Sentiment Analysis on Rotten Tomatoes reviews using Convolutional Neural Network Classifier

Sahand Asri
Department of Computer Science
Lakehead University
Thunder Bay, Canada
sasri@lakeheadu.ca

***Abstract*—In this article, we are introducing and designing a Convolutional Neural Network (CNN) for the goal of text classification. The dataset we are working on is Sentiment Analysis of the reviews of Rotten Tomato website. At the start we explain Sentiment Analysis, Text classification and Convolutional Neural Networks. Then, we introduce our classification model. Finally, we evaluate the our model using different performance metrics such as Accuracy, Precision, Reacall, and F1-score.**

***Index Terms*—Natural Language Processing, Text Classification, Sentiment Analysis, Convolutional Neural Network**

I. Introduction

Sentiment Analysis

Sentiment Analysis or opinion mining is one of the trending Natural Language Processing (NLP) research areas that studies people's opinions and ideas using the speech or text input. Regularly, there are several predefined labels (sentiment tags) available for each input record. The task of the Sentiment Analysis Model is to identify and select the most probable one for each entry. Therefore, Sentiment Analysis is often regarded as a classification problem.

Text Classification

For Sentiment Analysis using text inputs, firstly, one should divide the raw text into smaller chunks called tokens. Then these tokens should be preprocessed for the next step. The preprocessing step modifies the input entries in a way that is more understandable by the classification model. Also, The preprocessing assists the model to remove irrelevant input data, which could complicate the classification model. After the preprocessing step, the processed data and the associated label will be given to the model. Firstly, the model tries to recognize the connection between the input data and the tag. This step is called Training. After that, the model goes to the Testing Phase. In this state, the model is ready to accept the new input entries with the same structure as training data. After receiving the input entry, the model tries to choose the most relevant tag for it. This prediction could be evaluated using test data.

Convolutional Neural Network

One of the popular classification models for text classification tasks is Convolutional Neural Network (CNN). Convolutional Neural Networks were initially invented for computer vision tasks. However, they have shown a remarkable performance on other tasks such as text classification as well.

There are various architectures for CNN systems. However, there are some similarities between them in the primary elements. For instance, in LeNet-5, there are mainly three types of layers which are convolutional layer, pooling layer, and fully-connected layer.

In the following sections, we explain the basic components of a Convolutional Neural Network. After that, we introduce the dataset which we are working on. Then, we present our proposed CNN architecture for the text classification task. In the end, we discuss the performance of our proposed model with different models and parameters.

1) Convolutional Layer: The convolutional layer is the first layer in a CNN architecture. This layer receives the raw inputs from the user and aims to extract some features from them. A Convolutional layer may contain several convolutional kernels. These convolutional kernels are used for computing the feature maps. In this layer, first, inputs convolve with convolutional kernels. Then, the convolved results go to the non-linear activation functions. Typical activation functions are ReLU, sigmoid and tanh.

2) Pooling Layer: The pooling layer usually goes between two convolutional layers. The task of the pooling layer is to reduce the resolution of the feature maps, generated by the previous convolutional layer. By decreasing the resolution of the feature maps, CNN will achieve shift-invariance. In other words, by applying pooling layers, convolutional layers in CNN could better extract different features.

3) Fully-Connected Layer: After several convolutional and pooling layers, we may face one or more fully-connected layers. Now, we extract and gather the high-level features which we need for this layer. Fully-connected layer connects the previous nodes to every one of the neurons in the current layer.

4) Output Layer: The last layer of a classic CNN model is the output layer. Output layers could be different based on the system's application. For instance, for a classification

problem, output layers mostly perform softmax operators. In addition to that, the output layer may use SVM. The loss of CNN is calculated in this stage. There are many types of loss functions used to calculate the performance of the system's prediction. CNN finds the best and optimum parameters by reducing the loss calculated by the loss function in the last layer. Gradient descent is often used for network optimization.

Rotten Tomatoes Dataset

Rotten Tomatoes Movie Reviews is a Sentiment Analysis Dataset for movie reviews. In this dataset, we have 156060 entries with four columns, namely: PhraseID, SentenceID, Phrase, and Sentiment. The dataset is offered for multi-class classification since there are five different classes which are: negative, somewhat negative, neutral, somewhat positive, and positive.

The classification problem for this dataset is quite challenging since instead of each review, every phrase has a sentiment label. Moreover, there are plenty of language ambiguities, sarcasm, and sentence negations, which could make it complicated for the learning model to understand it.

Data Exploration: Below is a table of first ten entries of the training dataset of the Rotten Tomato Movie Reviews.

TABLE I
FIRST TEN ROWS OF THE TRAINING DATASET

PhraseId	SentenceId	Phrase	Sentiment
1	1	A series of escapades demo ...	1
2	1	A series of escapades demo ...	2
3	1	A series	2
4	1	A	2
5	1	series	2
6	1	of escapades demonstrating ...	2
7	1	of	2
8	1	escapades demonstrating th ...	2
9	1	escapades	2
10	1	demonstrating demonstrati ...	2

One of the important things about this dataset is that the label distribution is not balanced. We need to balance the training dataset using upsampling or downsampling methods before giving the data to model.

It is critical that we only perform normalization balance on training dataset and not the test dataset, since we have no information of the testing dataset and it should not have any effect on our proposed model. In Figure 1, the sentiment distribution of the training dataset is shown.

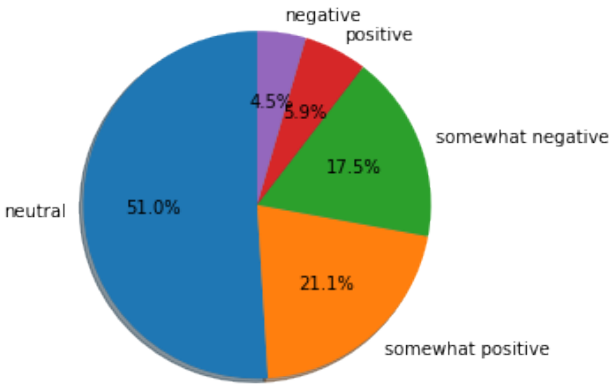

Fig. 1. Class distribution of the training set

II. Background

Convolutional Neural network (CNN) is one of the most popular Deep Neural Network (DNN) architectures based on natural visual perception of the living animals [1].

In [2], Hubel and Wiesel explained that some cells in the visual cortex of animals such as cats and monkies are responsible for detecting light in perceptive fields. Based on this discovery, Kunihiko Fukushima offered a self-organizing neural network model named neocognitron [3]. The proposed model is a neural network attempting to mimic the hierarchical and compositional characteristics of the visual cortex discovered by the previous paper. Neocognition is often regarded as the predecessor of convolutional neural networks. At that time, training phased, which consists of weight optimization and backpropagation, has not been applied to neural networks. Therefore, there was no method to optimize the neocognition model to use in different applications.

Later on, Alex Waibel introduced Time Delay Neural Network (TDNN) in 1989 [4]. This neural network was the first model that applied global weight optimization on the neocognition neural network model. In 1998, LeCun published a paper titled Gradient-based learning applied to document recognition. In this paper, they established modern CNN by developing a multi-layer artificial intelligence neural network named LeNet-5. They used this architecture for handwritten digit classification. This model was the first convolutional neural network that performed on a real industrial application. The strength of this model was that it accepted raw pixels (inputs) without any preprocessing steps to recognize the visual patterns of handwritten digits.

After that, various architectures and methods were offered for a deep CNN. The most notable one was proposed by Krizhevsky et al., which was developed for an image classification contest on ImageNet in 2010 [5]. This model is often referred to as AlexNet. AlexNet was using a deep CNN architecture with significant enhancements upon the previous architectures. AlexNet was designed to classify 1.2 million images into 1000 different classes. For top-1 and top-5, they achieved the error rates of 37.5% and 17% [5]. After the successful attempt of AlexNet, many CNN architectures were offered to improve the performance more. VGGNet [6], GoogleNet [7], ResNet [8] are some of the well-known ones.

III. Proposed Model

In this assignment, we divide it into three different stages, namely: Preprocessing, Data Balancing, Classifier. After loading the dataset, we split the training and testing dataset using the below code. Then, both training and test dataset

where went to preprocessing step.

```
train_test_split(data['Phrase'], data['Sentiment'
    ],
    test_size=0.3, random_state=2003)
```

A. Preprocessing

In this stage, we used several methods to clean and filter our input data.

1) Punctuation and Digit Removal: First, we removed all of the punctuations and numbers using the bellow code.

```
def remove_punctuations_and_numbers(review):
  # Remove Punctuations and numbers
  filtered_review = re.sub('[^a-zA-Z]',' ',review)
  return filtered_review
```

Second, we removed the stopwords using NLTK library. The bellow code is used for this reason.

```
def remove_stopwords(review):
  # Remove Stopwords
  word_tokens = word_tokenize(review)
  filtered_review = [w for w in word_tokens if not
       w in stop_words]
  filtered_review = ' '.join(filtered_review)
  return filtered_review
```

Finally, we used NLTK default tokenizer to split the input entries into seperate tokens. Then, we used NLTK lemmatizer named WordNetLemmatizer to only gather the core parts of the token. We used below code for that.

```
def lematize(review):
  filtered_review = [lemma.lemmatize(w) for w in
      word_tokenize(str(review).lower())]
  filtered_review = ' '.join(filtered_review)
  return filtered_review
```

B. Data Balancing

In this stage we normalize our training data distribution by both downsampling and upsampling. We will review and compare the performance of the model using both of approaches. Below is a sample code for data balancing using downsampling.

```
def downsampling(train_0, train_1, train_2,
    train_3, train_4):

  # returns the number of entries of the smallest
      class
  MIN_SAMPLES = min(len(train_0), len(train_1),
      len(train_2), len(train_3), len(train_4))

  train_0_sample = resample(train_0, replace=True,
       n_samples=MIN_SAMPLES, random_state=2003)
  train_1_sample = resample(train_1, replace=True,
       n_samples=MIN_SAMPLES, random_state=2003)
  train_2_sample = resample(train_2, replace=True,
       n_samples=MIN_SAMPLES, random_state=2003)
  train_3_sample = resample(train_3, replace=True,
       n_samples=MIN_SAMPLES, random_state=2003)
  train_4_sample = resample(train_4, replace=True,
       n_samples=MIN_SAMPLES, random_state=2003)

  df_normalized = pd.concat([train_0_sample,
      train_1_sample, train_2_sample,
      train_3_sample, train_4_sample])
  return df_normalized
```

C. Classifier

We are using Keras to implement our classification model.

First, we'll pass in words to an embedding layer. We need an embedding layer because we have thousands of words, so we'll need a more efficient representation for our input data than one-hot encoded vectors. In this case, the embedding layer is for dimensionality reduction, rather than for learning semantic representations.

First of all, the input words will go to the embedding layer. This layer is used to reduce the dimensions of the input data. In this way, we aid our model to understand the input words better and faster. For Embedding Layer arguments, which are input_dim, output_dim, and input_length, we pass MAX_FEATURES, 150, and 3MAX_WORDSrespectively.

Secondly, for the CNN part, we add SpatialDropout1D with a rate of 0.5. Then, we add Conv1D with 128 filters and a kernel size of 3. We want the output size of this layer to remain the same length as input. Using padding="same" will give us this option. Therefore, there is no need to get involved with the number of inputs and outputs later on in the model. Finally, we are using ReLU as the activation function of this layer.

For the Pooling Layer, we are using a MaxPooling1D with the pooling size of 2.

Then, we add the Flatten Layer to transform our pooled feature map to a line of multiple neurons. After that, we have our features on a single column. Therefore, we are ready to make a fully connected neural network layer to make the classification.

Finally, the output layer has five neurons in the end, and we are using softmax as the activation function since we are dealing with a classification problem.

The source code for designing our proposed model using Keras is provided below.

Listing 1. Our Proposed Model

```
model = Sequential()

# Input Layer
model.add(Embedding(MAX_FEATURES, 150,
    input_length=MAX_WORDS))

# CNN
model.add(SpatialDropout1D(0.5))

model.add(Conv1D(128, kernel_size=3, padding='same
    ', activation='relu'))
model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())

# Output layer
model.add(Dense(CLASS_NUM, activation='softmax'))
```

It should be noted that for the loss function we are using Categorical Crossentropy since we have a categorical (one-hot encoded) output. In addition, we are using "ndram" as our optimizer.

IV. Results

We used different approaches and models to receive the best performance. The best performance we reach for this assignment has the accuracy of around 74.80% using upsampling for balancing the data and 20 number of epochs to train the data. The precision, recall and F1-score for this model is 0.77, 0.72, 0.74 respectively. While increased the number of epochs to 10, we get a higher performance on the training phase. But, in the testing phase the performance of model dropped because it was over-fitted. The accuracy and loss of the best model during the training phase is shown in the Fig. 2.

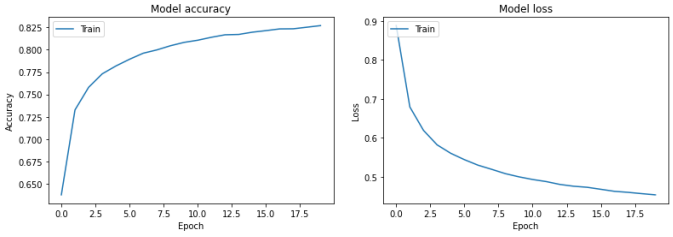

Fig. 2. The accuracy and loss of our proposed model during training phase

Using downsampling we only could achieve 57% and 54% for 5 and 10 epochs. However, they they train much faster the first one since the model is dealing with a quite smaller training set.

Also, we are using 25% of our training set as the validation dataset. This will help our model to avoid over-fitting which results in better performance when we evaluate the performance of our model using test data since the model does not get too complicated.

V. Conclusion

The high performance of our proposed classifier clearly show that in addition to image classification, Convolutional Neural Networks are perfectly suited for text classification. In addition, we can evidently see that increasing the number of epochs does not guarantee us to reach a better performance as we could see our performance increased while we increased the number of epochs. However, the performance of our model dropped in the testing phase since the model was too complicated and over-fitted.

References

[1] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai *et al.*, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018.

[2] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[3] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[4] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.