Neural Network Project (NLP) 2021

Zubayr Khalid - 7003003 (zukh00001)
Aditya Bikram Biswas - 2581493 (s8adbisw)

Abstract

This is a final project for the course Neural Networks: Theory and Implementation (NNTI). This project introduced us to Sentiment Classification and Analysis. We have created a neural sentiment classifier completely from scratch. We first train it on HINDI dataset and then apply it to BENGALI dataset. The project was divided into three main tasks mainly :
Task 1 -> Word Embedding
Task 2 -> Sentiment Classifier & Transfer Learning
Task 3 -> Improvement of Existing Result

Introduction

Sentiment analysis refers to the use of natural language processing, text analysis, computational linguistics, and/or bio-metrics to systematically identify, extract, quantify, and study affecting states and subjective information. Sentiment analysis models focus on polarity (positive, negative, neutral) but also on feelings and emotions (angry, happy, sad, etc), urgency (urgent, not urgent) and even intentions (interested v. not interested).

Transfer learning is a Machine Learning research problem that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.

Related Works

In the paper[1], the authors have introduced the Skip-gram model, an efficient method for learning high quality vector representations of words from large amounts of unstructured text data. Using a word offset technique where simple algebraic operations are performed on the word vectors, it was shown for example that vector(”King”) - vector(”Man”) + vector(”Woman”) results in a vector that is closest to the vector representation of the word Queen. Two new Log-linear Models are introduced: Continuous Bag-of-Words Model and Continuous Skip-gram Model.

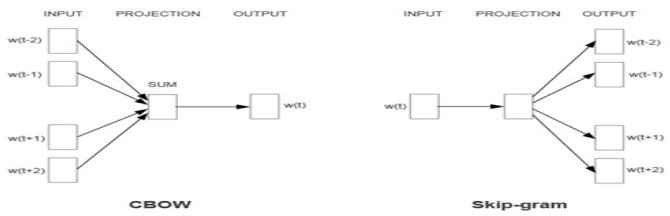

Figure 1: New model architectures. The CBOW architecture and Skip-gram Architecture

Unlike most of the previously used neural network architectures for learning word vectors, training of the Skipgram model does not involve dense matrix multiplications[2]. The authors have first identified a large number of phrases using a data-driven approach, and then treated the phrases as individual tokens during the training. Hierarchical Softmax function is used as a computationally efficient approximation of full softmax. Negative sampling is defined as a substitute of NCE, where NCE uses both samples and the numerical probabilities of the noise distribution, while Negative sampling uses only samples. Since the global corpus statistics are captured directly by the model, GloVe, a new model for word representation[3] is proposed. Noted that that the complexity of the model is much better than the worst case O(V2), and in fact it does somewhat better than the on-line window-based methods which scale like O(|C|).

Task - 1

The summary of Task 1 was to train a neural word embeddings like Word2Vec model[5]. A word embedding or word vector is a vector representation of an input word that captures the meaning of that word in a semantic vector space.

We first start with the data preparation for our model by removal of usernames(starting with @) and stopwords given to us in a separate file. We removed the special characters along with punctuation from the corpus by replacing them with space, thus separating each word in the corpus. Also, we removed the hashtag symbol('#'), but keeping the word associated with the symbol. Moreover, the emojis were not removed as they express emotions and would help to identify a hate speech or happy speech or funny speech. After removal of stopwords and punctuation we split the remaining words in the corpus and converting the corpus to lowercase to maintain uniformity.

Now, we build the vocabulary from the corpus. The vocabulary consist the list of unique words from the original wordlist. The size of the vocabulary is 19365 words. The input to the first layer of Word2Vec is an one-hot encoding[6] of the current word corpus. The output of the model is then compared to a numeric class label of the words within the size of the skip-gram window. One hot encoding is a process by which categorical variables are converted into a form that could be provided to Machine Learning algorithms to do a better job in prediction.

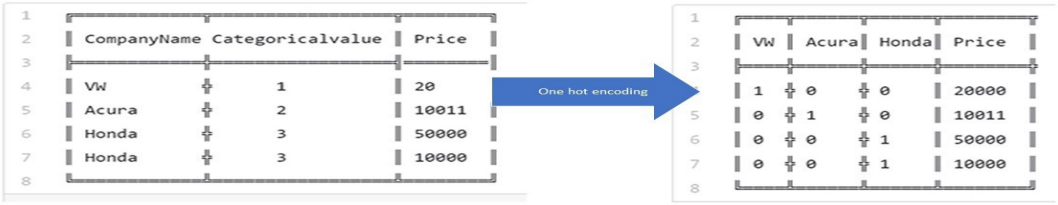

Figure 2: One hot encoding explained in an image

The probability to keep a word in a context is given by:

$$P_{keep}(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

Where $z(w_i)$ is the relative frequency of the word w_i in the corpus. Next, we calculated the frequency of each word and then computed the probability to keep a word in a context.

Now, we start computing the context words list along with the current list words after data preparation and building the vocabulary. We defined the Hyperparameters as independent variables and passed the values in the defined functions.

window size = 4 # embedding size = 300 # learning rate = 0.001 # epochs = 50

According to the paper[5] a typical window size is 5, we took 4 as keeping it is not low as 2. We also take the size of embedding size[7] within the range 250 to 350. The next part of the task comes with running the functions serially to get the proper output. Firstly, the data preparation function is executed, then computing the Wordlist and Vocabulary from the building vocabulary function and lastly executed the current contexts function to compute the current wordlist and the context list.

We then create the Word2Vec model and initialize two weights which are linear. We used softmax function as the activation function which lets us convert the output from a Linear layer into a categorical probability distribution. We used logsoftmax function which is basically just a more numerically stable way to compute Log(Softmax(x)).

We defined the optimizer and loss function next. 'ADAM' [4] is considered as a method of Stochastic Optimization which is a technique implementing adaptive learning rate. Whereas in normal SGD the learning rate has an equivalent type of effect for all the weights/parameters of the model. We used 'NLLLoss' as the loss function.

We defined the dataloder with batch size 64 before training the model and keeping a terminating condition when average loss reaches the local maxima.

```
avg loss at epoch  47   tensor(0.9952, grad_fn=<DivBackward0>)
--- 360.64675521850586 seconds ---
avg loss at epoch  48   tensor(0.9943, grad_fn=<DivBackward0>)
--- 367.23879051208496 seconds ---
avg loss at epoch  49   tensor(0.9945, grad_fn=<DivBackward0>)
--- 373.133962392807 seconds ---
Training finished
```

Figure 3: After training finished

After completion of training, we trained on the full dataset after removing the restriction of sentences on the number of sentences in our corpus. Then, we did the same subtasks like preprocessing, initializing the model along with two extra variables initialization for plotting the graphs shown below:

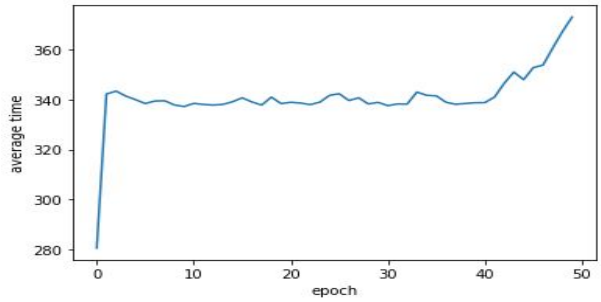

Figure 4: Average Time VS Epoch

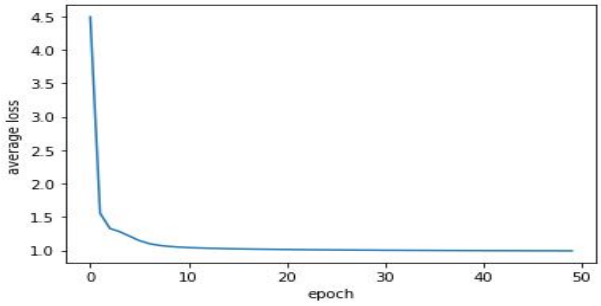

Figure 5: Average Loss VS Epoch

Task - 2

The summary of Task 2 was to create a sentiment classifier and then apply the knowledge of transfer learning by using our model from Task 1 to train the Bengali word embedding model and then used the trained classifier to predict hate speech on the Bengali dataset.

We implemented a binary neural sentiment classifier for the Hindi section of the corpus, using the word embeddings from Task 1. Then we created a custom dataset out of the main dataset so that it

can return the text and hate speech based on some index. The custom dataset was created by padding the indexed sentence to 30 words. For indexed hate speech, we made a tensor(0.0) if its NOT or tensor(1.0) if its is HOF(Hate speech or Offensive). We created the instance of same model we created in Task 1 before creating a new model that we will use for sentiment analysis.

To get a complex Neural Network with many hidden layers, we included some layers with batch norms and dropouts. We have also implemented ReLu(Rectified Linear units) to gain non-linearity. Rectified linear units find applications in computer vision and speech recognition using deep neural nets and computational neuroscience. We define the optimizer the criterion(loss function) and then we train and test the model.

We got the accuracy as 72% after testing the model on Hindi dataset. We saved the model and loaded the model to check whether it was saved correctly.

```
Testing started
avg loss at epoch  tensor(0.0346, grad_fn=<DivBackward0>)
--- 304.32085847854614 seconds ---
accuracy: 0.7159699892818864
Testing finished
```

Figure 6: Accuracy on Hindi Dataset

Now, we start preprocessing the Bengali dataset by removing the usernames, punctuations and Bengali stopwords. Then we split the data and make the list of all words and the vocabulary before creating the current and context list from the dataset.

Next we define the parameters and hyper parameters and then initialize the model to train it on the Bengali data. We plot the graphs to see how the training went just the way we did it in Task 1.

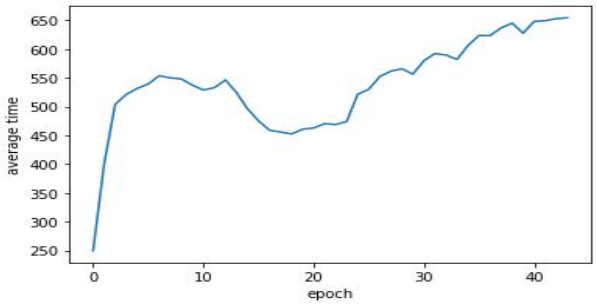

Figure 7: Average Time VS Epoch

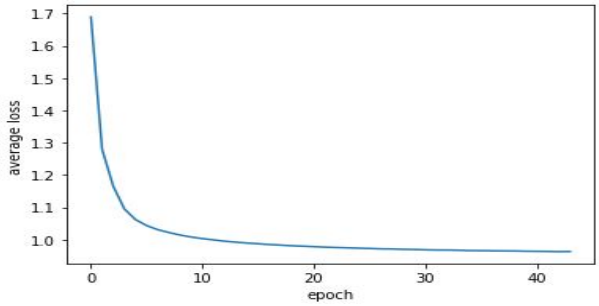

Figure 8: Average Loss VS Epoch

Now, we will apply classifier to Bengali data. First we will test Bengali data directly on the Classification model(the model we trained for the sole purpose of sentiment analysis of Hindi dataset).

We created custom dataset for Bengali data just like we did for Hindi dataset in Task 2a. We initialized the model that we used for embeddings after setting the parameters. After this we load the Hindi

word embedding model saved from Task 1 as it will be needed for the Hindi sentiment analysis class definition. We loaded the Hindi sentiment analysis that we saved in Task 2a after initialization of the class which will be used for sentiment analysis.

After testing with Hindi sentiment analysis model directly on Bengali data without no further training we see the accuracy is 48% which is really not optimistic. So, it states that without re-training the model on Bengali data, the model which is only trained in Hindi data will not give a good sentiment analysis.

```
Testing started
avg loss at epoch  tensor(0.0357, grad_fn=<DivBackward0>)
--- 557.9914381504059 seconds ---
accuracy: 0.48316708229426436
Testing finished
```

Figure 9: Accuracy on Bengali Dataset using Hindi sentiment analysis model directly.

Next we will load the Hindi sentiment analysis model but this time we will train it first with Bengali dataset. After training with the Bengali dataset we get an accuracy of 84% which is better than before.

```
Testing started
avg loss at epoch  tensor(0.1620, grad_fn=<DivBackward0>)
--- 481.49974608421326 seconds ---
accuracy: 0.8379052369077307
Testing finished
```

Figure 10: Accuracy on Bengali Dataset using Hindi sentiment analysis model after training on Bengali Dataset.

After saving the model, we loaded the Bengali word embedding model (saved in Task 2c) and train and test sentiment analysis on Bengali dataset (similar to what we did in Task 2a for Hindi dataset). We have already created the custom dataset and the train and test dataloader. So, we will just start from loading Bengali word embedding model. We define the sentiment analysis class and initialize for Bengali dataset. Before training, we define the optimizer and loss function. To be noted, we have used Bengali word embedding model to train it. Next, we test it on the dataset. After testing the Bengali sentiment analysis model created using Bengali word embedding model we get the accuracy as 83%.

```
Testing started
avg loss at epoch  tensor(0.0177, device='cuda:0', grad_fn=<DivBackward0>)
--- 319.8833885192871 seconds ---
accuracy: 0.8322942643391521
Testing finished
```

Figure 11: Accuracy on Bengali Dataset using Bengali sentiment analysis model.

From the accuracy report, we find that Hindi sentiment analysis model performs better on Bengali data (after it is further trained on Bengali data) than the Bengali sentiment analysis model created from Bengali word embedding model.

We can come to the conclusion that Hindi dataset somehow affects the results on Bengali data. This is the reason why Transfer Learning from Hindi data to Bengali data is giving better accuracy than the model generated from training on only Bengali data.

We also share the models we computed in this task in the below link :

https://github.com/zubayr1/NLP_Project_2021

Task - 3

The summary of Task 3 is to improve our results from Task 2 and then compute the accuracy to check whether the new methodologies(LSTM - Long Short Term Memory Neural Network) we implement in Task 3 gives us better result. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. Relative insensitivity to gap length gives an advantage to LSTM over alternative RNN(Recurrent Neural Network), hidden Markov models and other sequence learning methods.

We started by creating the instance of Hindi embedding model which was already created in Task 1. To be noted we are not using the Bengali embedding model as we done the training and testing in the fourth part of Task 2. After uploading both the Hindi and Bengali dataset we start with the data preparation phase similar to the data preparation in the previous tasks. We split the Bengali data and reduced it to same size of Hindi Vocabulary size which was computed and used in Task 1 and Task 2.

Data Augmentation

The process of applying simple and complex transformations like flipping or style transfer to the data – can help overcome the increasingly large requirements of Deep Learning models. We implemented data augmentation in Hindi and Bengali dataset so that we can insert bias into training set so that we get low variance in testing set.

We used the method of indexing as we were taking randomised data and reshuffling using method of swapping. We also used a method where a word was replaced with a synonym. Lastly, our data augmentation method calls all the mentioned methods and also creates separate Train set, Validation set and Test set.

After Data Augmentation, we create the wordlist and vocabulary of Hindi dataset and Bengali dataset created in the previous tasks. While creating the custom dataset, we passed index of each word present in the vocabulary instead of passing the One Hot encoding.

Next, we created the training, testing and validation loader for Bengali and Hindi Data before creating the LSTM class for new sentiment analysis model. After testing the model for Hindi dataset we got an accuracy of 77% and after retraining with Bengali data we got an accuracy of 87%

```
avg loss at epoch  tensor(0.0154, grad_fn=<DivBackward0>)
--- 62.70032811164856 seconds ---
accuracy: 0.7687366167023555
Testing finished
```

Figure 12: Accuracy on Hindi Dataset using new sentiment analysis model.

```
avg loss at epoch  tensor(0.0087, grad_fn=<DivBackward0>)
--- 101.91992402076721 seconds ---
accuracy: 0.8689138576779026
Testing finished
```

Figure 13: Accuracy on Bengali Dataset using new sentiment analysis model.

From the given accuracy we can conclude that the new sentiment analysis model created using LSTM gives a better accuracy than the neural network model in Task 2.

	Task2	Task3	Report
Hindi Sentiment Analysis	72	77	INCREASING
Bengali Sentiment Analysis (without training)	48	50	INCREASING
Bengali Sentiment Analysis	84	87	INCREASING

Table 1: Analysis

Conclusion

We have done word embedding on available Hindi and Bengali dataset. The trained embedding models are then used for Sentiment Analysis on the same Hindi and Bengali data. In our work we have done sentiment analysis on Hindi data using Hindi word embedding model. For this purpose, we have trained and tested a Hindi Sentiment Analysis model. We have further trained and tested the same model on Bengali data to find how transfer learning is working. We also have trained a Bengali Sentiment analysis model using Bengali word embedding model, without any influence of the Hindi data (except for the fact that Bengali word embedding model is generated using Hindi word embedding model). In our observation we have found that sentiment analysis model trained on both Hindi and Bengali data is more effective than sentiment analysis model trained only on Bengali data. In the final task we have achieved higher accuracy for Sentiment Analysis on both Hindi and Bengali data. For future works, we recommend trying further transfer learning on the final two models, to observe if a new model generated from training on the both datasets gives better accuracy than the both models, which in fact are trained on a single dataset separately.

References

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

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems, 26:3111–3119, 2013.

[3] Jeffrey Pennington, 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), pages 1532–1543, 2014.

[4] https://medium.com/@Biboswan98/optim-adam-vs-optim-sgd-lets-dive-in-8dbf1890fbdc

[5] http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

[6] https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f

[7] https://datascience.stackexchange.com/questions/51404/word2vec-how-to-choose-the-embedding-size-parameter

[8] https://github.com/zubayr1/zk_packages

[9] https://github.com/zubayr1/NLP_Project_2021