NLP - Assignment 2
Sentiment analysis with Deep Learning

Dhivya Chandrasekaran
Department of Computer Science)
Student ID : 1111017
dchandra@lakeheadu.ca

***Abstract*—This article describes the various steps taken to implement Sentiment Analysis using Rotten Tomatoes movie review dataset using 1-Dimensional Convolutional Neural Network. Preprocessing of text data is carried out by removing punctuations, special characters and stop words. Lemmatization is done with WordNet Lemmatizer and the vectorisation is carried out using pretrained Glove vectors. A 1D Conv model is built using Keras and classification metrics like accuracy, recall, precision and f1score are calculated. This report records the various steps along the process and the achieved results in detail.**

I. Introduction

Sentiment Analysis of text data is an important Natural Language Processing Task. Various business ventures adopt sentiment analysis as a good marketing tool to identify the acceptance rate of the product in the market and to fine tune the efforts to attract a specific group of people. In this assignment, the given task is to perform Sentiment Analysis on the Rotten Tomatoes Movie Review Dataset using a 1 dimensional Convolutional Neural Network(Conv1D). The given text data is converted to real value vectors called word-embeddings and the vectors are used to predict the class to which the given sentence/phrase is aligned to. The final prediction is performed using softmax activation. The article provides a brief overview of Text data processing in Section II and the experiment analysis of the model is carried out in Section III and section IV records the results obtained.

II. Background

The major challenge in processing Natural Language is the process by which we convert the given text data to machine readable format. Sentiment Analysis of text data maybe considered as a classification problem were the given text is assigned to a set of class variables. Earlier probabilistic models were proposed to handle the problem of text classification. However due to the versatility of natural language, there was never a dataset big enough to encompass all the possible combinations of the text data. Machine learning approaches like Naive-Bayes, Random-Forrests etc., were proposed for text classification. The given sentences were converted to vectors using techniques like Bag-of-Words(BoW), Term frequency-Inverse Document Frequency(Tf-Idf) that used frequency of a word over a given corpus as the vector values and then cosine similarity between these vectors were calculated and the similar words were grouped to a defined class based on their distribution in the training dataset. However these techniques didnot capture the semantic properties of the text data. Hence word-embedding using neural networks like Word2Vec and using word-co occurence matrix like GloVe vectors. These vectors were formed over a high dimensional space and they captured the semantic properties for example the vectors of the words that are similar are closely aligned.

A. *word2vec*

word2vec: [1] is developed from Google News Dataset containing approximately 3 million vector representations of words and phrases. $word2vec$ is a neural network model used to produce distributed vector representation of words based on an underlying corpus. There are two different models of $word2vec$ proposed: The Combined Bag of Words(CBOW) and the Skip-gram model. The architecture of the network is rather simple and contains an input layer, one hidden layer and an output layer. The network is fed with a large text

corpus as the input, and the output of the model is a vector representation of words. The CBOW model predicts the current word using the previous words while the Skip-gram model predicts the neighboring context words given a target word.

B. *GloVe:*

GloVe [2] was developed by Stanford University, relies on a global word co-occurrence matrix formed based on the underlying corpus. It estimates similarity based on the principle that words similar to each other occur together. The co-occurrence matrix is populated with occurrence values by doing a single pass over the underlying large corpora. $GloVe$ model was trained using five different corpora mostly Wikipedia dumps.

III. Experimental Analysis and Comparison

The entire code for the assignment is provided in Appendix A

A. Dataset:

For this assignment the *Rotten Tomatoes Movie Review Dataset* is used. This dataset for originally by Pang and Lee [3]. The training dataset contains a total of 8544 sentences that are parsed using Stanford Parser to form 156060 phrases. Each phrase has a phrase id and a sentence id assigned as a feature. The phrases repeated appear only once in the dataset. The dataset contains 5 different classses as in the Table I

Sentiment Id	Sentiment
0	Negative
1	Somewhat Negative
2	Neutral
3	Somewhat Positive
4	Positive

TABLE I
VARIOUS CLASSES IN THE ROTTEN TOMATOES MOVIE REVIEW DATASET

The dataset has significant class imbalance towards class 2 with nearly 50% of the phrases attributed to class 2.

B. Model building:

The code snippet for the proposed model is show in A. The proposed model has three convolution blocks. Each convolution block has one convolution layer, one batch normalisation layer and one dropout layer. The first convolution has 512 filters, the second convolution has 256 filters and the third convolution layer has 64 filters with a kernel size of 3. Five fully connected layers are attached to the end with 128,100,50,10 and 5 output dimensions respectively. The following table shows the combination of hyper-parameters the model was tuned with.

Hyper-parameters	Option 1	Option 2	Option 3	**Best Option**
Vocabulary size	5000	10,000	13,000	**10,000**
Number of Epochs	10	25	50	**50**
Filter size	3	5	7	**3**
Maximum sentence length	9	15	20	**15**
Embedding dimension	50	200	300	**300**
Activation function	Relu	Sigmoid	Tanh	**Relu**
Optimizer	Adam	RMSProp	NAdam	**Adam**

TABLE II
HYPER-PARAMETER TUNING.

The dataset as mentioned earlier has class imbalance, however oversampling or under-sampling did not result in good results hence the train_test_split is done stratified thus maintaining the same ration of test samples which resulted in improved performance. And early stopping is used to avoid overfitting. The loss function used is categorical_cross_entropy

IV. Results

The data is read from the internet using the code shown in A . The optimal solution for the model is obtained using Adam Optimizer, ReLU activation function, learning rate 0.001, repeated for 50 epochs with early stopping. The training accuracy is 0.7073 . The testing accuracy is 0.6650, precision is 0.5963, Recall is 0.5373, F1 Score is 0.5582 . The loss curve and the accuracy curve are shown in Fig 1 and Fig 2 respectively.

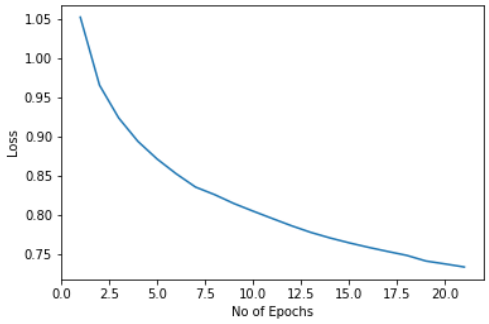

Fig. 1. Loss Curve

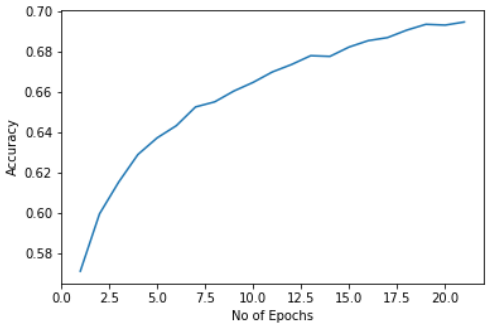

Fig. 2. Accuracy Curve

REFERENCES

[1] Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

[2] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532-1543. 2014.

[3] Pang, Bo, and Lillian Lee. "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales." In Proceedings of the 43rd annual meeting on association for computational linguistics, pp. 115-124. Association for Computational Linguistics, 2005.

APPENDIX

A. Reading data from source to a pandas dataframe

```
import pandas as pd
URL_Tr = 'https://raw.githubusercontent.com/
    cacoderquan/Sentiment-Analysis-on-the-
    Rotten-Tomatoes-movie-review-dataset/
    master/train.tsv'
dataset = pd.read_csv(URL_Tr, sep='\t')
```

B. Importing necessary libraries for data preprocessing

```
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer,
    PorterStemmer, LancasterStemmer
porter = PorterStemmer()
lancaster=LancasterStemmer()
wordnet_lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words("english")
```

C. Function which performs data preprocessing

```
def preprocessing(review,remove_stopwords =
    True, stem_lem = 'word_net'):
  temp = review
  temp.loc[:,"Phrase"] = temp.Phrase.apply(
    lambda x : str.lower(x))
  temp.loc[:,"Phrase"] = temp.Phrase.apply(
    lambda x : " ".join(re.findall('[a-zA-Z]+'
    ,x)))
  if remove_stopwords == True:
    temp.loc[:,"Phrase"] = temp.Phrase.apply(
    lambda x : ' '.join(word for word in x.
    split() if word not in stop_words))
  if stem_lem =='word_net':
    temp.loc[:,"Phrase"] = temp.Phrase.apply(
    lambda x: " ".join([wordnet_lemmatizer.
    lemmatize(word) for word in x.split()]))
  elif stem_lem == 'lancaster':
    temp.loc[:,"Phrase"] = temp.Phrase.apply(
    lambda x: " ".join([lancaster.stem(word)
    for word in x.split()]))
  elif stem_lem == 'porter':
    temp.loc[:,"Phrase"] = temp.Phrase.apply(
    lambda x: " ".join([porter.stem(word) for
    word in x.split()]))
  return temp
```

D. Store preprocessed dataset in a pandas DataFrame and remove null values.

```
preprocessed_review = preprocessing(review,
    remove_stopwords=True,stem_lem='word_net')

preprocessed_review.head()

null_index = preprocessed_review[
    preprocessed_review['Phrase']==''].index
preprocessed_review.drop(null_index,inplace=
    True)
```

E. Get values of columns as list and perform train_test_split at 70:30 ration and random state of 2003.

```
phrases = preprocessed_review['Phrase'].values
y = preprocessed_review['Sentiment'].values

from sklearn.model_selection import
    train_test_split
```

```
phrases_train, phrases_test, y_train, y_test =
    train_test_split(phrases, y, test_size
    =0.30, random_state=2003,stratify = y)
```

F. Code for trying to under-sampling and over-sampling

```
phrases_train_df = pd.DataFrame()
class_4 = phrases_buffer['Sentiment'] == 4
class_4_dataframe = phrases_buffer[class_4]
phrases_train_df = phrases_train_df.append([
    class_4_dataframe]*1,ignore_index=True)
class_0 = phrases_buffer['Sentiment'] == 0
class_0_dataframe = phrases_buffer[class_0]
phrases_train_df = phrases_train_df.append([
    class_0_dataframe]*1,ignore_index=True)
class_1 = phrases_buffer['Sentiment'] == 1
class_1_dataframe = phrases_buffer[class_1]
phrases_train_df = phrases_train_df.append([
    class_1_dataframe]*1,ignore_index=True)
class_3 = phrases_buffer['Sentiment'] == 3
class_3_dataframe = phrases_buffer[class_3]
phrases_train_df = phrases_train_df.append([
    class_3_dataframe]*1,ignore_index=True)
class_2 = phrases_buffer['Sentiment'] == 2
class_2_dataframe = phrases_buffer[class_2]
phrases_train_df = phrases_train_df.append([
    class_2_dataframe]*1,ignore_index=True)
```

G. Import keras tokenzier and tokenize with X_train data.

```
from keras.models import Sequential
from keras import layers
from keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(num_words=13000)
tokenizer.fit_on_texts(X_train)

X_train = tokenizer.texts_to_sequences(X_train
    )
X_test = tokenizer.texts_to_sequences(
    phrases_test)
```

H. Code for padding and converting class variable to one-hot encoded values

```
from keras.preprocessing.sequence import
    pad_sequences
sentence_length = 15
X_train = pad_sequences(X_train, padding='post
    ', maxlen=sentence_length)
X_test = pad_sequences(X_test, padding='post',
     maxlen=sentence_length)

X_train.shape

from keras.utils import to_categorical
y_train = to_categorical(y_train, 5)
y_test = to_categorical(y_test, 5)

import numpy as np
```

I. Function for creating the embedding matrix

```
def create_embedding_matrix(filepath,
    word_index, embedding_dim):
    vocab_size = len(word_index) + 1  # Adding
     again 1 because of reserved 0 index
    embedding_matrix = np.zeros((vocab_size,
    embedding_dim))

    with open(filepath) as f:
        for line in f:
            word, *vector = line.split()
            if word in word_index:
                idx = word_index[word]
                embedding_matrix[idx] = np.
    array(
                    vector, dtype=np.float32)
    [:embedding_dim]

    return embedding_matrix

embedding_dim = 300
embedding_matrix = create_embedding_matrix('/
    content/glove.6B.300d.txt',tokenizer.
    word_index, embedding_dim)
```

J. Define vocabulary size

```
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

K. Define the classification model

```

model = Sequential()
model.add(layers.Embedding(vocab_size,
    embedding_dim,weights=[embedding_matrix],
    input_length=sentence_length,trainable=
    False))
model.add(layers.Conv1D(512, 3, activation='
    relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))
model.add(layers.Conv1D(256, 3, activation='
    relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))
model.add(layers.Conv1D(128, 3, activation='
    relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))
model.add(layers.MaxPool1D())
model.add(layers.Conv1D(64, 3, activation='
    relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(128,activation = 'relu'
    ))
model.add(layers.Dense(100, activation='relu')
    )
model.add(layers.Dense(50, activation='relu'))
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(5, activation='softmax'
    ))
```

L. Build the model

```
model.compile(optimizer='adam',loss='
    categorical_crossentropy',metrics=['
    accuracy'])
model.summary()
```

M. Import and Initialize Early Stopping from keras

```
from keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', mode='
    min', verbose=1, patience=10)
```

N. Train the model

```
history = model.fit(X_train, y_train,epochs
    =50,verbose=True,validation_data=(X_test,
    y_test),batch_size=25,callbacks=[es])
```

O. Save the model

```
model.save('1111017_1dconv_reg')
```

P. Load the saved model

```
from keras.models import load_model
model = load_model('1111017_1dconv_reg')
```

Q. Plot the results

```
training_dict = history.history
accuracy_val = training_dict['acc']
loss_val = training_dict['loss']
epochs = range(1, len(loss_val) + 1)
plt.figure()
plt.xlabel("No of Epochs")
plt.ylabel("Loss")
plt.plot(epochs,loss_val)
plt.figure()
plt.xlabel("No of Epochs")
plt.ylabel("Accuracy")
plt.plot(epochs,accuracy_val)
```

R. Test the model

S. Calculate the accuracy, recall, precision and f1_score for testing sample.

```
from sklearn.metrics import f1_score,
    precision_score, recall_score,
    confusion_matrix
import numpy as np

from collections import Counter
rounded_labels=np.argmax(y_test, axis=1)
y_pred1 = model.predict(X_test)
y_pred = np.argmax(y_pred1, axis=1)
print(y_pred)
precision = precision_score(rounded_labels,
    y_pred , average="macro")
print("The Precision is : %.2f" % precision)
recall = recall_score(rounded_labels, y_pred ,
     average="macro")
print("The Recall is : %.2f" % recall)
f1_score = f1_score(rounded_labels, y_pred ,
    average="macro")
print("The F1_score is : %.2f" % f1_score)
```