SENTIMENT ANALYSIS & TRANSFER LEARNING

Baoshuang Zhang
Saarland University
Saarbrücken, Germany
bazh00001@stud.uni-saarland.de
Matriculation Number:7011241

Divya Nidadavolu
Saarland University
Saarbrücken
s8dinida@stud.uni-saarland.de
Matriculation Number:2581339

March 31, 2021

ABSTRACT

In this project, we have created a neural sentiment classifier from scratch. The project is essentially divided into 3 tasks. In "Task 1", we created our own word embeddings from the HASOC Hindi sentiment dataset and train a neural network to extract required data. In "Task 2", we applied the knowledge of transfer learning by using the model from Task 1 to train Bengali dataset. Further, we use this trained classifier to predict hate speech on the Bengali dataset. In "Task 3", we researched about different methodologies which would possibly improve our results and further working on our accuracy scores.

1 Task 1: Word Embeddings

Introduction: Word embeddings are the texts converted into numbers. Word Embedding is used for mapping words to vectors of real numbers. Using a dictionary, a Word Embedding format attempts to map a word to a variable. Word2Vec - In simple terms, we'd train a single hidden layer neural network to perform a specific task (fake task). Rather, the aim is to find out what the weights of the "word vectors" in the hidden layer are. We train a model on text data by converting the text data into some form of numerical representation.

Methodology:

- Get the Data : To begin, we were given the task of using the HASOC Hindi sentiment dataset and training a neural network to extract word embeddings for the data. The research project "Hate Speech and Offensive Content Identification in Indo-European Languages" examines the detection of hate or offensive content in English posts on Facebook and Twitter.
  To get a development set, we split the training set into two parts. So, we randomly choosed 116 data from the data set as the development set for tuning our parameters.

- Data preparation : We applied the following text pre-processing to the posts that are input to our systems: tokenization, the removal of stopwords, lowercasing (to avoid case sensitive issue), removing hashtags and emojis with its CLDR version Though emojis play an important role in representing an emotion or sentiment.But" hate" is a kind of strong emotion, we assume that it can be express clearly without out emojis. Also, removal of them can decrease the computing pressure.
  We append the data to a text list after it has been pre-processed. We generate the initial input by scanning our entire text document and appending the data, which we can then convert into a matrix form. Finally, we need to split the articles into sentences and extract each word from those sentences.

- Build the vocabulary : We cannot feed a word just as a text string to a neural network, so we need to build a vocabulary of words from our training documents.
  At this point, we would have transformed the noisy text dataset into a flat simple one. Further, we need to

transform it into floating-point tensors.

- Subsampling : Subsampling frequent words would help decrease the number of training examples. Word2Vec implements a "subsampling" scheme to address the problems with common words. Hence, for each word we encounter in our training text, there is a chance that we will effectively delete it from the text. The probability that we cut the word is related to the word's frequency.
  - The word2vec code implements an equation for calculating a likelihood with which to keep a given word in the vocabulary.

  $$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot \frac{0.001}{z(w_i)}$$

  P(w_i) is the probability of keep the word w_i, where z(w_i) is the fraction of the total words in the corpus that are that word.

- Skip-grams : Given the current word, Skip gram predicts the surrounding context words within specific window.
  The input layer contains the current word (center word), the hidden layer contains the number of dimensions in which we want to represent current word and the final output layer contains the context words (predictions). Consider an array of words W, if W(i) is the input (center word), then W(i-2), W(i-1), W(i+1), and W(i+2) are the context words, if the sliding window size is 2.
  This function will build a vocabulary from the words found in the training file as well as a hash table which allows for fast lookup from the word string to the corresponding vocabulary word. Words that occur less than the minimum count will be filtered out of vocabulary.
  A word embedding or word vector is a vector representation of an input that captures the meaning of that word in a semantic vector space. The basic idea of word embedding is words that occur in similar context tend to be closer to each other in vector space. For generating word vectors in Python, modules like `nltk` and `gensim` are required.

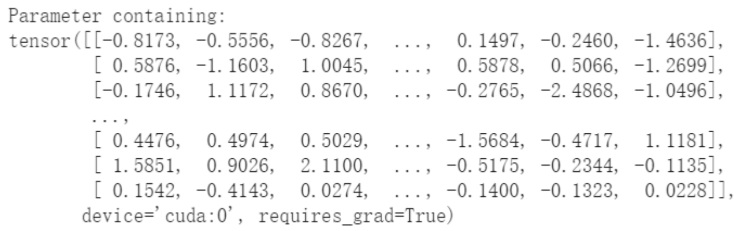
- Hyperparameters : - We have set the values of Window size as 2 and Embedding size as 64.Actually, the embedding size is quite important. Actually, at first we set it just 5 to simplify the computation. However, the accuracy of Hindi when using CNN(see more in Task 2) is just around 0.67.Then we change it to 64. The accuracy is improved significantly. Also, 64 is very useful when we use transformer(see more in Task 3). In conclusion, how to choose this hyperparameter will effect quite a lot on the behavior of classifier.

- Pytorch Module : In this module, we use the derived word vectors with the `nn.Embedding layer`, and `nn.Linear layer`.
  For implementation, we initialized the two weight matrices of word2vec as fields of the class. Later on, we override the `forward` method which takes input a one-hot encoding to perform matrix multiplications. At the end, we apply a log softmax as our output layer.
  In the final stage, we create the iterators which would over the data in training phase. The output is a batch of examples which are indexed and converted into tensors at every iteration.
- Loss function and optimizer : The optimizer `torch.optim.SGD(net.parameters(), lr = learning_rate)` implements stochastic gradient descent. Here, the parameters will be updated by the optimizer and is defined by the first argument. The second argument tells us about the learning rate.
  Our criterion function (loss function) would calculate the loss but we required a function to calculate the accuracy, hence, we use the function `binary_accuracy`. This function initially feeds the predictions (context words) through a sigmoid layer, all the values are then squashed between 0 and 1. For instance, if we have 7/10 right predictions, the value returned would 0.7, and not the number 7. Then, we round these values to their nearest integer. If the value is greater than 0.5 it will round itself to 1 indicating a negative sentiment and similarly, to 0 if the value is less than 0.5 indicating a no negative sentiment. Then we compare the output with the true label, we can calculate the accuracy.
- Training the model : At this stage, decide which word combinations represent phrases. In this step, we go back through the training text again, and evaluate whether each word combination should be turned into a phrase. When training this network on word pairs, the input is a one-hot vector representing the input word and the training output is also a one-hot vector representing the output word. But when we evaluate the trained

network on an input word, the output vector will actually be a probability distribution (that is, a bunch of floating point values, not a one-hot vector). We are going to train the neural network to do the following:
- Given a specific word in the middle of a sentence (the input word), look at the words nearby and pick one at random.
- The network will tell us the likelihood of each word in our vocabulary appearing in the window size parameter we specify.
- The output probabilities will represent the likelihood of each vocabulary word being found near our input word. We'll feed word pairs from our training documents to the neural network to train it.
- The network can learn required statistics based on the number of times each pairing appears.

- Train on the full dataset : Due to lack of technical capabilities for computing, we only trained half of the entire dataset and save the embedding result in to a file .

Results: The unique words in our vocabulary is 11,137. And the weight of the matrix is in the figure. Also the file of embedding result is upload.



```
Parameter containing:
tensor([[-0.8173, -0.5556, -0.8267,  ...,  0.1497, -0.2460, -1.4636],
        [ 0.5876, -1.1603,  1.0045,  ...,  0.5878,  0.5066, -1.2699],
        [-0.1746,  1.1172,  0.8670,  ..., -0.2765, -2.4868, -1.0496],
        ...,
        [ 0.4476,  0.4974,  0.5029,  ..., -1.5684, -0.4717,  1.1181],
        [ 1.5851,  0.9026,  2.1100,  ..., -0.5175, -0.2344, -0.1135],
        [ 0.1542, -0.4143,  0.0274,  ..., -0.1400, -0.1323,  0.0228]],
       device='cuda:0', requires_grad=True)
```

Figure 1: Word embedding result

Conclusion: We have implemented word embeddings from scratch using Word2vec. Given the HASOC Hindi sentiment dataset, we cleaned the data and train the neural network to extract the word embeddings.And if two words have similar meaning, the vectors of them will be similar.

2 Task 2: Sentiment Classifier & Transfer Learning

Introduction Over the past years, sentiment analysis has nurtured and grown into an essential analytic tool for companies, especially for the social media platforms like Twitter, Facebook, Instagram and more. In the recent developments, deep neural architectures like CNN (Convolutional Neural Network) , RNN (Recurrent Neural Network) or Transformer,etc.for sentiment classification have been established.

The aim of sub-task A is to classify short-posts into two classes: "Hate and Offensive" (HOF) and "Non-Offensive" (NOF). It is a coarse-grained binary classification in which the system has to classify tweets into classes, namely, HOF and NOF.
-> (NOT) Non Hate-Offensive - Does not contain any Hate Speech, offensive content.
-> (HOF) Hate and Offensive - Contains Hate, offensive and profane content.

Methodology

- Binary neural sentiment classifier : We have implemented a binary neural sentiment classifier for the Hindi section of the corpus. We utilise the word embeddings from Task 1.
  We use just one convolutional layer to extract the sentiment of different sentences. . First, we will pad each sentence,using zero to make all sentence have the same length,same size,like the images. Then we use three different filters or kernels which scan across the sentence. The main advantage of the convolutional layers is that the weight are learned via the backpropagation. The intuitive idea is that for extracting parts of the sentences, the convolutional layers act as feature extractors by learning the weights. And comparing with RNN, it can calculate parallel.
  We implement the convolutional layers with `nn.Conv2d`. Since we are using text, we only have a single channel which is the text itself. Hence, the `in_channels` argument which specifies number of channels in the image are going into the convolutional layer. The number of filters is given by argument `out_channels` and

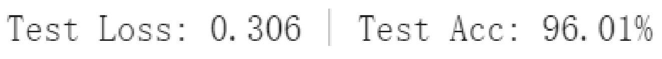
kernel_size gives us information about the size of the filters. Each of the kernel_size will be calculated as [n * emb_dim], where n determines the size of the n-grams.
After passing the sentence through an embedding layer to get our embeddings, we unsqueeze our tensor to create a channel dimension, as a text does not have one. This later on matches with the initialization values of in_channels=1. We then create an instance for our CNN class. The dimensions of the one-hot vectors will be the input dimension which is also equal to the size of the vocabulary.
We save the weights of CNN model to a file"CNNweight"(uploaded),for transfer learning.we trained the data 50 times and calculate the first accuracy number of our whole project.

- Pre-process the Bengali data: We applied the same model in Task 1 to pre-process the Bengali data. One important thing is to split off a part of the Bengali corpus such that it roughly equals the Hindi corpus in size and distribution of classes(hatespeech/non-hatespeech).This will apply classifier result of Hindi better to Bengali data.

- Bengali word embeddings Through the embedding layer, the input batch is feed to get a dense vector representation of our sentences.
  Also, because of the time and technical capabilities for computing, we just train 1100 data.
- Apply classifier to Bengali data: We then implemented the result of Hindi to Bengali data without training it and obtain the accuracy.

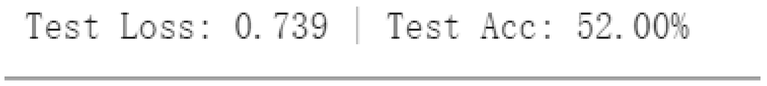
- Retrain Bengali data: using CNN trains the data and we can know the exact accuracy.We trained 20 times. One batch at a time is iterated at a time while training process. For every batch, we initialized the values of gradients as 0. For ever parameter, we can see the value of the gradient which is calculated by criterion (loss function). Later, we feed the batch of sentences into the model to get the batch size. The loss can be calculated using our predictions, similarly for accuracy, can be derived from the labels with the loss being average all across the batch.

Results We can see from the following figures, the first number is 0.96. However, transfer learning result is 0.52. Then retraining the data, the accuracy improve to almost 0.8.



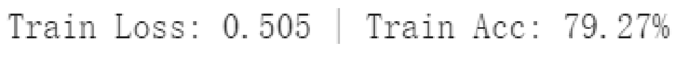
```
Test Loss: 0.306 | Test Acc: 96.01%
```

Figure 2: CNN accuracy for Hindi



```
Test Loss: 0.739 | Test Acc: 52.00%
```

Figure 3: transfer accuracy for Bengali



```
Train Loss: 0.505 | Train Acc: 79.27%
```

Figure 4: Retrain accuracy for Bengali

Conclusion In this task, we used a Convolutional Neural Network to train both the obtained datasets. We have performed similar operations to clean the Bengali datasets and further process the clean data to obtain word embeddings. We have implemented the knowledge acquired from "Transfer Learning" by using the same model from previous task to train the Bengali dataset.And the result shows that CNN model brings relatively high accuracy. Even the result of transfer learning is beyond 0.5.

3 Task 3: Challenge Task

Introduction In this task, a Transformer structure is used to improve the three accuracy scores in task 2.Transformer structure is a kind of "Encoder-Decoder"[4]. But, we just use "Encoder"part to implement the sentiment classifier. Also, in this task, we just want to verify its effect and compare with CNN, so we used part of the code in this article[5]. The reason why using Transformer model to retrain the data is following:

- Single convolution layer cannot capture long-distance features. That means, using CNN, the ability of obtaining the correlation between different features is limited. As in task 2, the sizes of three filters we used are 2,3,4. But the length of one sentence is quite long(beyond 50). So,we can not know the relationship between features beyond the filters. Though we can increase the number of convolution layers to capture further information.But this problem is not well solved.
- The base structure of transformer decide that, when one sentence is inputted, the current word has a connection with any word in the sentence.It is done in one step and directly.So,the meaning of the whole sentence will be understand more accurate comparing with CNN. Also, like CNN, this model can compute the data parallel. So, transformer can extract much more information without Without sacrificing calculation speed. Under this circumstance, it can improve the accuracy and guarantee the speed, theoretically.[6]

Methodology The structure we use is showed in Figure 1. The same as CNN, we need to input fixed-length sentences.

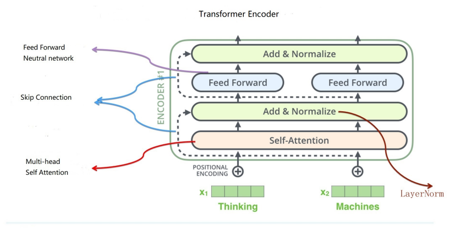


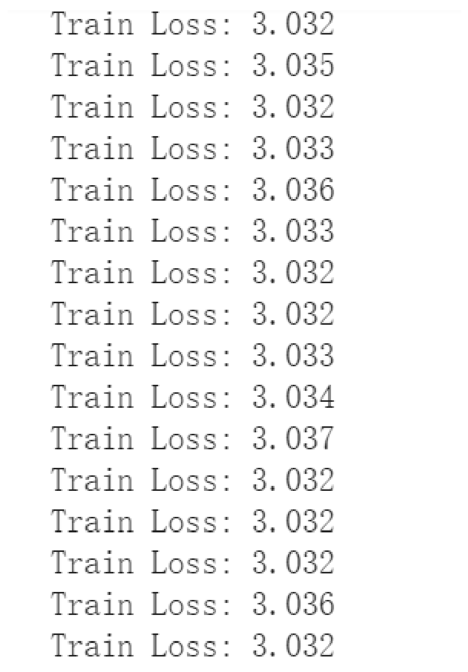
Figure 5: structure of the model

However, we have to add position information to each word in a sentence. Since the order of every word in a sentence pays a very important role for the sentiment(negative or not).Actually, in the CNN method, position information of every word is retained relatively.

Results We retrained each data set.The results for two accuracy is showed.

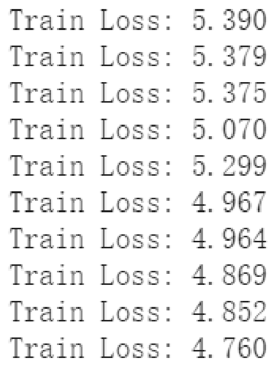
Conclusion As shown in the results, the loss is not smaller than the numbers in CNN,even bigger than the model used in Task 2.It means the accuracy is not improved.Also, it is no doubt that the Transformer model is much more complex than the CNN model used in Task 2. In our opinion,the reason maybe the following:

- The aim of the task:the role of a model in our code, no matter CNN or Transformer,is to extract features of our corpus,of course,after embedded.But,considering the tasks, it does not mean that,the more features, the better. From the analysis of the model structure, transformer can calculate more than CNN and is more comprehensive. However, the aim of our tasks is to decide whether the sentence is "hate-speech".It is a kind of quite strong sentiment Certain part of the sentence which means the information within one "filter",will make the main role. Though "attention" is a kind of mechanism to care more about certain words in one sentence, in this case, it is too comprehensive. So the result is not ideal.
- Distance of words:Actually,in one sentence, the longer distance between two words, the smaller relationship they have.Especially, for this task, we do not need to understand the whole meaning of a sentence. So, maybe CNN has high efficiency and transformer is redundant.
- Size of corpus:since transformer is a kind of strong tool, because of the size of our data,transformers' abilities are not fully utilized. In contrast, CNN is a quite simple model and more suitable for relatively small amount of data. Maybe, for bigger corpus,transformer will behave much better.
- The role of hyper-parameters:Transformer has more hyper-parameters,the setting of hyper-parameters will have a great influence on the result, as stated in Task 2. So,changing them makes better result.



```
Train Loss: 3.032
Train Loss: 3.035
Train Loss: 3.032
Train Loss: 3.033
Train Loss: 3.036
Train Loss: 3.033
Train Loss: 3.032
Train Loss: 3.032
Train Loss: 3.033
Train Loss: 3.034
Train Loss: 3.037
Train Loss: 3.032
Train Loss: 3.032
Train Loss: 3.032
Train Loss: 3.036
Train Loss: 3.032
```

Figure 6: Hindi data



```
Train Loss: 5.390
Train Loss: 5.379
Train Loss: 5.375
Train Loss: 5.070
Train Loss: 5.299
Train Loss: 4.967
Train Loss: 4.964
Train Loss: 4.869
Train Loss: 4.852
Train Loss: 4.760
```

Figure 7: Bengali data

- the problems of model: to be honest, transformer is quite complex. There may be some problems about the model itself we used. But we did not find during the operation process. And it affects the result.
  The link of the whole project is in[7]

References

[1] Geeks for Geeks. In *Implement your own word2vec (skip-gram) model in Python.*

[2] Chris McCormick In *Word2Vec Tutorial - The Skip-gram Model.*

[3] Ben Trevett In *Pytorch Sentiment Analysis.*

[4] Ashish Vaswani,Noam Shazeer In *Attention Is All You Need.*

[5] Alexander Rush In *The Annotated Transformer.*

[6] Jay Alammar In *The Illustrated Transformer.*

[7] link of project In *https://github.com/zhangbssan/NNTI*$_N LP_F inal - project.$