Paper 2 | Twitter Sentiment Analysis using combined LSTM-CNN Models by Pedro M. Sosa – published by 7th June 2017

Abstract

1. Propose 2 neural network models: CNN-LSTM and LSTM-CNN to do sentiment analysis on Twitter data
2. Perform comparisons of the 2 models above against regular CNN, LSTM, and Feed-Forward networks
3. Results show that both models achieve better results, with the best model being LSTM-CNN, achieving 2.7% - 8.5% better than regular models

Motivation

1. Past decade has seen the exponential growth of social media services such as Facebook, Twitter, LinkedIn, and many others. Companies and organisations have realised that these platforms can be an invaluable source on information to interact and learn about their customers
2. Deep learning and other ML techniques can provide an automated method to process and extract sentiments from these large amounts of information
3. This paper is to use NN to classify any given tweet into a "positive" or "negative" one. This could be very useful to gauge the overall sentiment of tweets targeting a specific product, company, organisation, or other entity

Models

a) Background

- **The intuition behind using CNNs** on text relies on the fact that text is structured and organised. Therefore, we can expect a CNN model to discover and learn patterns that would otherwise be lost in a feed-forward network. For example, it might be able to distinguish "down" in the context of "down to earth" to be of positive sentiment. It can also extract these features regardless of where they occur in the sentence
- **Introduction to Long-Short Term Memory (LSTM)**
    - Designed to "remember" previously read values for any given period of time. Usually contain three gates that control the flow to and from their memories
    - **Input gate:** controls the input of new information to the memory
    - **Forget gate:** controls how long certain values are held in memory
    - **Output gate:** controls how much the value stored in memory affects the output activation of the block
    - The LSTM model might be able to handle sentences with changing sentiment

b) Proposed 2 models

- **CNN-LSTM model**
  - The intuition behind this model is that the convolution layer will extract local features and the LSTM layer will be able to use the ordering of said features to learn about input's text ordering

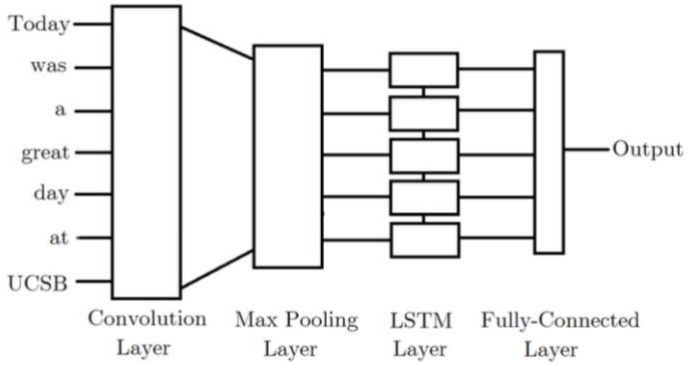

Figure 3: CNN-LSTM Model

- **LSTM-CNN model**
  - The intuition is that its output tokens will store information not only of the initial token, but also any previous tokens. Therefore, the LSTM layer is generating a new encoding for the original input. Then the convolution layer is expected to extract local features

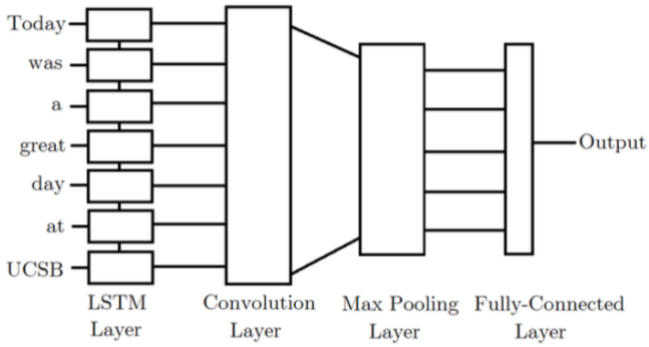

Figure 4: LSTM-CNN Model

Results

1. **Model comparison**

Neural Network Model	Avg. Accuracy
Feed-Forward (Word Embeddings) [1]	58.4%
Feed-Forward (Feature Vectors) [1]	66.8%
CNN	66.7%
LSTM	72.5%
CNN-LSTM	69.7%
LSTM-CNN	75.2%

Table 2: Average accuracy of different neural network models.

- The ordering of the layers in the models play a crucial role on how well they perform

- The CNN-LSTM model is underperforming probably because the initial convolutional layer is losing some of the text's order/sequence information and if the CNN layer does not really give us any information, the LSTM layer will just be a fully connected layer. Performed worse than a regular LSTM
- The LSTM-CNN model is the best because its initial LSTM layer seems to act as an encoder such that for every token in the input there is an output token that contains info not only of the original token, but all other previous tokens

2. **Further results**
   - Learning rates
     - Optimal number of epochs to train our models was 10. However, we realised each model had different "learning rates"

Epochs	CNN	LSTM	CNN-LSTM	LSTM-CNN
3	56.5%	62.1%	58.1%	64.5%
10	66.7%	72.5%	69.7%	75.2%
20	70.7%	51.9%	68.2%	70.1%

     - LSTM and LSTM-CNN models achieved 60%+ accuracy with just a few epochs but too many epochs resulted in overfitting
     - The CNN and CNN-LSTM behaved the opposite way, where too many epochs resulted in less overfitting
   - Effects of dropout
     - For both models, the paper tested the effects of dropout by adding a dropout layer after the CNN layer. The results show that dropout was extremely important for the models to work more effectively

Dropout Prob.	CNN-LSTM	LSTM-CNN
20%	58.4%	59.1%
50%	69.7%	75.2%
98%	51.7%	48.1%

   - Using pre-trained word embeddings
     - Aside from using embedding layer to learn the word embeddings during training, the paper also experimented using GloVe pre-trained word embeddings
     - However, using GloVe resulted in worse accuracy. This might be due to the textual irregularities that tweets present. It is not unlikely to find misspellings, unconventional word abbreviations, internet-related slang and other tokens that are unknow to the GloVe word embeddings

Embeddings	CNN	LSTM	CNN-LSTM	LSTM-CNN
Pre-trained (GloVe)	62.0%	64.8%	66.8%	70.3%
Not Pre-trained	66.7%	72.5%	69.7%	75.2%

Conclusion

1. **The CNN-LSTM model achieved 3% better than a regular CNN model but did 3.2% worse than an LSTM model**
2. **LSTM-CNN model does 2.7% better than a regular LSTM model and 8.5% better than a CNN model**

3. This paper conducted an in-depth exploration of the effects of epochs, dropout, and pre-train word embedding in the models

Further work

1. Combining CNN and LSTM seems to be effective and it might be interesting to apply the proposed models to other NLP tasks
2. Test out different types of RNNs. For example, bidirectional LSTMs might yield an even better result as each token fed to the CNN layer would contain information from all the other tokens in the original input

Code is available at: https://github.com/pmsosa/CS291K (Tensorflow)