Sentiment Classification for Movie Reviews using Neural Networks

Daniel Kim
New York University
jhk774@nyu.edu

Abstract

This paper reports on the building process of a deep learning based sentiment analysis system that was used to test out results on Kaggle's movie review sentiment classification. Specifically, my main goal was to achieve the most accurate sentiment classification result possible by applying the following strategies: 1) Implementing the Convolutional Neural Network (CNN) based model which is known to capture wide ranges of context and the Long Short-Term Mermoy Network (LSTM) based model which is known to capture sequential patterns within a sentence. 2) Exploiting the idea of transfer learning (e.g., using pretrained word embeddings or dataset from other sentiments).

1 Introduction

Sentiment analysis is the process of letting us understand personal opinions on various types of content (e.g. place, object, person, etc). The study of sentiment analysis exemplifies the significance of the interconnection between a sentiment written by a customer (opinion) and the business that offers the content. For example, sentiment analysis plays a crucial role for platforms such as online shopping malls or movie review websites. In this paper we are focusing on a movie review web engine, specifically Rotten Tomatoes. By implementing a specific algorithm to test out the vast range of sentiments, in this case movie reviews, we are able to determine correct sentiment analysis. There are many machine learning techniques to classify movie review sentiments including Naive Bayes, Logistic Regression and Support Vector Machines. However, these methods have rather shown low performance when tested (Pang et al., 2002). So what is it that makes achieving high performance so difficult? Performance could become detrimental when encountering a complex type of sentiment. For example, the following sentence *"How could anyone sit through this movie?"* barely contains words that have negative connotation, yet is considered a negative review. Such complex scenarios often occur in a majority of modern day reviews. Therefore, an accurate sentiment analysis requires a machine learning model that can deeply understand the overall semantics of a sentence. Currently the process of extracting sentiments from a certain sentence remains as a challenging operation. Not only does it require learning all potential definitions of each word but also the interaction of these words that implicitly builds up the overall definition of the sentiment.

Fortunately, several pieces of work have shown promising results in this specific task. Among the various deep neural network models, I chose the CNN based model called Text CNN proposed by (Kim, 2014) and the LSTM (Hochreiter and Schmidhuber, 1997) based many-to-one classification model. The Text CNN is well known for capturing the semantic of each word with its surrounding context using various types of convolution filter. The underlying idea of Text CNN is that the latent semantic of a certain word is likely to be related to its adjacent words. Therefore, Text CNN convolutes consecutive words to extract their high-level semantic. Another deep learning model is the Recurrent Neural Network (RNN). In this paper, I am mostly focused on Long-Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) that is the most representative variant of RNN. LSTM is widely known to be effective on datasets that have sequential patterns (e.g., natural language). To capture the pattern, LSTM extracts the latent semantic of the current word using the latent semantic of the previous word. To summarize, these two deep learning based methods heavily analyze on closely placed words and flow of sentence to deeply understand the semantic of a sentence.

However, because this model is often trained on a large dataset, we also applied transfer learning which transfers the general knowledge obtained from the dataset into a target task. In the Natural Language Processing (NLP) field, there have been many attempts to exploit this idea, specifically to encode general language semantics obtained from large datasets into word vectors (Mikolov et al., 2013; Pennington et al., 2014). This implementation is shown to be effective in learning the deep learning model.

We are now able to finalize our task approach to sentiment analysis. To achieve our objective of enhancing the overall accuracy of movie review sentiment classification, these will be the methods implemented: 1) The Text CNN architecture for learning complex relationships between words within a sentence and the Recurrent Neural Network (RNN), specifically the LSTM architecture for learning the semantic flow within a sentence. 2) Exploiting the idea of transfer learning (e.g., pre-trained word embeddings or including a different corpus).

2 Notations

In this section, we discuss about notations. Suppose we have a training dataset D_{train} that has N number of sentences denoted by x^i and label y^i that has C types of sentiments. The model for classification is denoted by $f(x, \theta)$ where θ represents model parameters. We denote the predicting probability for class j given x^i as $f_j(x^i, \theta)$. Our goal is to learn θ by using D_{train} to make predictions on D_{test}.

3 Deep Learning Approach

This descriptive section expands our knowledge on the various deep learning approaches that have been used for our analysis.

3.1 Text CNN

We first describe the Text CNN model and explain the steps for sentiment classification using this model. The overall architecture of Text CNN is represented in (Fig.1). The first layer is the *Word Embedding Layer* which embeds the word semantic into a low-dimensional vector and creates a representation of a sentence $x_{1:t} \in R^{t \times d}$ by selecting word embeddings which is represented on the leftmost grid in Fig.1. The second and most important layer of our model is the *Convolution layer* which creates high-level features by convoluting adjacent words in a specific sentence. This allows us to capture a wide range of context that consists consecutive words. In order to enable this concept, we have different sizes of the convolution kernel which is represented as the overlapped grid in Fig.1. The next step is the *Max Pooling layer* which selects the maximum elements from the outputs convoluted by each convolution filter. These selected maximum elements are concatenated and passed onto the fully-connected layer. The last step is calculating the probability of the sentiment using the *Softmax layer*. The cost function for classification (i.e., Cross-Entropy over Softmax) is

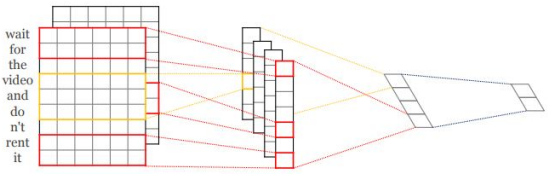

Figure 1: The CNN architecture proposed by (Kim, 2014). The overall architecture is a sequence of layers (Word Embedding layer → Convolution layer → Max Pooling layer → Concatenation Operation → Fully-connected layer → Softmax)

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} log(\hat{p}_{ij}),$$
$$\hat{p}_{ij} = \frac{exp(f_j(x_i, \theta))}{\sum_{j=1}^{C} exp(f_{j'}(x_i, \theta))} \quad (1)$$

where $\hat{p}_{ij}$ represents the probability of x^i to class j, and y_{ij} is 1 if the class of x^i is j. Otherwise it is 0.

3.2 Bidirectional RNN

The second method is the many-to-one Recurrent Neural Network (RNN). Given an input sequence x = (x_1,...,x_t), a standard RNN computes the cell state sequence c = (c_1,...,c_t) and hidden sequence h = (h_1,...,h_t). The Long Short Term Memory Network (LSTM) is a kind of RNN which also generates the cell state sequence c = (c_1,...,c_t) and hidden sequence h = (h_1,...,h_t). LSTM is specifically designed to preserve the longer context (i.e., the information from previous steps) than RNN by preventing the gradient vanishing problem. I therefore adopted the LSTM rather than using the RNN model. The LSTM layer computes h_t as

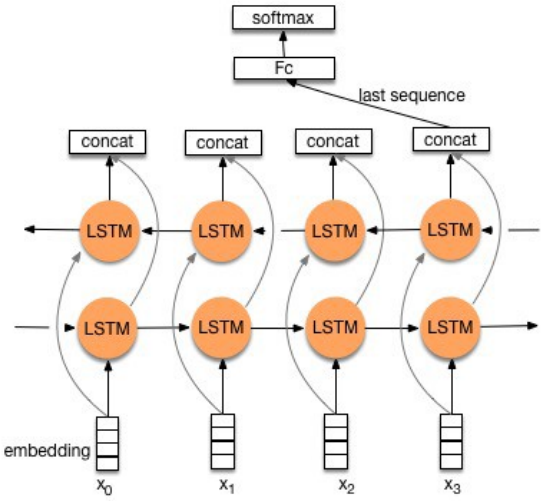

Figure 2: The many-to-one Bi-directional LSTM-based classification model. The overall architecture is a sequence of these layers (Word Embedding layer → LSTM layer → Concatenation Operation → Fully-Connected layer → Softmax)

written below

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ct}c_{t-1} + b_i), \\
g_t &= \sigma(W_{xg}x_t + W_{hg}h_{t-1} + W_{cg}c_{t-1} + b_g), \\
c_t &= g_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \\
h_t &= o_t tanh(c_t)
\end{aligned}
\tag{2}
$$

where σ is the sigmoid function, i is the input, g is forget, o is the output gate function, c is the cell state, and h is the hidden vector. Note that these vectors all have the same dimension.

One limitation of conventional RNNs including LSTM is that they are only able to make use of previous context. To overcome this limitation, we adopt the bi-LSTM which exploits bi-directional context. The idea is to concatenate hidden vectors from two LSTM layers (Fig. 2). Specifically, we implement the many-to-one architecture in order to classify our sentiment which is described in (Fig. 2).

3.3 Joint Model

In this section, We make the joint model incorporating two models (i.e., Text CNN and LSTM). We first describe the motivation for incorporating two models and then the joint model architecture. I took advantage of the different approaches of Text CNN and LSTM. The Text CNN model can capture various ranges of window context using the varying size of the convolution filter while the LSTM model can capture the sequential patterns within the sentence by taking previous context as an input. Therefore, both the LSTM-based model and the Text CNN model are able to differently capture the semantic of a sentence. The intuitive way to capture both patterns is putting the bi-LSTM layer before the convolution layer. Thus LSTM generates the input (i.e., h_t) of Text CNN. One thing to note is that unlike the many-to-one LSTM based model, we utilize all hidden vectors of the LSTM layer. The following steps after the LSTM-layer are the same with Text CNN.

3.4 Pre-trained Word Embeddings

We mentioned earlier that learning semantics of each word is a cost heavy process which requires a vast amount of resources (e.g., training set with a large corpus and training time). Therefore, some researchers provide pre-trained word vectors. In previous works (Mikolov et al., 2013; Pennington et al., 2014), these pre-trained vectors have proved to give a good starting point for the neural network model and provide global semantics of words that cannot be learned from the limited task-specific corpus. We utilize the pre-trained word vectors from a large-scale corpus such as Wikipedia in order to improve our sentiment analysis accuracy. In general, these word embeddings are trained on corpora that do not contain sentiments. Therefore, we finely tune word embeddings in our task. These pre-trained word embeddings come from the following source[1].

4 Experiment

Now that the methods are set place, I have executed experiments to validate my approaches and also conducted error analysis. Finally, the actual results from Kaggle are reported.

Setup The dataset is composed of tab separated files with phrases from the Rotten Tomatoes dataset. Kaggle offers both training and test corpora. The Training corpus contains phrases and their sentiment labels that are distributed into five categories: (0 - Negative, 1 - Somewhat Negative, 2 - Neutral, 3 - Somewhat Positive, 4 - Positive). The test corpus only contains phrases. In order to tune different models with their hyperparameters, I split the original training dataset into two separate sets: a training (80%) set and a validation

[1] http://vectors.nlpl.eu/repository/

Data	Class 0	Class 1	Class 2	Class 3	Class 4
Training Set	5673	21808	63675	26277	7415
Valid Set	1399	5465	15907	6650	1791

Table 1: Dataset Sentiment Statistics

(20%) set. Next, I tuned my model on the validation set and made a submission file for Kaggle using only the training set (80%). This training set is compared with using the whole training set which becomes our test set (100%) because of the limited amount of computing resources. For the evaluation metric for classification performance, I simply used accuracy which is calculated as the following:

$$Accuracy = \frac{\text{The number of true predictions}}{\text{The number of total predictions}}$$

Baselines: Here are the written baselines for each hypothesized model.

- **LSTM**: This is the standard many-to-one LSTM model. The first layer is the Word Embedding layer. The second layer is the LSTM layer. I only focused on the output of the LSTM layer in the last step. It is then passed onto the fully connected layer to classify the sentiments.

- **Bi-LSTM**: This is a variant of the LSTM. This model additionally takes into consideration the flow of a sentence in reverse. Similar to the LSTM model, two output vectors from each LSTM layer are concatenated and then it is passed onto the fully connected layer to classify the sentiments.

- **TextCNN**: This model is originally proposed by (Kim, 2014). We set the window size of convolution filters as [2,3,10,20] and the number of each filter to 64.

- **TextCNN + Glove**: This model architecture uses TextCNN and also pre-trained word vectors (Pennington et al., 2014). These vectors will help solidify the general knowledge of the sentence.

- **TextCNN + Aux**: This model architecture also uses TextCNN but has another fully connected layer to classify the sentiment of the auxiliary dataset (i.e., IMDB[2]'s binary sentiment dataset).

- **Bi-LSTM + TextCNN**: This model is the combination of the Bi-LSTM model and the TextCNN model. This convolutes hidden vectors for each step of the Bi-LSTM model.

Dataset	Valid Set	Test Set
Neutral Prediction	0.509	0.51
LSTM	0.660	0.612
Bi-LSTM	0.661	0.627
Text CNN	0.671	0.634
Text CNN +Glove	0.673	0.634
Text CNN +aux	**0.674**	**0.635**
Text CNN +Bi-LSTM	**0.674**	**0.635**

Table 2: Sentiment Classification Accuracy

4.1 Performance Analysis

Table 2 shows our final results. Here are the observations based on the table:

1) All neural architecture models are able to achieve a higher accuracy result than the benchmark score (0.51) of Kaggle's sample submission dataset.

2) The Bi-LSTM model shows slightly better results than the LSTM model because Bi-LSTM takes into consideration both flow and reverse flow of a sentence to make context representations.

3) TextCNN predictions are more accurate than RNN models. We analyze our results this way due to the pattern of the dataset. We note that there can be a sequential pattern but this exists on a higher-level than a word-level. For example, the sentence *"If a horror movie 's primary goal is to frighten and disturb, then this works spectacularly well ... A shiver-inducing, nerve-rattling ride"* is classified as 4 (Somewhat Positive). However, if we extract this chunk *"If a horror movie 's primary goal is to frighten and disturb"* from the following sentence and test it, it is classified as 2 (Somewhat Negative). This proves that the TextCNN model is able to capture a higher level of sentence patterns than word-level sequential patterns.

4) Including the pre-trained vectors (Glove) improves the TextCNN model in the validation set, yet this improvement is very limited. This is because there are many corpus-specific words which

[2]https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

limits the model to only use a small portion of pre-trained vectors.

5) Including the auxiliary sentiment dataset also improves the TextCNN model in both validation and test set. However, this approach also shows a limited amount of improvement. This is because the auxiliary dataset has binary sentiment labels but our target task requires predicting 5 levels of sentiments.

6) Incorporating the Bi-LSTM model and the Text CNN model shows the most accurate result without an additional dataset. This captures a wide range of context with different kernel sizes and also the word-level sequential patterns with the LSTM layer. However, the limited improvement of this approach implies that we must also learn about capturing varying-level window context along with the word-level sequential pattern.

4.2 Error Analysis and Future Work

This section describes the error analysis conducted on our validation dataset and the guidelines for better improvement in future work.

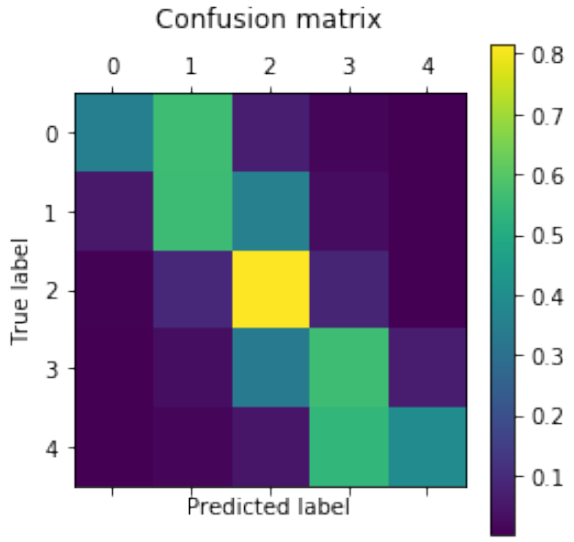

Figure 3: The Confusion Matrix

As shown in Figure 3, this model performs well on sentences that have a neutral connotation (neither positive nor negative). On the other hand, the model seems to have trouble predicting positive and negative labels. I analyzed this result due to the fact that there are quite a few number of examples that are labeled Positive (4) and Negative (0). Therefore we must take into consideration how to effectively capture both positive and negative examples. For future work, we can consider either the following: 1) Using the adaptive sampling technique which differently samples the training examples depending on mis-classification ratio for each sentiment type. 2) Designing a weighting strategy for each class which penalizes more on positive and negative examples rather than neutral examples. 3) Design a loss function that can clearly distinguish the five classifications of the sentiment (e.g., somewhat negative vs. negative). This idea comes from the model in Figure 3 not being able to clearly distinguish the level of sentiment classification.

5 Conclusion

Based upon the following implementations of various models upon a wide range of sentiments, we can conclude that sentiment analysis is a task heavy process. The larger the test corpora becomes, the higher the possibility of encountering semantic variations. Such variations are predictable yet coming up with an accurate semantic model remains a challenge. To overcome these challenges I had to compare and contrast various methods that could be used. My test results with the application of adjacent words, bidirectional sentence flow and pre-trained word vectors showed constant progression in the accuracy score. I have also concluded that the infusion of TextCNN with other techniques such as Glove vectors and auxiliary datasets demonstrate the best performance. It is without question that a simple CNN model is far away from obtaining a high accuracy score compared to models that imbue different methods together.

My final accuracy score on Kaggle came out to as 0.64. A handful of accuracy scores from Kaggle's sentiment analysis on movie reviews were similar to my test results. Out of a total of 861 public scores, I ended up in between the 120th and 170th place range. Yoon's TextCNN architecture on the public leaderboard measured a score of 0.68 which is similar to my highest score on the whole test set using TextCNN and also Bidirectional LSTM. This parallel correlation in both of these accuracy scores supports my overall study of sentiment analysis and proves that implementing the Text CNN with the Bi-LSTM method can enhance the accuracy score in this field.

References

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA. Curran Associates Inc.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.