Sentiment Analysis using DNN and LSTM

Juan Chen
chenjuanhello@gmail.com

1. Introduction

Language modeling is getting a lot of importance due to the recent development of conversational agents such as chatbots and voice recognition devices. Sentiment analysis is an important step towards comprehension in natural language processing, and data sources such as movie reviews and emoji categorizations are convenient sources of highly polarized sentences for use. It focuses on understanding the positive or negative tone of a sentence based on sentence syntax, structure and content.

Language problem is not an easy task mainly because of two reasons: sparsity and ambiguity. It usually involves big vocabulary and results in a large sparse matrix. This work aims to predict one out of five categories of emoji sentiments, and uses a more powerful method, word embedding, to pre-process the data. A simple 2-layer network (deep neural network) is built to fit the data and a more complex 2-layer LSTM (recurrent neural network) model is then tried to account for the effect of sequence in words.

2. Data Description

The emoji data has 132 samples in training set and 56 samples in test set. Each sample consists of one sentence and corresponds to one of 5 emoji sentiments: ❤️ (love), ⚾(sport), 🍴(food), 😄(happy), and 😞(sad).

Table 1: Data Samples

Data sample	Label
I love you mum	❤️
I like you a lot	❤️
the game just finished	⚾
he is the best player	⚾
So sad you are not coming	😞
How dare you ask that	😞
I am proud of your achievements	😄
This is so funny	😄
I will have a cheese cake	🍴
do you want to join me for dinner	🍴

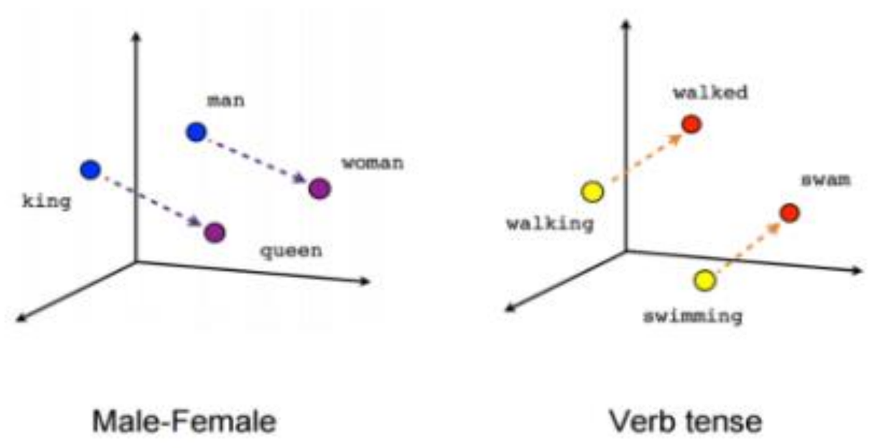
3. Data Pre-processing

Instead of using the nltk package and TF-IDF method to vectorize the words into features for model input, a powerful method, word embedding, is used to represent words in a continuous vector space where semantically similar words are mapped to nearby points. It is quite useful especially for our small dataset as the input vectors have semantic meanings.

There are a lot of public models trained on much larger datasets that create such vectors. We chose GloVe, an unsupervised learning algorithm that trains a vocabulary of 400,000 words and projects each word into a 50-dimensional space. GloVe is able to capture significant semantic and linguistic relationships. For instance, the difference vector between 'man' and 'woman' is found to be approximately parallel to the difference vector between 'king' and 'queen' -- making it well-suited to our task. The pre-trained GloVe is downloaded from:

https://nlp.stanford.edu/projects/glove/

Figure 1: Word Embedding



4. DNN Modeling and Evaluation

Each sample in our dataset has multiple words, and we project each of these words into a 50-dimension vector and take average, resulting in a single 50-dimension vector for each sample.

The first model we use is a simple 2-layer network for classification with softmax activation function. This is a very simple DNN model. Figure 2 plots the model performance. The left plot is loss vs. epoch, both the training and validation loss decrease nicely and began to flat out at around 300 epochs. The right plot is accuracy plot. The validation accuracy stops at around 82%.

With this simple 2-layer network, there are certain misclassifications because the model cannot correctly take into account the sequence of the words. For example, the last sample in Table 3 has a ‘not’ in the sentence but the 2-layer DNN cannot correctly capture its meaning and wrongly predicts it as 😄.

Table 2: Code Snippet of DNN Model

```
emoji_model = build_nn_model(
            input_dim=50,
            layers=[50],
            output_dim=5)
emoji_model.compile(
             loss='categorical_crossentropy',
            optimizer='adam',
            metrics=['accuracy'])
```

Figure 2: DNN Model – Loss and Accuracy

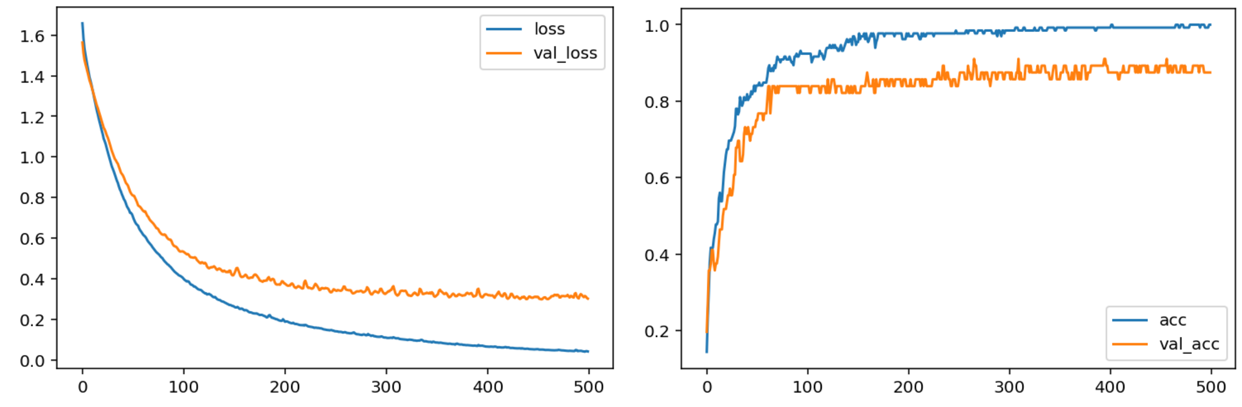


Table 3: DNN Model – Sample Predictions

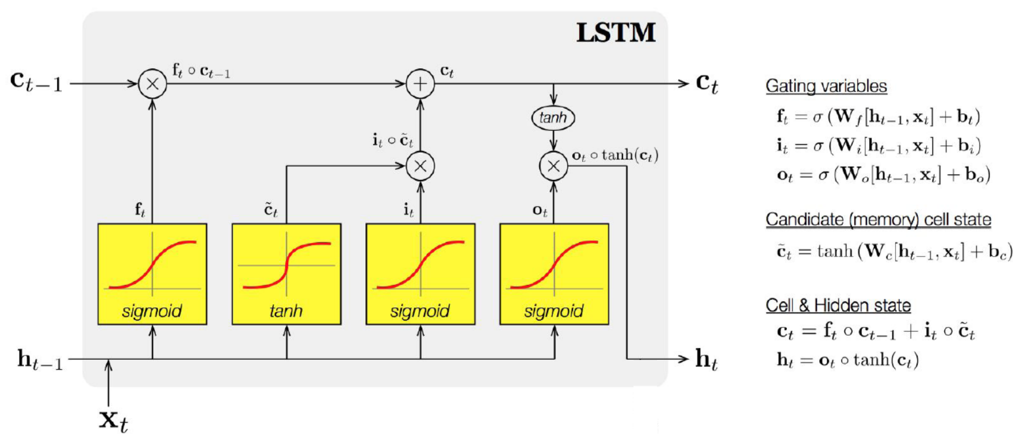
Data sample	Prediction
I love you	❤️
It’s horrible	😞
funny lol	😄
lets play with a ball	⚾
food is ready	🍴
i am not feeling great	😄

5. LSTM Modeling and Evaluation

In order to solve the word sequence problem as shown in the misclassified sample, we try to build an RNN model, the long short term memory (LSTM) network, that is quite useful for sequence input.

LSTM is a new type of RNN architecture and used to learn long-term dependency. The main difference from basic RNN unit is that LSTM has more complex node design. It adds 3 gates to control how much information gets through. They are forget gate, input gate, and output gate, all of which are composed out of a sigmoid neural net layer. Forget gate is applied on the previous cell state to control how much of previous state enters the current cell state, input gate controls how much current input enters the current cell state, and output controls how much to output. Using these 3 gates, the flow of information is controlled. A value of zero from the sigmoid means "let nothing through," while a value of one means "let everything through". These 3 gates allow the network to prioritize between what is 'important' and what is 'not-important', so important information at the beginning can be carried on to the point where it is needed. It is well-suited to process sequence input where the relevant information has gaps.

Long-Short Term Memory (LSTM) unit


LSTM
$\mathbf{c}_{t-1}$
$\mathbf{f}_t \circ \mathbf{c}_{t-1}$
$\mathbf{c}_t$
tanh
$\mathbf{i}_t \circ \tilde{\mathbf{c}}_t$
$\mathbf{o}_t \circ \tanh(\mathbf{c}_t)$
$\mathbf{f}_t$
$\tilde{\mathbf{c}}_t$
$\mathbf{i}_t$
$\mathbf{o}_t$
sigmoid
tanh
sigmoid
sigmoid
$\mathbf{h}_{t-1}$
$\mathbf{h}_t$
$\mathbf{x}_t$
Gating variables
$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_t)$
$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$
$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$
Candidate (memory) cell state
$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$
Cell & Hidden state
$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t$
$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t)$

LSTM takes each word as an input. As shown in Table 4, both layers have 128 nodes, and I use dropout to reduce overfitting. The convergence is bumpier than the DNN model, as there are more parameters to train. It tries to find the directions, but eventually it is able to converge. Compare the performance with DNN as shown in Table 5, there are not much difference, only a small increase in accuracy. But the main improvement is that it is able to correctly predict the last sample, meaning the sequence of the input is captured by the model to make the right prediction.

Table 4: Code Snippet of LSTM Model

```
X = LSTM(128, return_sequences=True, recurrent_dropout=0.5)(embeddings)
X = Dropout(rate=0.2)(X)
X = LSTM(128, recurrent_dropout=0.5)(X)
X = Dropout(rate=0.2)(X)
X = Dense(5, activation='softmax')(X)
```

Table 5: Model Performance of DNN and LSTM

		Loss	**Accuracy**
DNN	Train	0.047	0.99
	Validation	0.288	0.82
LSTM	Train	0.013	0.99
	Validation	0.464	0.89

Figure 3: LSTM Model – Loss and Accuracy

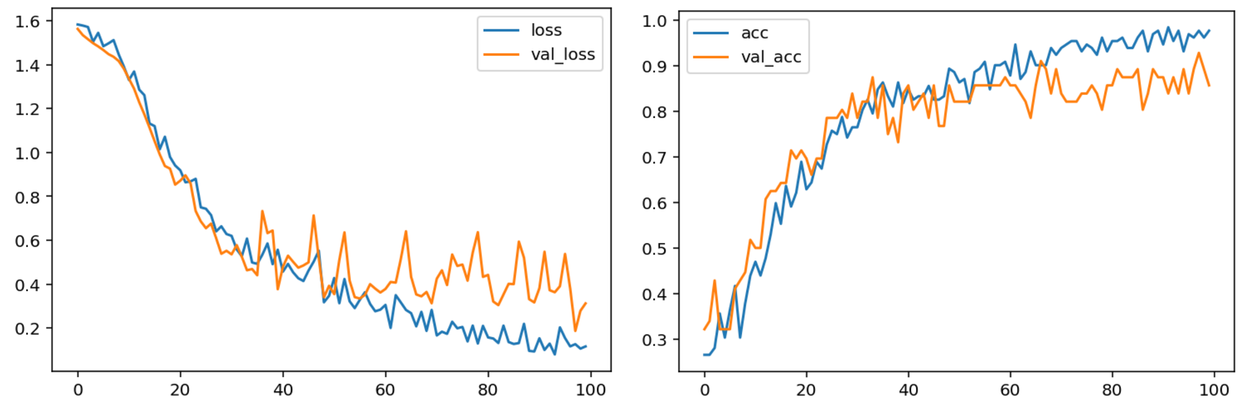


Table 6: LSTM Model – Sample Predictions

Data sample	Prediction
I love you	❤️
It's horrible	😞
funny lol	😄
lets play with a ball	⚾
food is ready	🍴
i am not feeling great	😞

6. Conclusion

Sentiment analysis is an active field of machine learning research in which the goal is to find the opinion or emotion expressed by a text. It can be viewed as two tasks – how to represent the words in the text, and how to classify sentiments to full sentences using these word representations.

There are mainly two methods for the first task: encoding (count vectorization and TF-IDF vectorization) and embedding (GloVe). Encoding would create a high dimension feature space with each word being a feature, and result in a large sparse matrix. We can reduce the dimension by including only the top N words, where N is latent and needs to be fine-tuned. One potential problem is that the top N words may not include the most useful information and thus we throw away the possible useful information in the rest of words. We can also use *SelectKBest* from *sklearn,* and use f*_classif* to help us pick k best features (words). Word embedding is better in regard of dimension, but one potential problem is that it may not fit in to our problem very well since we keep it fixed. One solution is to re-train the word embedding together with the weights of the network (simply set trainable=True). The training may take a long time thus a GPU may be necessary to speed up the training process.

For the second task, there are many methods available such as Naïve Bayes classifier, DNN, and RNN. RNN model is quite versatile for sequence input, and the advance version LSTM is capable of modeling the long-term dependency in the input sequence. Another alternative to LSTM is the gated recurrent unit, with fewer gates than LSTM, it has potential to reduce both overfitting and training time.