Multi-Class Sentiment Analysis using CNN

Pinak Divecha
Department of Computer Science
Lakehead University
Student id. 1101608
pdivecha@lakeheadu.ca

***Abstract*—This paper is targeted to finding the most effective One dimensional convolution neural network for predicting the sentiment value using the movie reviews which is available in text form. In order to predict the sentiment value I have used the Rotten Tomatoes movie review data set which was available on GitHub. I have build a nonlinear regression model for predicting the y attribute which is sentiment value of the given movie review. This Convolution Neural Network(CNN) model is build up of 2 convolution layer and used Rectified Linear Unit (RELU) as the activation function. I have done the hyper-parameter tuning in order to find the value of parameters for which model gives the highest accuracy and lowest L1Loss. After getting best hyper-parameters I am able to achieve the accuracy of 66.5%.**

***Index Terms*—Convolution Neural Network, Keras, Natural Language Processing**

I. Introduction

Convolution neural network is not only used for developing computer vision application such as image classification but also it is used solving the problems related to Natural Language Processing(NLP). CNN is nothing just several layers of convolution which uses Rectified Linear Unit (RELU) as an activation function. RELU is a non- linear activation function hence it makes the model build becomes non-linear model.
This paper talk about predicting the sentiment value of based on movie review. There are total five sentiment values whose description is is shown in table 1.

TABLE I
Understanding Sentiment value and its meaning

Sentiment Meaning	Sentiment Value
Very Negative	0
Negative	1
Neutral	2
Positve	3
Very Positive	4

II. Literature Review

Convolution neural network has a wide application in the domain of machine learning, image classification and natural language processing. CNNs are like NN which are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output[3]. Each Layer takes the input from the previous layer and preserves the relationship between the attributes. After doing number of convolution it passes throught the RELU which performs non-linear operation. RELU is important as it bring the non0linearity in our conv-net. The output of RELU becomes the input for the pooling layer. There are three types of pooling:

- Sum Pooling
- Avg Pooling
- Max Pooling

Out of the above types of pooling, majorly max pooling is used. After doing pooling we flatten our matrix vectors feed into the fully connected neural network. Finally, we have an activation function such as softmax to classify the outputs.[5] Similarly, in this assignment I have used the CNN for predicting the sentiment value based on textual movie review.

III. Proposed CNN Model

A. Dataset

The Rotten Tomatoes movie review corpus is a collection of movie reviews collected by Pang and Lee in [2]. This corpus has been analysed in [3] where each sentence is parsed into its tree structure and each node is assigned a fine-grained sentiment label ranging from 0 to 4 where the numbers represent very negative, negative, neutral, positive and very positive respectively. I have total 156060 instances and out of which I took 109242 as training and 46818 as testing which is exactly as 70:30 ratio of the total data-set. Also, while doing the division I kept the random_state value to 2003. Out of all the 5 different types of sentiments, highest instances are of "Neutral" and lowest is of "Highly Negative". Distribution of the instances across the sentiment value is shown in figure 1.

B. Libraries

In order to do the classification using 1D-CNN, I have used the pandas library to load the data and do some pre-processing part. In order to do mathematical calculation or converting the pandas data-frame to array I have used numpy. Finally, in order to develop the CNN model I have used the Keras framework. So, below is the list of library used:

- pandas
- numpy
- keras
- random
- matplotlib
- sklearn

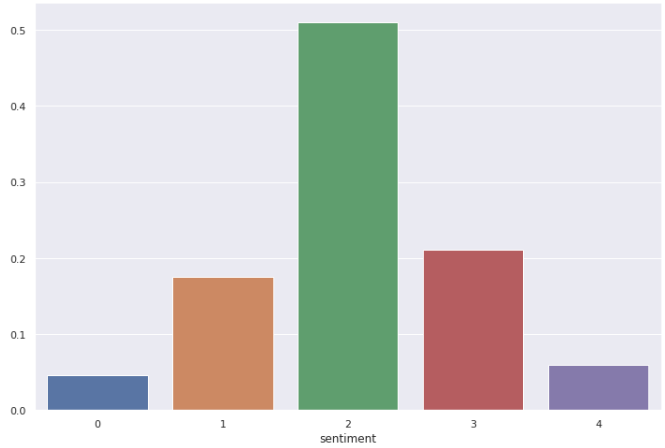

Fig. 1. Sentiment Distribution

C. Data Pre-Processing

The given data-set consists of the movie review of the form of string. This string need to be processed in order to develop a proper and efficent vectorization of the training dataset. In order to do that I have used the following pre-processing steps:

a) Removing Stop-Words: I have removed stop-words(which include "the", "a", "an", "in") because these words don't waste space in my database, and don't take up valuable processing time. I can remove them easily, by storing a list of words that you consider to be stop words. I have used Natural Language Toolkit(NLTK) in python because it has a list of stop-words stored in 16 different languages.

b) Removing Punctuation: An important point to note is that - stop-word removal doesn't take off the punctuation marks or newline characters. We will need to remove them manually. I have taken twelve punctuation which are as follows- ? : ! . , ; ' " - ( ).

c) Lemmatization: For grammatical reasons, documents are going to use different forms of a word, such as organize, organizes, and organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy, democratic, and democratization. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. In order to do that I have used the most used and efficient python library name NLTK.

This were three steps which I used to pre-process my phases/sentences of the data-set. Also, I have performed this pre-processing only on training data(not on testing data). Using this pre-processing the number size of the array has decreased drastically as pre-processing step helps to remove the unwanted words and punctuation.

D. Text Vectorization

Processing natural language text and extract useful information from the given word, a sentence using machine learning and deep learning techniques requires the string/text needs to be converted into a set of real numbers (a vector) — Word Embeddings. Word Embeddings or Word vectorization is a methodology in Natural Language Processing(NLP). To map words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/semantics. In order to do the Word Vectorization, I have used the Term Frequency-Inverse Document Frequency(TF-IDF) technique

a) Term Frequency-Inverse Document Frequency: TF-IDF contain two concept Term Frequency(TF) and Inverse Document Frequency(IDF). Term Frequency is defined as how frequently the word appear in the document or corpus. Term frequency can be mathematically described as:

$$TF = \frac{Number\_of\_times\_word\_appear\_in\_the\_docs}{Total\_number\_of\_word\_in\_the\_docs}$$

While Inverse Document frequency(IDF) is another concept which is used for finding out importance of the word. It is based on the fact that less frequent words are more informative and important. IDF is represented by formula:

$$IDF = \log_{10} \frac{Number\_of\_docs}{Total\_No\_of\_docs\_in\_which\_word\_appears}$$

TF-IDF is basically a multiplication between TF value and IDF value. It basically reduces values of common word that are used in different document. That is the reason of using the TF-IDF.

E. Proposed Model

I have used Keras which provides a Sequential model API. The Sequential model API is a way of creating deep learning models where an instance of the Sequential class is created and model layers are created and added to it. I used the 'add()' function to add layers to our model. I have created a CNN model with one function: feed. The feed() function includes the pre-processing steps outline above and also compute the a forward pass for the CNN. So, when the instance of the class, CnnRegressor, is created, I define internal functions to represent the layers of the net. During the feed pass, we call these internal functions. I have used 4 major function which are:

- Conv1D - This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs.
- MaxPooling1D – Max pooling operation for temporal data.
- Flatten Layer - This layer that flattens an input
- Dense Layer- Regular densely-connected Neural Network layer

After the Keras model model is ready, I need to build training model. In order to make training model, we need

to find the best and optimum hyper parameters which gives the best results for our proposed model. After doing Hyper-Parameter tuning, I have came up with the optimum hyper-parameters which are shown in table 2.

TABLE II
OPTIMUM HYPER-PARAMETERS FOR PROPOSED MODEL

Hyper-Parameter	Value
Optimizer	Adadelta
Epoch	15
Batch Size	128
Number of Layer	2
Loss Function	categorical_crossentropy

From the table 3 it can observed that the Adadelta optimizer works well and give the highest r2 score and lowest L1Loss.

TABLE III
COMPARISON OF OPTIMIZER AND BATCH SIZE AND ACCURACY

Batch Size	Optimizer	Inference Time	Accuracy
264	Adadelta	514.72	0.619
128	Adadelta	523.39	0.6228
64	Adadelta	503.17	0.643
264	RMSprop	511.89	0.593
128	RMSprop	524.52	0.6028
64	RMSprop	513.03	0.61

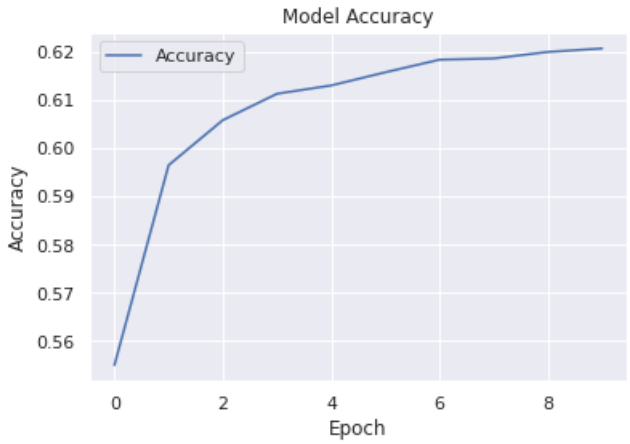

Fig. 2. Accuracy vs Epoch

F. Trainable Hyper Parameters

Hyper-parameters are the variables which determine how the network is trained. Hyper-parameters are set before contracting the model. In my model following are the hyper-parameters:

- Drop-out rate- 0.25, 0.50
- Epochs- 10 to 30
- Batch Size- 128, 64, 32
- Optimizer- Adadelta, Rprop
- Kernel Size- 3 to 5

After analysing the accuracy for different combination of the hyper-parameters, I have received highest accuracy when optimizer was Adadelta, epoch was 25 and batch size was 64. I received accuracy of 66.5% for the testing dataset.

G. Inference Time

In machine learning and NLP domain inference time is the time taken by the trained model used to infer/predict the testing array or samples. I have used python library to calculate the inference time that is time taken to train the model. In order to do that I have the 'timeit' library. On an average model takes around twelve minutes to build the model.

H. Model Storing

After creating the keras CNN model, I have stored the model into the drive. In order to save the model I have first connected my drive with the colab mount function. Once the drive was mounted, I used the keras model save function to save the model in h5 extension. Also, I tested the saved model to by loading the model again and testing it using the test instances.

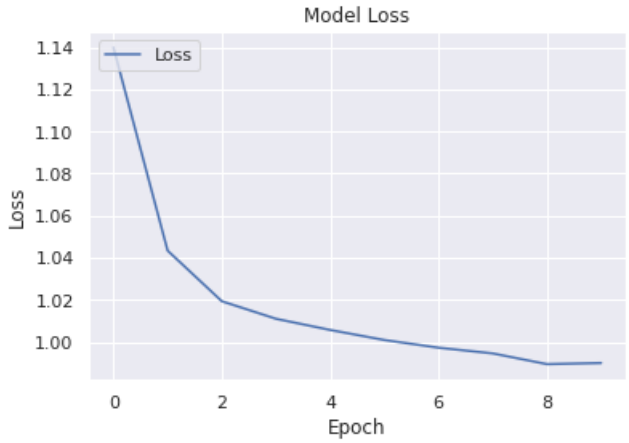

Fig. 3. Loss vs Epoch

I. Results

After doing proper pre-processing method which I explained in above sections, I converted the textual movie review in numerical form using TF-IDF method. I used TF-IDF as it is the most suited and reliable method for sentiment analysis method. I feed the matrix into CNN and received the results which is show in table four.

TABLE IV
TESTING RESULTS

	Values in %
Accuracy	66.5
Recall	60.8
F1 score	62.3
Loss	0.91
Precision	69.4

ACKNOWLEDGMENT

I want to deeply thanks my professor Dr. Thangarajah Akilan who helped me understand the concept of CNN and help me in my class labs. I also want thank the TA's Andrew and Punardeep who took keen interest and solved my issues and concerns about the assignments on time.

CONCLUSION

In this assignment, I have used the Rotten movie review data-set containing the more than One Lakh instances and 5 sentiment values. In this project, I have used the CNN model in order to predict the sentiment value based on the textual based movie review. I perform proper pre-processing method, I have converted the sentences into the matrix format by using the TF-IDF method. I used the matrix to build the CNN model using Keras framework and evaluated the model after doing proper hyper-parameter tuning. After hyper-parameter turning, I am able to achieve the highest testing accuracy value of 66.5%.

REFERENCES

[1] https://keras.io/getting-started/sequential-model-guide/
[2] https://towardsdatascience.com/fine-grained-sentiment-analysis-in-python-part-1-2697bb111ed4
[3] https://keras.io/getting-started/faq/how-can-i-save-a-keras-model
[4] https://towardsdatascience.com/report-time-execution-prediction-with-keras-and-tensorflow-8c9d9a889237