Sentiment Analysis

Tweets sentiment analysis is to determine whether the sentiment of a tweet is positive, negative, or neutral. The method used for tweets sentiment analysis is Sentiment Analysis with Long Short Term Memory Units (LSTMs). This method is from “Perform sentiment analysis with LSTMs, using TensorFlow" (Deshpande, A., 2017). We do this task following O'Reilly tutorial and use his teaching sample code (O'Reilly, 2018).

LSTMs is a deep-learning-based method. We choose deep learning for several reasons: In the past, sentiment analysis requires a lot of domain knowledge, such as linguistic and psychological knowledge, which may need several years’ study. However, in recent years, deep learning has greatly reduced the difficulty of sentiment analysis. Instead of requiring a lot of domain knowledge, deep learning methods use general and understandable mathematical and statistical methods to process text, and then use the processed data to train the model. It greatly reduces the threshold of entry into this field and often performs well.

The data set we use is the Imdb movie review dataset. It has 25,000 labelled movie reviews, half of which have positive labels and half have negative labels. People tend to express emotions in movie reviews, so this dataset should be suitable for sentiment analysis. We divided this data set into 90% training data and 10% testing data.

The task then can be divided into 4 steps:

1) Build the word vector model and create id matrix for training data

2) Build RNN (with LSTMs)

3) Train the model

4) Test and Evaluation

1) Build the word vector model

The input for neural network can not be raw string, because some basic operations such as backpropagation or dot products cannot be performed on string. The input should be some scalar numbers or vectors or matrices of scalar numbers. In order to turn the raw text into the input for neural network, we need to create word embeddings, mapping words from the vocabulary to vectors of real numbers. “Word2Vec” model is useful is to this task. It creates word vectors by taking as its input a large corpus of text and produces a vector space. Word vectors are positioned in the vector space such that words that share similar contexts in the corpus are placed in close proximity to one another in the space (Mikolov, T., Chen, K., Corrado, G., & Dean, J., 2013). After process data though Word2Vec model, it will output an embedding matrix, which contains word vectors for every word in the training dataset.

For simplicity, we use GloVe pre-trained word vectors to generate a 400,000\*50 dimensional embedding matrix (Pennington, J., Socher, R., & Manning, C., 2014). Each row of the matrix is a word vector.

For the training set, we remove punctuation, parentheses, question marks, etc., and leaves only alphanumeric characters for each sentence. Then we use Tensorflow’s embedding lookup function to generate the vector representation for each sentence in the training set and create an id matrix containing these vectors as training input.

2) Build RNN (with LSTMs)

Recurrent Neural Networks (RNNs)

The temporal information of the text is important when it comes to natural language processing, because each word in a text is very dependent on its context. In order to extract and use context information, we use RNNs instead of traditional feedforward neural network.

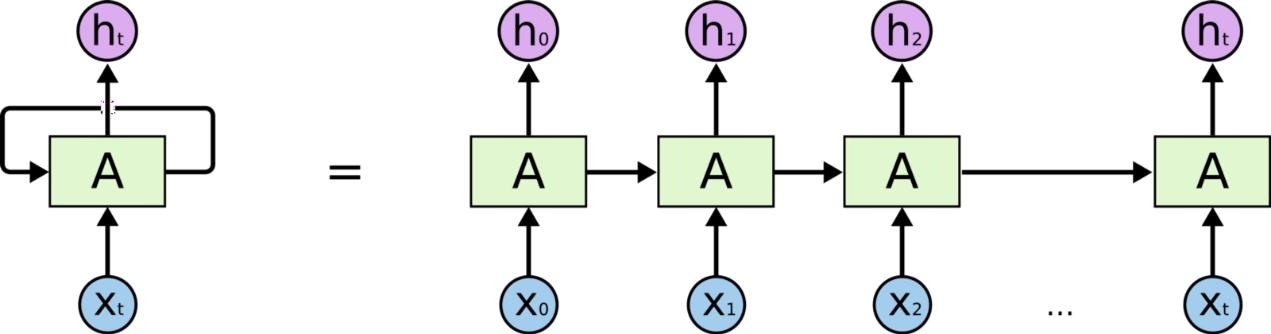
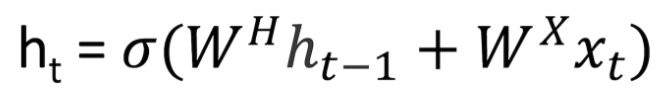


Figure-1. Sequential processing in RNN, from: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Figure-1 is the sequential processing in RNN. represents for input word. Each is related to a time step t and each time step t is also corresponds to a hidden state . The hidden state contains the information from previous time steps. Hidden state is calculated by the following equation:



In the above equation, is the activation function. and represents the weight matrices. For all time steps, is the same, while varies for each input, so that the hidden state is affected for both current input and previous hidden state. These matrices are updated through backpropagation as time goes. Finally, a binary softmax classifier is used for the final hidden state and output values between 0 and 1, which represents the probabilities of positive and negative sentiment.

Long Short Term Memory Units (LSTMs)

There is a problem in the traditional RNNs: when the gap between the relevant information become very large, RNNs are unable to learn to connect the information. In other words, the traditional RNNs performs bad in the long-term dependencies. To solve this problem, we add a long short term memory units into the previous RNNs.

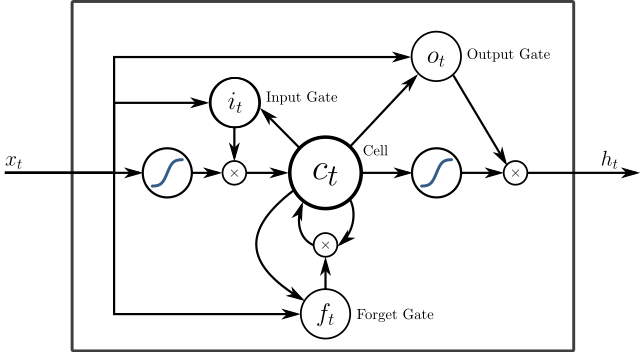


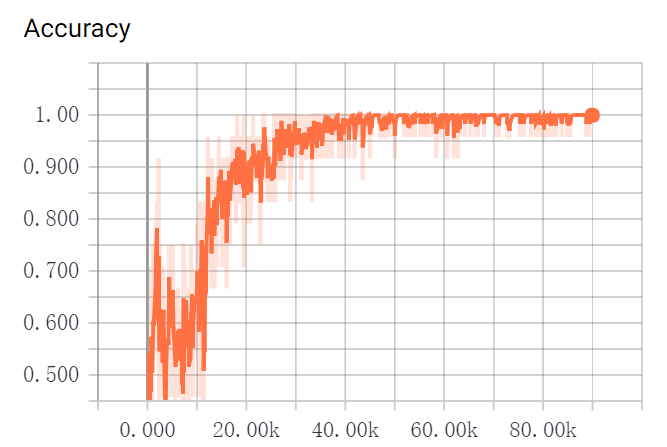
Figure-2. A peephole LSTM unit with input (i.e. i), output (i.e. o), and forget (i.e. f) gates, from <https://www.wikiwand.com/en/Long_short-term_memory>

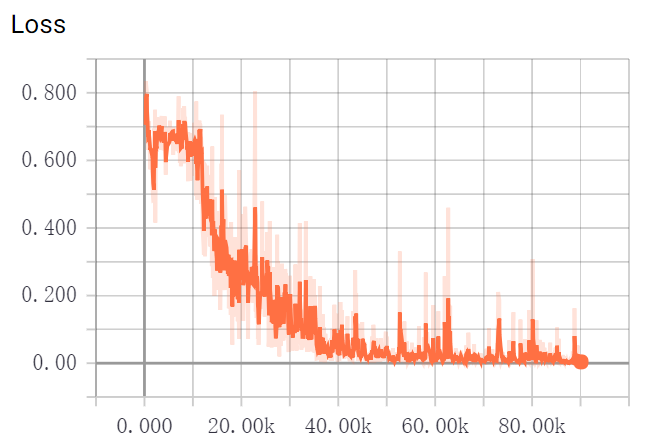
As shown in Figure-2, instead of using a simple function discussed above to calculate hidden state vector , LSTM units use a more complex function to calculate . It introduces an input gate to decide how much should the model care about each input, a forget gate to throw away some information the model don’t needed, and an output gate to get input from the intermediate state and output the final hidden state.

We firstly construct a LSTM cell with 64 units using tensorflow’s nn.rnn\_cell.BasicLSTMCell, then use a dropout wrapper to the LSTM cell to prevent overfitting. After that, we put both input data and the LSTM cell into the dynamic RNN then go through a dense layer to get the final output. The output contains 2 classes, positive or negative. We use standard cross entropy loss with a softmax layer for the final prediction then use Adam optimizer to update the neural network.

3) Train the model

We use Tensorboard to monitor the loss and accuracy. The following charts show the change of accuracy and loss over time. The model is run for 90,000 iterations and finally converged. However, there is possibility that the model overfits the training data.





4) Test and Evaluation

After training the model, we first test the performance of this model on the testing data. The accuracy is 87.5%. The result seems general acceptable. However, when it is applied to the twitter data, compared to other sentiment analysis APIs such as textbolb and vaderSentiment, our model performs not as well as expected. The following examples reflects performance and some problems in the LSTMs model. For the first two examples, our model predicts sentiment as expected. However, as shown in the third example, it cannot tell neutral sentiment because itself is a binary classifier. In addition, as shown in the fourth and fifth examples, LSTMs model cannot tell the sentiment of the emoji in tweets. The reason is that when we process the training data, we only leave the tokens of words or numbers. In this case, the model hadn’t been trained with any emoji data, so it cannot classify correctly the tweets containing a large number of emojis but only a small number of words.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Tweet | LSTMs | textblob | vaderSentiment |
| 1 | “Happy birthday! @Josh” | Positive | 1.000 | 0.6114 |
| 2 | “Beautiful Friday #smile” | Positive | 0.575 | 0.5994 |
| 3 | “It's just 10 days 'til #AllStarLanes #ShepherdsBush opens.” | Negative | 0.000 | 0.0000 |
| 4 | “😊😊😊 #Saturday” | Negative | 0.000 | 0.5614 |
| 5 | “Photoshop? 🙄😒” | Negative | 0.000 | -0.5423 |

In order to ensure the accuracy of the sentiment analysis part of this task, vaderSentiment model is finally chosen. The reason for not choosing textbolb is that it does not perform well for emoji as well. As shown in the fourth and fifth examples, textbolb incorrectly classify the tweets to neutral sentiment. These tweets are special, in which words has neutral sentiment but there are still some emoji reflecting strong sentiments. This indicates that textbolb cannot deal with emoji well.

Future work

In the future, there are some possible improvement in the LSTMs motel to make it more practical:

1. Change the binary classifier to classifier with 3 outputs, respectively positive, negative and neutral.

2. Use larger data set. Google created 3 million word vectors. Each word vector has a dimensionality of 300. This larger word vector can produce a more general model.

3. Select emojis as features of the network to make the classifier sensitive to emoji information.

Reference

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

Deshpande, A. (2017). Perform sentiment analysis with LSTMs, using TensorFlow.

<https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow>