Assignment 2: *Rotten Tomatoes Movie Reviews Classification*

Bhargav Lad
Department of Computer Science
Lakehead University
Thunder Bay, ON Canada
bamratbh@lakeheadu.ca

***Abstract*— This report explains convolutional neural networks (CNN) trained on top of pretrained GLOVE word vectors for sentence-level classification tasks. We show that a simple CNN with little hyperparameter tuning and dense vectors achieves excellent results on multiple benchmarks. Learning task specific vectors through finetuning offers further gains in performance. In this work, we use rotten tomatoes movie review dataset. We test our model's performance using precision, recall and F1 score.**

Github: https://github.com/isbhargav/Rotten-Tomatoes-Classification

Keywords—Sentiment Classification, 1D Convolution.

I. Introduction

In today's time where everything is being digitized, we have a huge number of documents which are available on the internet and easily accessible to everyone. Finding information as per the user's requirement is becoming increasingly important. Most of the information available on the internet is in text format. We obtain a huge number of review documents that include user's opinions from online sources such as yelp, amazon etc. These reviews can help users to decide if they should go for the product/service or not. Therefor methods which are effective and accurate are required to evaluate these reviews. This task of classifying reviews into discrete categories is known as sentiment analysis.

Sentiment analysis is a classification task of labeling review document according to the polarity of its prevailing opinion. Generally, sentiment classification algorithm is a supervised learning algorithm where we have a small collection of labeled documents which is used to train our model and the rest of the unlabelled documents are used as test set.

II. Literature Review

Deep learning models have achieved remarkable results in computer vision and speech recognition in recent years. Within natural language processing, much of the work with deep learning methods has involved learning word vector representations through neural language models [1] and performing composition over the learned word vectors for classification[2].Word vectors, wherein words are projected from a sparse, 1-of-V encoding (here V is the vocabulary size) onto a lower dimensional vector space via a hidden layer, are essentially feature extractors that encode semantic features of words in their dimensions. In such dense representations, semantically close words are likewise close in euclidean or cosine distance in the lower dimensional vector space[3].

Generally, RNN or CNN based architecture are used for NLP task such as sentiment analysis. The CNN network tries to extract information about the local structure of the data by applying multiple filters (each having different dimensions). The RNN based better suited to extract the temporal correlation of the data and dependencies in the text snippet.

CNNs have been very successful for several computer vision and NLP tasks in the recent years. They are specially powerful in exploiting the local correlation and pattern of the data through learned by their feature maps. One of the early works which used CNN for text classification is by Kim [3], which showed great performance on several text classification tasks.

To perform text classification with CNN, usually the embedding from different words of a sentence (or paragraph) are stacked together to form a two-dimensional array, and then convolution filters (of different length) are applied to a window of h words to produce a new feature representation. Then some pooling (usually maxpooling) is applied on new features, and the pooled features from different filters are concatenated with each other to form the hidden representation. These representations are then followed by one (or multiple) fully connected layer(s) to make the final prediction. Figure 1 shows general architecture of CNN.

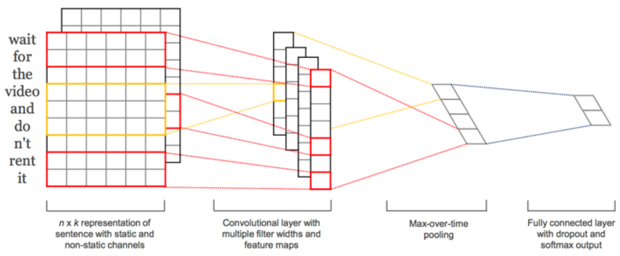

wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

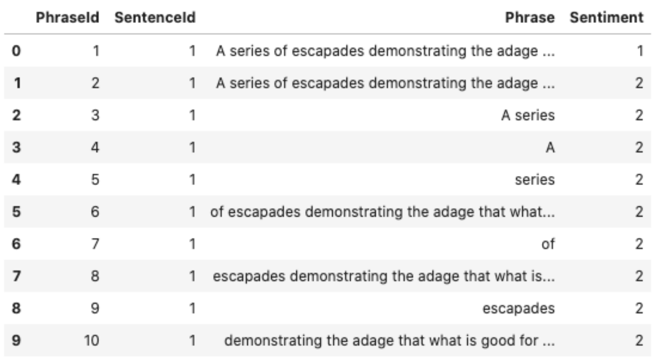
Figure 1: General Architecture of CNN based Networks

The most natural fit for CNNs seem to be classifications tasks, such as Sentiment Analysis, Spam Detection or Topic Categorization. Convolutions and pooling operations lose information about the local order of words, so that sequence tagging as in PoS Tagging or Entity Extraction is a bit harder to fit into a pure CNN architecture (though not impossible, you can add positional features to the input).[5]

III. Data Analysis

A. Data Source

The dataset is obtained from *https://github.com/cacoderqua /Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review - dataset/blob/master*. The data is in TSV format and provides information like sentence Id, Phase Id, Phrase and Sentiment. Here, each review if subdivided in phases and if given a sentiment. In this project we will try to predict the sentiment given the phrase. This is a multi-class classification problem.



	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2
5	6	1	of escapades demonstrating the adage that what...	2
6	7	1	of	2
7	8	1	escapades demonstrating the adage that what is...	2
8	9	1	escapades	2
9	10	1	demonstrating the adage that what is good for ...	2

Figure 2: Dataset

B. *Preprocessing*

We will be using Artificial Neural Network(ANN) for this task, therefore we need to convert text to numbers in order to feed it into the network. For preprocessing we will use trochtext and SpaCy. These are open-source library for natural language processing in python. We first change each word to lower case so that our vocabulary does not contain duplicate entries for same word. We then use SpaCy tokenizer to tokenize the text and build vocabulary out of it. We will also use pretrained GLOVE embeddings hence we need to numericalize our tokens same way as the GLOVE tokens.

C. *Train and Test Split*

We split out data using sklearn's train_test_split method into 7:3 ratio where 70% samples are in our training dataset and 30% samples are in our testing set. We also set our random seed equal to 2003 for the reproducibility of our experiment.

D. *Visualiing the dataset*

Visualizing is very import aspect of analyzing the data. Here we will use matplotlib to visualize our dataset.

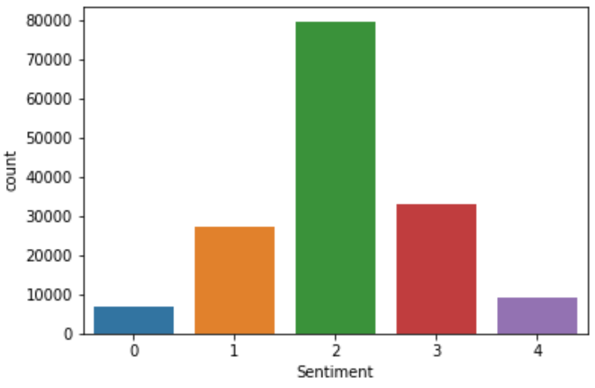


Figure 3: Distribution of data

Figure 3 shows number of instances in each class. We can clearly see that out dataset is biased towards class 2.

IV. Model Architecture

The first layer of the model is an embedding layer which takes in a sequence x which contains 53 entries. Each entry is represented by 50-dimensional dense vector thus input x is represented as feature map of dimensionality 53x50. This embedding layer is initialized with GLOVE weights. After this we have 6 convolution layer with filters 64, 100, 128, 150, 200 each followed by Exponential Linear Unit(ELU) activation layer. The final layer of the network is a dense layer with 5 output neurons for each class and we use softmax on this to get probability of each class.

V. Experimental Setup

Tools and Libraries used for the experiment.

- Jupyter Notebook : Development environment
- Pandas : Reading the data
- Sklearn : Split the data
- Pytorch : Tensor library
- Torchtext : preprocessing text data
- SpaCy : Tokenize the text.
- Poutyne : Training and Metrics
- Livelossplot: Plot live metrics during training

VI. Loss Function and Metrics

For the experiment we optimize our network on Cross Entropy loss which is also known as log loss. Cross Entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. It is one of the commonly used loss functions for classification problems.

$$L(\hat{y}, y) = -\sum_{k}^{K} y^{(k)} \log \hat{y}^{(k)}$$

Figure 4: Cross Entropy Loss

To measure accuracy of the model we use recall, precision and F1 score. Precision is the number of True Positives divided by the number of True Positives and False Positives. Put another way, it is the number of positive predictions divided by the total number of positive class values predicted. Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. Put another way it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity. F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. The best possible value for F1 score is 1 and the worst at 0.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

TP = True positive

TN = True negative

FP = False positive

FN = False negative

Figure 5: Precision, Recall and F1 Sore

VII. Hyerparameter Initialization

We will use pretrained GLOVE embeddings hence we initialize our embedding layer with GLOVE weights. To train our model we will use adaptive learning rate optimization algorithm called ADAM. We have used pytorch's implementation of ADAM with initial learning rate of 0.003.

VIII. Results

The following results were obtained during the experiment. Figure 6 and 7 shows the accuracy and loss after each epoch.

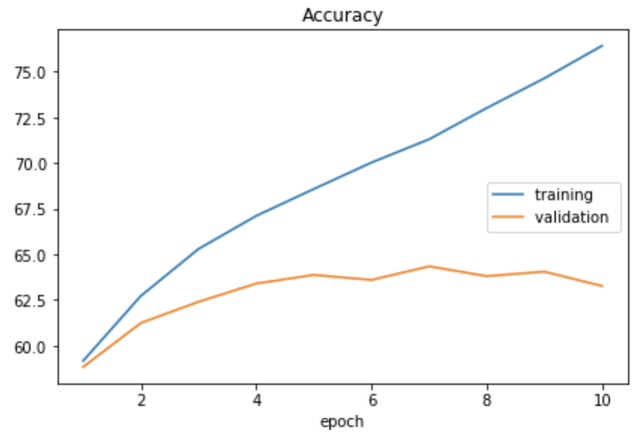


Figure 6: Epoch vs Accuracy

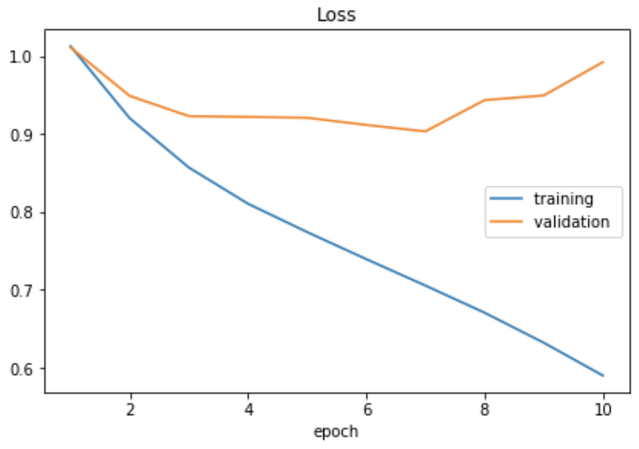


Figure 7: Epoch vs Loss

No	Results			
	Dataset	***Precision***	***Recall***	***F1***
1	Training Set	0.764	0.765	0.77
2	Testing Set	0.632641315	0.632641315460	0.63264131546020

IX. Conclusion

In this project, we have used Rotten Tomatoes Movie Reviews dataset to perform Multi-class classification task using 1D convolution neural network. In the above experiment we were able to show how you can train you convolution neural net using text data. We looked at how we can preprocess text data and feed it into a neural network. We also saw how to improve the performance of model by using pretrained embeddings which is trained on huge corpus. For the overfitting problem we use drop out and L2 regularization in our network.

The prediction accuracy can be improved by tuning both the algorithm and the data for specific applications. Although this model has low accuracy as a prediction model, it provides a preliminary framework for further analyses.

Acknowledgment

This project is part of Assignment 2 of the Natural Language Processing course. The author would like to thank Dr Akhilan and GAs of the course for their valuable support.

References

[1] Y. Bengio, R. Ducharme, P. Vincent. 2003. Neu-ral Probabilitistic Language Model.Journal of Ma-chine Learning Research3:1137–1155.

[2] R. Collobert, J. Weston, L. Bottou, M. Karlen, K.Kavukcuglu, P. Kuksa. 2011. Natural LanguageProcessing (Almost) from Scratch.Journal of Ma-chine Learning Research 12:2493–2537.

[3] Kim Y. Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882. 2014 Aug 25.

[4] Tsutsumi K, Shimada K, Endo T. Movie review classification based on a multiple classifier. InProceedings of the 21st pacific Asia conference on language, information and computation 2007 Nov (pp. 481-488).

[5] Lopez MM, Kalita J. Deep Learning applied to NLP. arXiv preprint arXiv:1703.03091. 2017 Mar 9.