Sentiment Analysis using neural architectures

Soumith Chintala
New York University
New York, NY 10012
soumith@gmail.com

Abstract

Most sentiment analysis approaches are based on heavy preprocessing of the data which involves carefully choosing the right features based on the nature of the data, intuitive analysis and factors like language. I explore an unconventional approach to sentiment analysis based on recently proposed neural network architectures for NLP that parallel traditional well-performing approaches, both in accuracy and speed, and have the advantage of requiring no manual preprocessing or feature selection.

1. Introduction

Sentiment Analysis is the method of analyzing a piece of data for human emotions. Recently, textual sentiment analysis has garnered a lot of attention from scientific researchers, as well as non-scientific areas such as advertising, trading, marketing and technology domains. This could be attributed to recent breakthroughs in information latency and the social sharing paradigm. With vast amounts of live data available for analysis, conveniently through third-party APIs, this is being seen as an opportunity for extremely cheap and low-latency market analysis[1][4] [5][8]. However, most of these approaches are often engineered to perform well on specific types of data, and when applying the algorithm to new data, new, hand designed features have to be extracted, a task that could be avoided by using well performing unified architectures that learn good feature representations implicitly. A quest for a unified architecture, combined with my prior affinity towards deep learning architectures made it a natural choice to adopt an architecture proposed by Collobert et al.[3] which converts words into probability space using word-indices from a predefined or a learnt dictionary and applies a convolutional network[9] over these indices. This approach has been shown to do well in traditional NLP tasks such as Parts of speech tagging, chunking, named entity recognition and semantic role labeling, which justifies its usage in my system.

2. Architecture

Artificial neural networks have an extremely simple algorithm which can be briefly summed up for classification problems as shown. A criterion for predicting a class could

```
Input: N Training Samples, K modules in network
Output: Predicted Class
foreach sample do
    input[1] = sample
    foreach module i in neural network do
        output[i] = module.forwardPropagate(input[i])
        input[i+1] = output[i]
    end
    predictedClass = criterion(output)
    if training then
        error[1] =
        criterion.loss(predictedClass,groundTruth)
        foreach module [K-i] in neural network do
            output[i] =
            module.backwardPropagate(error[i])
            module.updateWeights()
            input[i+1] = output[i]
        end
    end
end
```

be any function which takes a vector and produces a class number. For example, a criterion could be a max function that takes an input set of probabilities and returns the index of the maximum probability. The network architecture and the modules are discussed next.

2.1. Network

Each input data sample (it could be either a sentence or the whole text in the data sample) is padded with PADDING tags to make the input of a consistent size. The padding tags are then treated as another regular word by the network while it learns its weights. Padding could've been avoided and the network be implemented with a max-over-time module, which chooses the most significant features

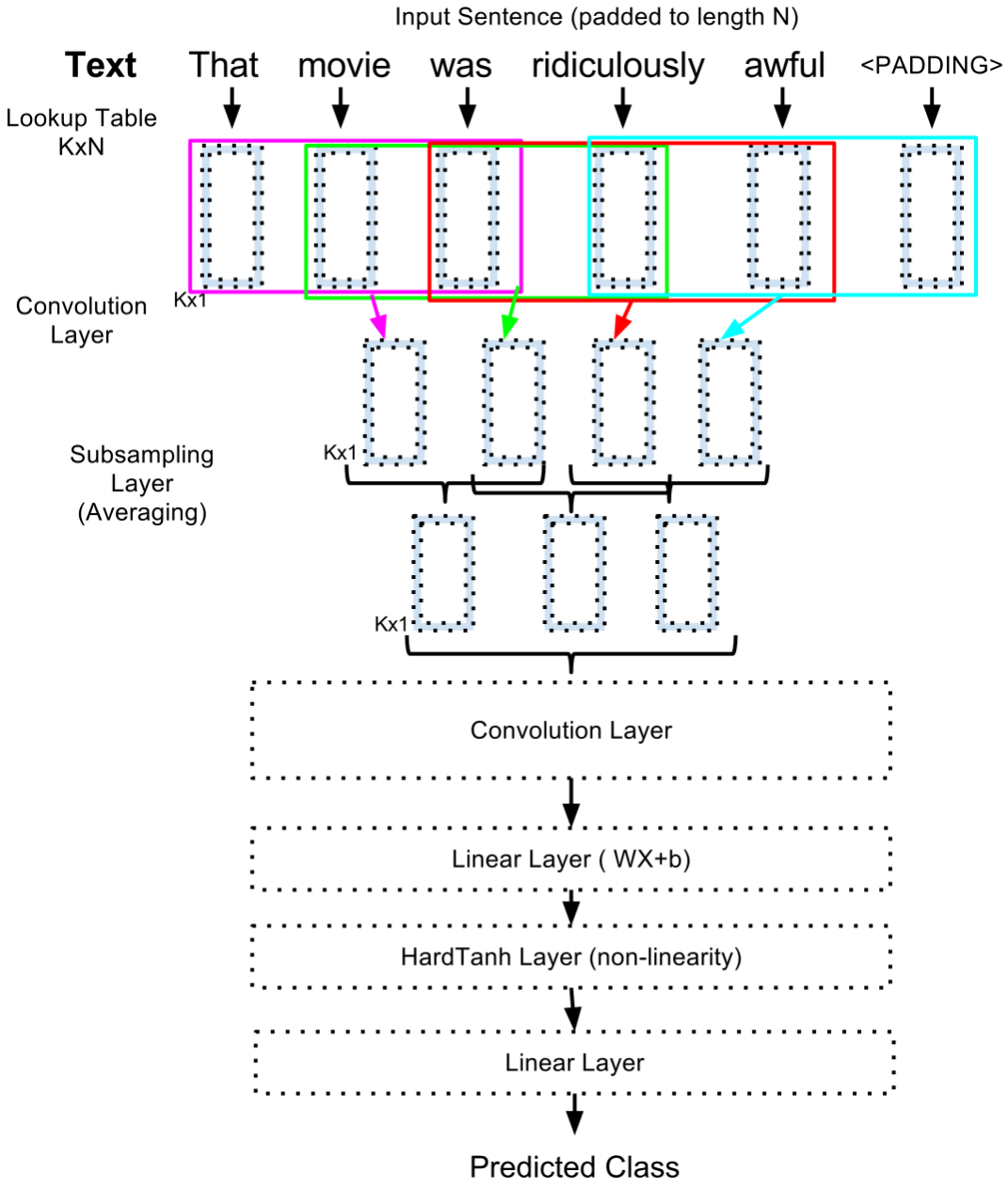

Figure 1. Architecture of the network

from variable-length sentences to get a fixed size feature vector, but this isn't completed at the time of writing this report. The architecture used for the neural network is a slight variant of the convolutional neural network proposed by Collobert et al.[3]. There are 7 layers in the network as seen in Figure 1. Each of the layers are described in the subsections below.

2.1.1 Lookup Table

The lookup table is the key concept to use neural networks for NLP tasks. Initially, for each word in the input sample, its corresponding index is looked up in a given dictionary D, and upon finding the word, it returns the word's index. If the word isn't found in the dictionary, it returns the index of the UNKNOWN tag instead. The dictionary is generally learnt from a large corpus of text in an unsupervised manner, and given a size limitation, the most significant words are kept and the rest are discarded. The lookup table has

a weight matrix of size KxN where N is the length of the input sentence(number of words) and K is the per-word feature vector size. This feature vector is called an embedding for the word and can be initialized randomly, after which, it is learnt in training using back-propagation. However, initializing these embeddings using large corpus of text gives the search a good starting point that would be closer to the global minimum and is often seen to improve performance. Usually, these initialization weights are learnt using unsupervised clustering over large data and over several weeks and are shared by researchers, just like sharing hand-coded features. In my system, I used embeddings from Collobert et. al(2011) which were trained over 4 weeks over Wikipedia. The dictionary size was 130,000 words. For words in my training set that weren't present in their dictionary, I used random initializations.

2.1.2 Convolution Layer

A 1D convolution layer in nlp terms is like learning n-grams over a kernel-size n. This layer models the basic assumption that nearby words have some relative significance. This is not limited to words/sentences and is often seen in higher dimensions in a range of other signals, like images, sound etc. In this layer, given a 1D kernel of size S, stride D and an input matrix NxK from the previous layer, the convolution is applied to each subset matrix of size SxK as shown in figure 1. The subset indices are incremented by the stride D after every convolution. This can be seen as a window sliding over the words, but the window moves by a step-size D in every iteration. I used convolutions of different sizes to see performance differences. Convolutions 5 and above perform equally well. I always used a stride of 1.

2.1.3 Linear Layer

The linear layer is a simple matrix multiplication + an additive bias layer, i.e weights W are multiplied with input matrix M and a bias b is added to each element. The weights and bias are learnt using back-propagation.

2.1.4 HardTanh Layer

HardTanh Layer basically applies the HardTanh function to each element in the input matrix. This introduces a non-linearity which make the architecture different from regular linear classifiers. HardTanh is defined as

$$HardTanh(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases} \quad (1)$$

3. Training

All training was done using Mean Square Error(MSE) criterion as the loss function, using stochastic gradient descent as the optimization method. Learning rate was the only hyperparameter that was tuned manually apart from experimenting with the number of hidden units in each layer. A validation set wasn't used. The datasets were split 70%-30% for training and testing respectively.

3.1. Datasets

The initial goal of the project was to work on atleast three datasets, i.e Movie Review dataset[6], Congressional Speech dataset[10] and Twitter data from various sources. However, due to time constraints, experiments were done on two datasets, namely Movie Review dataset and another dataset by Pang & Lee called the sentence polarity dataset[7], which has short sentences (less than 50 words in length) with a positive/negative polarity to each sentence. All words were converted to lower-case and this was the only preprocessing done. [1]

3.2. Experiments

3.2.1 Movie Review Dataset

The movie review dataset has 2000 reviews, 1000 positive and 1000 negative. The data was split into 1400 training and 600 testing samples. There were two sets experiments that were performed on this dataset. One set involved padding the data to the longest review, which was 2400 words. This was however not expected to be a good choice, and was seen in the experiments, primarily because the average review length was significantly shorter. Adding so much padding made the actual words weaker overall, because of subsampling and was this turned out to be a slight detriment. The second set of experiments that were done was taking the first 500 words of every review(and if the review was shorter, adding a padding upto 500 words). Though the mean review size was slightly more than 500, I was curious to see whether it was enough to determine the polarity of the review using the first 500 words. The only parameters that were tuned(strictly on the training set only) were the learning rate and the number of hidden units per layer.

3.2.2 Sentence Polarity Dataset

The sentence polarity dataset has 5331 positive and 5331 negative sentences, 70% of which were in the training set and the rest in the test set. All sentences were padded to 51 words (which was the maximum sentence size in the dataset). The only parameters that were tuned(strictly on

[1] This could've been avoided, but it was more memory-efficient to implement, and if anything, losing capitalizations is losing a feature that might've been a positive contribution.

Algorithm	Movie Review Data	Sentence Polarity Data
Baseline	86.4%	N/A
ConvNet	76.67%	77.56%

Table 1. Classification accuracy on the test set

the training set only) were the learning rate and the number of hidden units per layer.

4. Implementation

The programming was done in C++ and Lua, using Torch7[2] which is a machine learning library mainly focusing on neural networks and is co-developed at NYU, NEC Labs, Princeton and IDIAP, Switzerland.

5. Results

Table 1 shows the classification accuracy of my algorithm and compares it with the baseline. Unfortunately, there is no baseline for the Sentence Polarity dataset. The primary purpose of this dataset was different and there seems to be no one out there who did a binary classification on this dataset. For the Movie Review Dataset, the result reported in the table uses the first 500 words in each review. If the entire review was used(but padded significantly, to 2400 characters), the classification accuracy dropped to 30%.

6. Conclusion

A couple of ideas remains to be explored. The first is implementing the max-over-time module as described in 2.1. This would most likely improve results because it was observed to be so in [3] . The second idea that remains to be explored is the main advantage of this approach. This approach is supposed to give an advantage when hand-coded features are not trivial to create, i.e. on data that doesn't have a very intuitive structure, like twitter data. This will be explored in the future. Though the results failed to improve upon the baseline, I don't think it is a failure of the algorithm, but rather its a pretty good start upon which I could improve accuracies, and I mainly believe in this because two very important ideas were left unexplored.

References

[1] J. Bollen, H. Mao, and X. Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1 – 8, 2011.

[2] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[3] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

[4] N. A. Diakopoulos and D. A. Shamma. Characterizing debate performance via aggregated twitter sentiment. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1195–1198, New York, NY, USA, 2010. ACM.

[5] A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision. *Processing*, pages 1–6, 2009.

[6] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the ACL*, 2004.

[7] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the ACL*, 2005.

[8] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2:1–135, January 2008.

[9] P. Sermanet and Y. LeCun. Traffic sign recognition with multi-scale convolutional networks. In *Proceedings of International Joint Conference on Neural Networks (IJCNN'11)*, 2011.

[10] M. Thomas, B. Pang, and L. Lee. Get out the vote: Determining support or opposition from Congressional floor-debate transcripts. In *Proceedings of EMNLP*, pages 327–335, 2006.