Sentiment Analysis using a Deep Convolutional Network

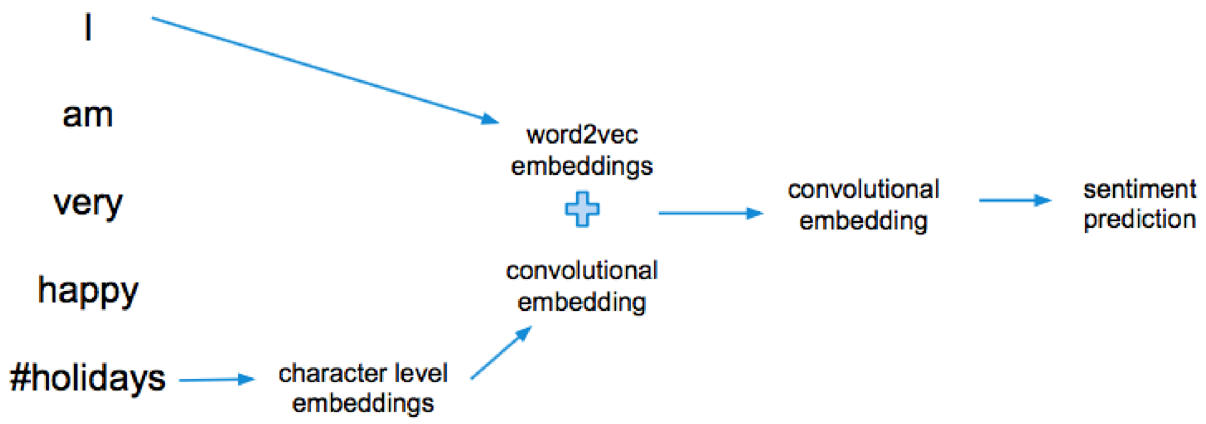
Satwant Rana, 2012MT50618
Surbhi Goel, 2011CS10257

November 23, 2014

Sentiment mining in Twitter is a challenging task due to lack of regular language in tweets. Merely using the bag-of-words approach is often unable to capture the correct sentiment. Specially in figurative language, two parts of the sentence often reflect contrasting emotions and relative position becomes important. Often a hashtag such as *#not* can completely reverse the meaning of the sentence. However, training only on sarcastic data easily overfits as mostly sarcasm is used as a negative emotion. Also, it is more important to be able to handle sarcasm in general data and not sarcasm specific data. Motivated by this, we chose to use a convolutional neural network to capture translational similarity in text, handle variable length sentences and extract structure and semantics at both character and phrase level. We modified the deep convolutional neural network CharSCNN given by researchers at IBM to use in the mixed domain of figurative and direct language. Limited by computational capabilities, we were able to only run a smaller version of the network on 10000 tweets and obtain an accuracy of around 70%. Despite this, we believe that this approach holds potential provided sufficient computational power and time.

1 Architecture

The network consists of a hidden layer, two convolutional layers and a final soft-max layer. The first hidden layer converts each word to a fixed length embedding. This embedding is fed into the first convolutional layer which produces local features around each character of the word and then combines them using a max operation. This helps in capturing information that appears in different parts of the words like in case of hashtags such as *#notreally* and adverbs such as *beautifully* which have information in the part before the suffix. The finall character embedding for a word is concatenated with its *word2vec* embedding trained on a corpus of 1.6 million tweets. This is then inputted to the second convolutional lever which extracts local features around each word and max pools them to give a final sentence embedding. This is then fed to a 2-way softmax layer to give the final sentiment.



2 Convolutional Layers

Both the layers are similar in structure so we will explain only the character level layer. Following the first hidden layer, we have each word represented as the same length d^{chr} vector. The convolutional layer applies a matrix-vector operation to each window size k^{chr} of overlapping successive frames. To accommodate corner cases, we add padding on both sides of length $\frac{k^{chr}-1}{2}$. The convolved output is then reduced to a single vector of size cl_u^0 by using cl_u^0 filters and max pooling across the characters in each filter as shown in the figure below. At the word level, the embedding from *word2vex* of size d^{wrd} is concatenated with the vector obtained above. The layer does a similar operation using cl_u^1 filters of size k^{wrd} and max pooling along each word of the sentence. The learning rate of the network is denoted by λ.

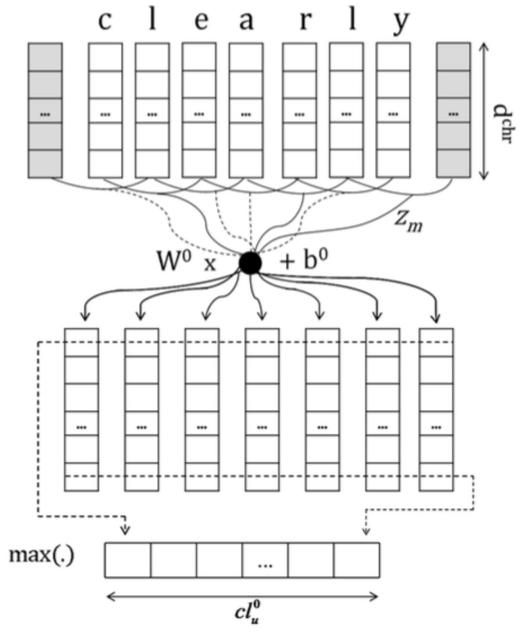


Figure 1: Convolutional approach to character-level feature extraction.

3 Implementation

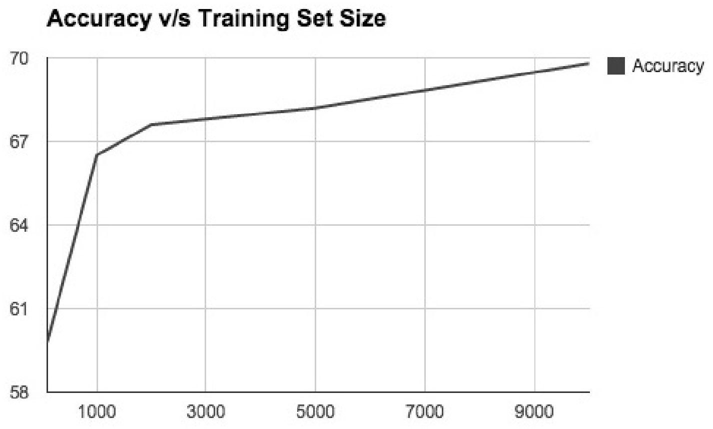
The network is trained by minimizing the negative log likelihood over the training set. This is done by performing stochastic gradient descent on the network. Due to efficient gradient calculation along with easy calculation of other mathematical expressions in *Theano*, a versatile Python library, we implemented the network in *Theano*. The code can be found at the following link: `https://github.com/satwantrana/CharSCNN`.

The hyper parameters used in the network are listed in the following table. Due to computational constraints and small size of training data, the parameters chosen are smaller than we anticipated but they can be tweaked in the code provided to run at any value.

Parameter	**Value**
d^{wrd}	30
k^{wrd}	5
d^{chr}	5
k^{chr}	3
cl_u^0	10
cl_u^1	20
λ	0.0001

4 Results

With the above parameters, the system is trained on a mixed corpus with about 75% of the tweets randomly chosen from Stanford Twitter Sentiment (STS) corpus, and 25% of the tweets coming from SemEval 2015 Task 11's training data comprising of figurative language tweets. The performace plot is as follows:



The extrapolated score on the portion of STS in the training data is about 77.1% which is considerably lower than the State of the Art of 86.4% on the complete data, set by the original paper. The obvious reasons are a smaller training dataset and smaller parameters which don't capture the complete dependency of the network. Also the results can't be directly compared with the original paper, as about 25% of our training data is figurative (majorly sarcastic, and some ironical tweets).

Analysis of the test set reveal that about 43.3% of the wrongly classified tweets (13% of the complete test set) are sarcastic, and with a trailing sarcastic modifier (#not, #sarcasm, etc). Sarcastic tweets with non terminal modifier were correctly classified. For instance, "So excited to work until 10 tonight and then come home and study for my finals . #not" is labelled incorrectly with postiive sentiment, whereas "and the party was so much fun #not because it was in her house but she was outside with some people all the time" is labelled correctly with negative sentiment.

A speculated reason for the same could be padded zeroes at the end of sentence which, possibly, reduce the sentiment of the terminal token, and in case of sarcastic tweets with terminal modifer, the terminal token governs the sentiment of the tweet.

5 Conclusions

Although in our experimentations, the network didn't train and perform to its potential, but the approach is promising, and provided sufficient data, computational power and time, it can reach near SOTA performace. Our experimentations with sarcastic data open a new window for sentiment analysis on sarcastic data using neural networks. As discussed earlier, a potential issue with wrong classification of sarcastic tweets with terminal sarcastic modifier, is padding of zeros at end. This can potentially be taken care by larger pooling size in sentence level convolutional network. Also, this opens scope for experimantaions and improvisations on the convolutional network structure, and the whole network in general.

References

[1] *Santos and Gatti* Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts