Sentiment Classification with Deep Learning

Vishwashrisairam Venkadathiriappasamy
The University of Texas at Dallas
Richardson, Texas-75080, USA
vxv180043@utdallas.edu

Abstract

Sentiment analysis or opinion mining refers to the task of classifying texts based on their affect i.e. whether the expressed opinion is positive, negative or neutral. Opinions are central to most human activities and it plays a key role in influencing our behavior and often people make/change a decision based on based on the opinion of others .This project aims to implement and analyze a deep learning model that performs a simple binary classification task of determining whether the expressed opinion is positive or negative.

1. Introduction

Sentiment analysis is one of the most active research areas in Natural Language Processing. Its popularity is mainly because it has a wide range of applications. Sentiment analysis is extremely useful in social media monitoring as it allows us to gain an overview of the wider public opinion behind certain topics. In the recent years with the advent of social media we have witnessed that certain opinionated posts have helped reshape business and influenced public sentiment and emotions.

The applications of sentiment analysis are broad and powerful. The ability to extract insights from social data is a practice that is being widely adopted by organizations across the world. Shifts in sentiment on social media have been shown to correlate with shifts in the stock market. It can also be an essential part of your market research and customer service approach. Not only can you see what people think of your own products or services, you can see what they think about your competitors too. [1]. Even before people buy any products from online retails sites like Amazon, Walmart they have access to the user comments and reviews which helps them in deciding whether to go with that product or not.

Some other advanced opinion mining tasks classify texts on the basis of mood (e.g. happy, angry, sad, etc.), expression (e.g. sarcastic, didactic, philosophical, etc.) or intention (e.g. question, complaint, request, etc.) and Opinion Spam Detection. The next section will describe the related research works on sentiment analysis/opinion mining.

2. Related Work

As discussed in the previous section, sentiment analysis is one of the most active research areas in Natural Language Processing. Its not only because it has a wide application areas but also there was not much research carried out in the past.

The outbreak of modern sentiment analysis happened only in mid-2000's, and it focused on the product reviews available on the Web. Since then, the use of sentiment analysis has reached numerous other areas such as the prediction of financial markets and reactions to terrorist attacks. Additionally, research overlapping sentiment analysis and natural language

processing has addressed many problems that contribute to the applicability of sentiment analysis such as irony detection and multi-lingual support. Furthermore, with respect to emotions, efforts are advancing from simple polarity detection to more complex nuances of emotions and differentiating negative emotions such as anger and grief. [3] presents a very good history of sentiment analysis, evaluates the impact of sentiment analysis and its trends through a citation and bibliometric studies, and produced a comprehensive taxonomy of the research topics of sentiment analysis with text mining and qualitative coding. Most of the recent research had more focus on social media platforms like Facebook, Twitter, etc. Some other popular topics included mobile devices, stock market and human emotions. On the contrary paper published relatively earlier focused on produce reviews, product features and analysis of political situations. This paper also shows the commonly techniques used for sentiment analysis tasks. Some of the more commonly used techniques were knn, Bayesian, regression based algorithms, ngrams, stopword subjectivity, negation, wordnet, etc.

Machine learning techniques are quite commonly used in sentiment analysis where we train a classifier to correctly identify the opinion as positive, negative or any other class. These classifiers are generally of two types: Generation classifiers, which when given an observation return a class most likely to have generated the observation e.g Naïve Bayes classifer, and Discriminative classifiers, which learn the features from input that are most useful to discriminate between possible cases. Logistic Regression and Neural Network based classifiers are examples of discriminative classifies. In this project we'll focus mainly on neural networks and other deep learning based classifiers as they have advantages like they don't need smoothing and have much higher predictive accuracy compared to other discriminative classifiers. Deep learning is the application of neural networks to learning tasks using networks of multiple layers. It can exploit much more learning power of neural networks, which once were deemed to be practical only with one or two layers and a small amount of data. In a nutshell deep learning uses a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. The lower layers close to the data input learn simple features, while higher layers learn more complex features derived from lower layer features. The next section will describe the tasks carried out related to this project.

3. Implementation

The goal of this project is to implement a deep learning model for sentimental analysis. It is a simple binary classification problem that detects whether the sentiment associated with the sentence is positive or negative. Figure 1 gives few examples of the given problem statement. For this project we would be using IMDB Movie Review dataset which that contains a collection of 2000 reviews of different movies collected by IMDB. The overall distribution of dataset is even i.e. it contains 1000 positive and 1000 negative reviews.

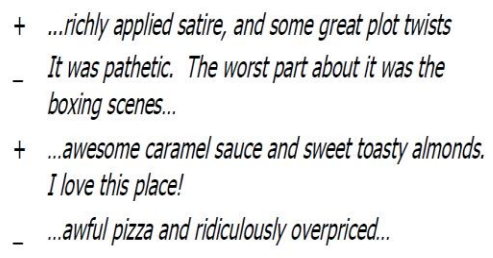

Figure 1 Example of the sentiment classification task for movie review

Figure 2 gives a high level flow chart of the steps involved in the implementation. As seen from the figure, we can divide the process into five different steps: Loading the Data, Building the embedding dictionary, Creating the TensorDataset and DataLoaders, Defining the deeplearning models and finally training/testing the models.

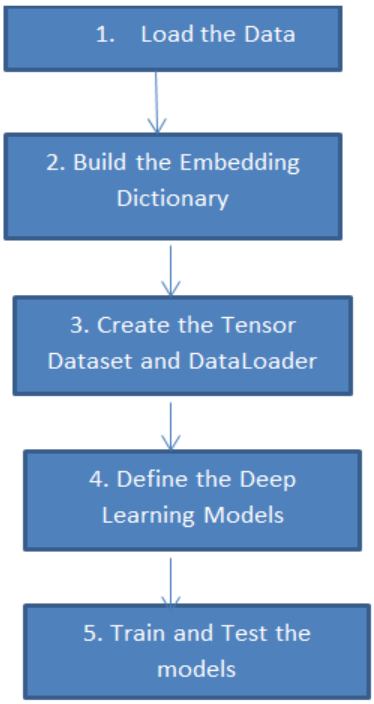

Figure 2 Flow chart of the tasks involved in the implementation

3.1 Loading the Data

The first step involves loading the data as well as the labels from the dataset and creating a vocabulary that contains the integer encoding for a particular word. This step will generate two lists, one containing all the reviews and the other containing all the labels corresponding to the movie reviews and encode it to a particular integer value, and a vocabulary dictionary that contains the integer encoding for that particular word. Consider the example below to get a better understanding.

Before encoding:

DATA = [“This movie sucks", “Good movie"]
LABELS = ["NEGATIVE", "POSITIVE"]

Vocabulary: {'this': 1, 'movie': 2, 'sucks': 3, 'good': 4}

After encoding:

DATA = [[1, 2, 3], [4, 2]]
LABELS = [0, 1]

In order to feed this data into the classifier, all input documents must have the same length. So we limit the maximum review length to max_words by truncating longer reviews and padding shorter reviews with a null value (0). Padding to the above data to fixed length 5 would give:

DATA = [[1, 2, 3, 0, 0], [4, 2, 0, 0, 0]]

3.2 Building the Embedding Dictionary

This step involves loading the pre-trained Word2Vec model containing 1 Million words which would be used for associating the token id with the pre-trained vector having 300 dimensions.

For example, if the word2vec file contains these vectors:
{
this: 0.001 0.102 -1.221, movie: 9.011 -0.119 2.112, the: 1.223 0.911 -2.113
}

And the dictionary obtained from the previous step was:
{'this': 1, 'movie': 2, 'sucks': 3, 'good': 4}

Then the embedding dictionary would look like:
{
1: tensor[0.001 0.102 -1.221],
2: tensor[9.011 -0.119 2.112],
3: tensor[0.0 0.0 0.0],
4: tensor[0.0 0.0 0.0]

}

The words in the vocabulary that are not included in the word2vec model would initialized with zeros

3.3 Creating the TensorDataset and DataLoader

Once we have the data ready the next step is to split the data into train and test sets. In this particular implementation we have taken 1900 words for training and the remaining 100 words we have divided into test set and validation set containing 50 words each having uniform distribution of the positive and negative reviews.

Next we create the TensorDatasets and the DataLoaders. TensorDatasets are datasets wrapping around tensors that can be run over GPU and which can be retrieved using indexing over first dimension. Data loader, on the other hand, combines a dataset and a sampler, and provides an iterable over the given dataset. It helps with the loading and shuffling of data in batches required while training. Figure 3 shows a sample data generated for the loader having 50 as batch size and 200 as sequence length.

```
Sample input size:  torch.Size([50, 200])
Sample input:
 tensor([[  24,   65, 1011,  ..., 4692,    2, 1999],
        [  72,   87,   14,  ..., 6667,   31,    4],
        [   1,  240,    9,  ..., 1940,    7,    7],
        ...,
        [  11,   18,   14,  ...,  151,   23,  786],
        [   0,    0,    0,  ...,   21,  397,  243],
        [   0,    0,    0,  ...,  129, 1314, 3622]], dtype=torch.int32)

Sample label size:  torch.Size([50])
Sample label:
 tensor([0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
        0, 0], dtype=torch.int32)
```

Figure 3 Sample batch generated by the DataLoader

3.4 Define the Deep Learning Models

The next step is to define the model for classification. Here we are implementing three different deep learning models and comparing their performance for the given dataset. The three different models tried out in this project are:

1) Baseline model
2) RNN model
3) Self-attention

Figure 4 shows the simplified architecture for the three models implemented.

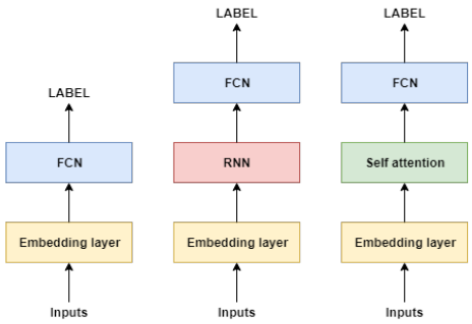

Figure 4 Simplified architecture for the models implemented

The Baseline model consists of two layers:

1. Embedding layer: This takes in an input sample and represents it as the word embeddings obtained from the previous task.
2. FCN: A fully connected layer is added on top of the Embedding layer that gives the class value associated with the input.

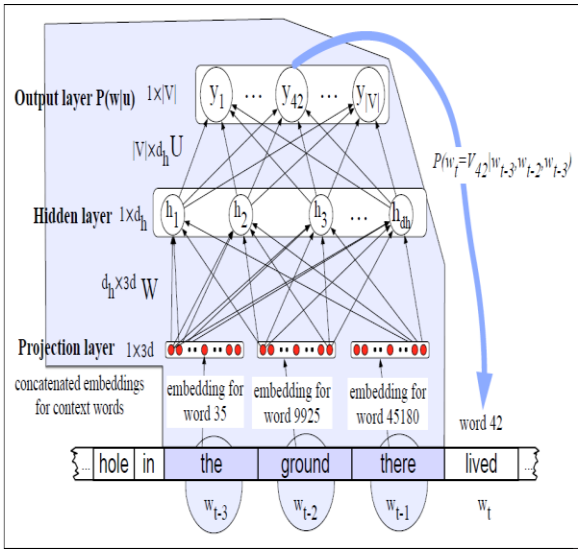

Figure 5 Language Model for Feedfoward Neural Network

In RNN model we add a RNN layer in between the embedding and the fully connected layers. Unlike the feedforward neural networks, RNN can use internal memory to process a sequence of input which makes it popular for processing sequential information. There are different types of RNNs available depending on the topology of the network. Here we have implemented the Vanilla RNNs, LSTM and GRU for comparison purpose. Figure 5, Figure 6 , Figure 7 and Figure 8 show the topology for the different models. The key differences GRU and LSTM differ from simple RNN is the network topology. In GRU an update gate is introduced, to decide whether to pass Previous output (ht-1) to next cell (ht) or not. Whereas in LSTM, two additional Gates are introduced (Forget and Output) in addition to the update gate of GRU.

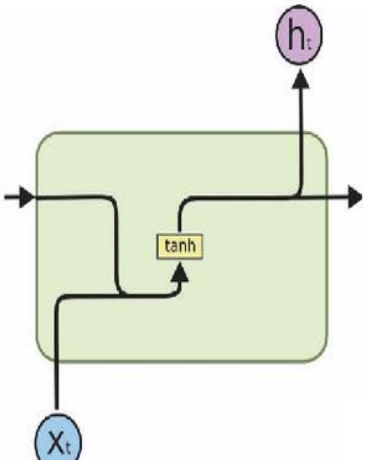

Figure 6 Architecture for Vanilla RNN

In the Self-attention method we replace the RNN layer with the self-attention layer. Attention is a concept that helped improve the performance of neural machine translation applications by taking more words into the context compared to the RNN models. We'll replace it with the Transformer – a model that uses attention to boost the speed with which these models can be trained. Figure 9 shows the architecture of the self-attention layer.

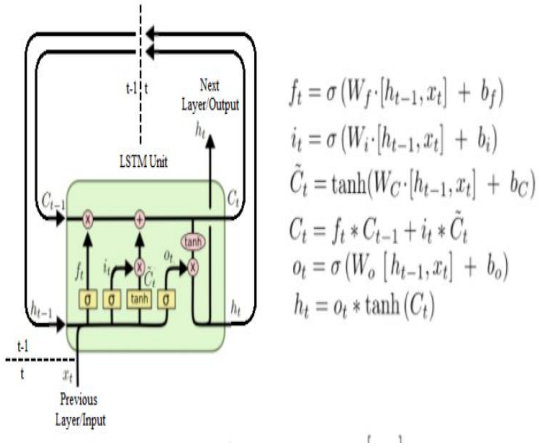

Figure 7 LSTM Cell Architecture

3.5 Train and Test the Models

For training the model we define a loss function and optimizing function. We use the binary cross-entropy loss function and use the stochastic gradient descent as the optimizer to the objective function. We train the model for a number of epochs using the training dataset. Then use the validation and test data set to test the accuracy of the model. The accuracy can change based on different values of the hyperparameters, like the batch size, number of epochs, learning rate, number of layers, of the model. The detailed results and the observations from running the models is provided in the next section.

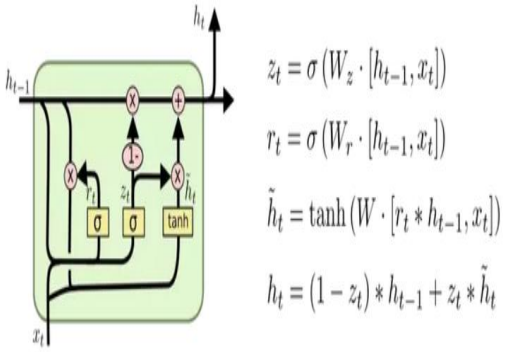

Figure 8 Architecture for GRU RNN

4. Experiment Result Analysis

As discussed in the previous section the IMDB dataset was trained on several deep learning models. Each of these models were run several times with different parameter values in order to see the influence of these parameters in the overall accuracy of the model. Table 1 shows a detailed comparison of the accuracy achieved for different model parameters. Accuracy is the metric used for comparing the results.

As evident from the table the accuracy of most of the models were in range of 40-60%. The RNN LSTM and GRU models fared comparatively better compared to other models. They could achieve maximum accuracy of 62%.

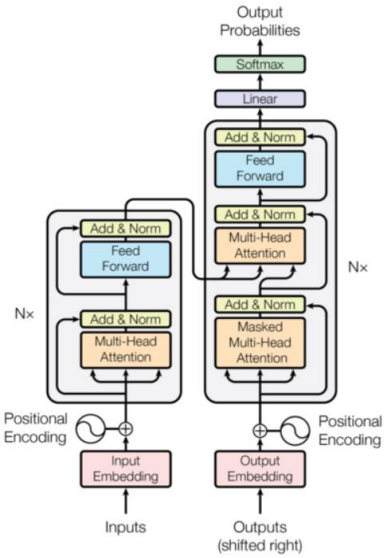

Figure 9 Architecture for Self-Attention Transformer

Additionally from the table we can generally observe that the accuracy increased with increase in number of epochs for almost all deep learning models. Changing the hyperparameters significantly affected the accuracies of the model. As seen in the Baseline model, the accuracy was higher when the learning rate was high. In the case of RNN-LSTM model, the accuracy was higher when the batch size is 25 as compared to the batch size of 50. In addition to the batch size the padded sequence length also significantly affected the accuracy. The average length of the movie reviews in the dataset was close to 500. Keeping the padded sequence length to 600 gave better accuracies as compared to the sequence length of 200.

Adding bi-directionality to the RNN models increased the training time almost twice. While it improved the accuracy in case of

Vanilla RNN, it didn't perform well when run on LSTM and GRU RNN models for the given test scenarios. Replacing the RNN layer with self-attention layer significantly improved the training time i.e. it took less time for training. However no improvement in accuracy was observed in this scenario. Another observation the accuracy in self-attention increased significantly with the number of epochs. So it is possible after training it for higher number of epochs, it might perform better.

Table 1 Detailed comparison of Accuracies for different Deep Learning Models

Deep Learning Model	Model Parameters	Accuracy	Training time(s)		Deep Learning Model	Model Parameters	Accuracy	Training time(s)		Deep Learning Model	Model Parameters	Accuracy	Training time(s)
Baseline Model	Num Epochs =10 Batch Size = 50 Learning rate = 0.001	38%	192.91		RNN-LSTM	Num Epochs =4 Batch Size = 25 Learning rate = 0.01 Num_layers=1 Biderectional = False Zero-pad seq length =600	56%	2323.516		Self-Attention	Num Epochs =4 Batch Size = 50 Learning rate = 0.01 Num_heads=1	50%	264.145
	Num Epochs =50 Batch Size = 50 Learning rate = 0.1	52%	953.937			Num Epochs =10 Batch Size = 25 Learning rate = 0.01 Num_layers=1 Biderectional = False	50%	2487.579			Num Epochs =10 Batch Size = 50 Learning rate = 0.01 Num_heads=1/2	46%	726.65
	Num Epochs =100 Batch Size = 50 Learning rate = 0.02	58%	1805.11			Num Epochs =10 Batch Size = 25 Learning rate = 0.01 Num_layers=2 Biderectional = False Zero-pad seq length =600	62%	15334.606			Num Epochs =20 Batch Size = 25 Learning rate = 0.01 Num_heads=1 zero-pad length =600	54%	2298.47
	Num Epochs =50 Batch Size = 50 Learning rate = 0.001	48%	934.32			Batch Size = 50 Learning rate = 0.01 Num_layers=2 Biderectional = True	52%	2701.887					
RNN- Vanilla	Num Epochs =4 Batch Size = 50 Learning rate = 0.01 Num_layers=1 Biderectional = False	40%	188.062			Num Epochs =10 Batch Size = 50 Learning rate = 0.01 Num_layers=2 Biderectional = True	52%	8698.72					
	Num Epochs =10 Batch Size = 50 Learning rate = 0.01 Num_layers=1 Biderectional = False	52%	448.55		RNN-GRU	Num Epochs =4 Batch Size = 50 Learning rate = 0.01 Num_layers=2 Biderectional = False	54%	985.92					
	Num Epochs =10 Batch Size = 50 Learning rate = 0.01 Num_layers=2 Biderectional = False	48%	787.31			Num Epochs =10 Batch Size = 50 Learning rate = 0.1 Num_layers=2 Biderectional = False	62%	2492.83					
	Num Epochs =10 Batch Size = 50 Learning rate = 0.01 Num_layers=2 Biderectional = True	54%	1446.544			Num Epochs =10 Batch Size = 50 Learning rate = 0.01 Num_layers=2 Biderectional = True	48%	5198.48					

5. Summary and Conclusion

The goal of this project was to perform a binary classification the movie reviews sentiment as positive and negative. We started off with introducing the sentiment analysis problem in Natural Language Processing and briefly highlighting the research trends in the recent years. It was followed by stating why we are using deep learning models to perform this task and proposed a high level breakdown of the tasks involved in this project. We discussed each of the tasks in details. Finally we implemented different deep learning models and experimented with different parameters and compared the accuracies among different models.

As observed from the previous section, we got accuracy in range of 40-60% across all the models. RNN-LSTM and RNN-GRU models gave the highest accuracy of 62%. The dataset we used in this project had only 2000 reviews. So these results can serve as the baseline of what these models can achieve. As future work, these models can be trained on datasets having more movie reviews i.e. around 50000 and also for higher number of epochs. With more architectural exploration and fine-tuning these models can achieve very good accuracies.

6. References

[1] https://www.brandwatch.com/blog/understanding-sentiment-analysis/

[2] https://www.cs.uic.edu/~liub/FBS/SentimentAnalysis-and-OpinionMining.pdf

[3] Mäntylä, Mika V., Daniel Graziotin, and Miikka Kuutila. "The evolution of sentiment analysis—A review of research topics, venues, and top cited papers." *Computer Science Review* 27 (2018): 16-32.

[4] https://towardsdatascience.com/sentiment-analysis-using-lstm-step-by-step-50d074f09948

[5] Zhang, Lei, Shuai Wang, and Bing Liu. "Deep learning for sentiment analysis: A survey." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.4 (2018): e1253.

[6] http://jalammar.github.io/illustrated-transformer/

[7]https://colab.research.google.com/github/agungsantoso/deep-learning-v2-pytorch/blob/master/sentiment-rnn/Sentiment RNN Exercise.ipynb

[8] https://towardsdatascience.com/how-to-code-the-transformer-in-pytorch-24db27c8f9ec

[9]https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57