods is reducing the size of corpus, thus reduce computation costs during training, while keeping comparable prediction performance, not to mention in some cases our pre-processing methods even outperform the raw corpus.

### C. Discussion: Different Deep Learning Models

Based on different pre-processing methods, we also compare the results using different deep learning techniques. We can discover that while speaking of the original recurrent neural network models, simple RNN performs worst. It's not surprising cause the gradient vanishing problem always degrades the performance of simple RNN. We also found that the performance of GRU networks is slightly better than the LSTM networks. This may results from the average length of the input sequences. LSTMs are known for handling very long sequences. However, in this task, the average length is about 20 words for each input. Therefore, LSTMs might not dominant in experiments.

While incorporating attention mechanism with recurrent neural networks, the hybrid model didn't achieve any improvement in experiments. Attention mechanism are capable of handling extremely long sequences, which is well known for solving question answering tasks, but the complexity of emoticon prediction task seems too low for attention models. As a result, over-fitting will easily degrade the performance of attention models in this task.

### D. Discussion: Assembling

In addition to models mentioned in Section 2, we have also tried to utilize multiple models simultaneously to see if the combination is helpful. For a sentence s and an emoticon prediction e, the score of the prediction is the weighted sum of multiple models' prediction scores (say M models).

$$\sum_{i=1}^{M} w_i Model_i(s, e) \tag{13}$$

We use LSTM and GRU with four different corpus respectively. That is to say, we use totally eight models for our prediction. We tune the models' weights on our development set, and the weights of each model are assigned as the following table. It can be shown that the models' weights do not necessary depend on their performance. For example, although top 60,000 frequency terms LSTM model's performance is inferior to Chinese (Scope = 50) LSTM model, its weight is heavier than the latter one. The models should be diverse in order to gain better interpolation performance.

| | Raw Data | Chinese+ English+!? | Chinese (Scope=50) | Top 60000 Frequency Terms |
|---|---|---|---|---|
| LSTM | 1 | 0.2 | 0.4 | 1 |
| GRU | 1 | 0.2 | 1 | 1 |

TABLE II
WEIGHTS FOR EACH MODEL IN INTERPOLATION

It is not surprising that the performance of assembling is higher than each of the individual models. We can view interpolation process as a voting by a group of experts. Although each expert might make mistakes, other experts can correct the mistake through voting. As a result, with assembling, we achieve our best performance **0.30823**.

### E. Discussion: Emotional Vocabulary

To search top ten vocabulary of every emoticon, We applied `Word2Vec`, a word-embedding toolkit proposed by Tomas Mikolov, and calculating cosine similarity to compute the similarity between the emoticon and words. From Table 3 ,we can see the top five vocabulary of every emoticon, the details of all emoticons are given in the appendix (emoticon_top_ten.pdf).

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Emoticon1 | 埃 | 哇哩 | OH | 美滿 | 協助 |
| Emoticon2 | 哇哩 | 埃 | 咳 | さ | 嘎 |
| Emoticon3 | 埃 | 好大 | 協助 | 奕 | 過著 |
| Emoticon4 | ? | 哪有 | 哇哩 | 怎辦 | 蝦米 |

TABLE III
EMOTICON TOP FIVE VOCABULARY

## IV. CONCLUSIONS

In this project, we explored the use of machine learning based approaches, rather than traditional natural language processing techniques. We also tried different pre-processing methods for reducing the noisy problem in data set. After assembling different models we implemented in experiments, we had additional improvement. Our best score is **0.30823**, which is also the second-best performance in Kaggle Leaderboard.