Text Classification using Deep Neural Networks.

Keerthi Kandadi (KK22M), Sri Vennala Kandibedala (SK22BV)
Surendra Reddy Goddali (SG22BP), Vishak Udupa(VU22)

1 Preface

Our first goal was to build a python library for text classification that would abstract all the inner workings of training a neural network. But, as we are submitting it as a jupyter notebook and also the effort required to build a library in python does not relate to data mining. We are simplifying our problem statement to 'Text Classification using Deep Neural Networks.

2 Introduction

Text classification is one of the important topics that help organize and gather insights from text data. It is used in a wide range of applications such as topic labeling, sentiment analysis, spam detection, etc. And as the amount of text data that is collected explodes the requirement for classification of those data also increases. And one of the ways to do text classification is by using machine learning algorithms. The machine learning algorithms that can be applied are of two types, the first being the traditional methods such as K-Nearest Neighbor, SVM, Decision Trees, etc. And the second class is Deep Learning methods such as LSTM, CNN, and Transformers. After 2011 Deep learning methods dominated this field. Even though after extracting the features traditional methods might work as well as deep learning models, feature extraction is not an easy task. Hence, recently people have shifted to deep neural networks as it abstracts out the feature extraction part. In this project, we intend to build 'Continuous Bag of Words' [1] and 'Continuous Skip Gram model' [1] to learn the word embeddings, and after that, we have built 3 neural network models namely, Recurrent Neural Networks, Gates Recurrent Unit, and Long Short Term Memory Neural Network to classify the text data. And lastly, we worked on model interpretability to understand how the models classify the text data.

3 Literature Survey

Traditionally, text classification was done in two parts. 1. Feature Engineering, 2. Classification. Feature Engineering can be as simple as assigning weights to words based on their occurrence or as complicated as predicting the highlights of the text. Using the extracted features then we could classify text using classification techniques like Naive Bayes, SVM [7], Decision Trees [8], etc.
Recently, deep neural networks have taken over this field as they do not require feature engineering which is one of the hardest problems in text classification. From 2011 till 2018 Recurrent Neural Networks [3] led the charge for text classification, as they were able to understand the previous context. In particular, LSTMs [4], that are a type of RNN without vanishing and exploding gradient problems. In 2014 Gated Recurrent Unit Networks, also a type of RNN were proposed. GRU [2] is better on the smaller dataset and was much simpler than LSTM. After 2018, transformers took the lead in text classification. The problem with RNNs was that they had to be processed sequentially, whereas the attention model used in the transformer does not need to be run sequentially. This gave huge performance benefits that led to effective training on huge datasets.

4 Methodology

4.1 Word Embeddings

The very naive way of representing words would be through one-hot vectors. However, this has some serious drawbacks. First of all, the vector would be as huge as the vocabulary which adds to the computational complexity of the system. Secondly, all the words in the dictionary are not equidistant from each other. Some of the words can have a similar meaning. Hence, we use word embeddings. Word embeddings are vectors assigned to each word, such that similar words are closer in distance and the words which are not similar are far apart.
There are two ways of learning these embeddings, first one is through feature learning, where the weights are assigned using traditional methods. And the second one is through language models. These weights can also be learned while running the classification models, as these weights support backpropagation through them. But, these weights would be very specific to the classification model.

4.1.1 Word2Vec

This was one of the initial models proposed to learn word associations using a neural network language model. There are two types of Word2Vec Models

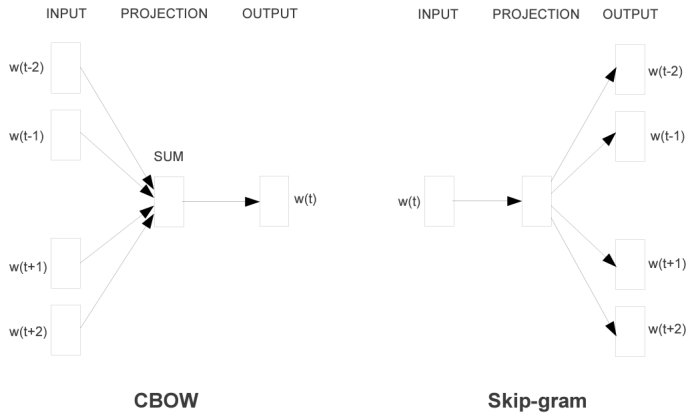

Figure 1 CBOW and Continuous Skip Gram Models.

1. **Continuous Bag of Words Model:** In CBOW we embed n words using an embedding layer, then we pass it through a linear layer, then we sum all the values and apply sigmoid on it. Here we will get the d logits, where d of the size of the dictionary. But the prediction would be only one word. So, based on these logits and predictions, we calculate the loss and backpropagate.

2. **Continuous Skip Gram Model:** In the Continuous Skip Gram model, for each word in the middle of the bag of words, we pass it to the model with the prediction as the other $n-1$ words. Other things are similar to the CBOW model.

4.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are highly suitable for text as they can capture long-range dependencies. RNNs are also location-aware and have historical context. The architecture of a fully connected RNN is to

connect the whole output layer to the input layer. This is done using hidden layers. Here we take the initial hidden layer and combine it with the first input, then pass it through a linear layer and apply activation on it. Then we take this output and combine it with the second input and pass it through the linear layer again. This forms a loop such that the neural network has the previous context as well as the new input. For text classification, the list of inputs is the words in a text sentence.
The main drawback of Fully Connected Neural Networks is vanishing gradients. As these neural networks are very deep, when backpropagating through them gradients would be very small and it would prevent the optimizer from updating the weights. And another problem is exploding gradients, as each input accumulates context from the last input the gradients also accumulate and result in a large update of weights. This is solved by Long Short-Term Memory Networks and Gates Recurrent Unit Networks.

4.3 Long Short Term Memory Networks

To solve the exploding and vanishing gradient we use Long Short Term Memory Networks (LSTM). LSTM has a cell, input gate, output gate, and forget gate. The cell remembers the historical context. Forget gate decides whether to forget the previous information or not. In a sentence, there would be multiple words and some of the words might not be relevant in the current context. In order to filter these words we use an input gate. After passing the hidden layer values and the input through the linear layer, to regulate whether the output of the linear layer is important or not, we use an output gate.

4.4 Gated Recurrent Unit

Gated Recurrent Units (GRU) are also based on RNN and are similar to LSTM in the sense they use gates to regulate the flow of information. However, they work better in the case of smaller datasets. There are two gates in GRU namely update get and reset gate. The update gate controls the flow of previous information into the future. The reset gate decides how much of the historical information to forget. GRUs are simpler than LSTM as they have fewer gates.

4.5 Model Interpretability

Even though our objective is to classify the text data. Understanding how the model classifies the data is critical for debugging and removing biases. So, in order to increase trust, understand causality, and make the model fair, we would need to interpret the model. And we do this using a tool called captum [6].

5 Datasets

5.1 Datasets

We are using 4 datasets to test out our model.

1. **IMDB Dataset** has two columns. The first is the reviews by users and the second is the sentiment of the review.
2. **Spam/Not-Spam Dataset** has two columns, the first being the emails and the second being the boolean indicating whether it is a spam email or not.
3. **Fake Job Dataset** has two columns, the first is the job description and the second column tells whether it is a fake job or not.

4. **Fake News Dataset** has two columns, the first is the description of the news article and the second column indicates whether it is fake news or not.

Each of these datasets has 3 files, the first is the training set which has 80 percent of the records, the second is the test set which has 20 percent of the records and a text file which is a blob of text that has all the words in the dataset.

5.2 Building Vocabulary

We read the text file and tokenized the huge blob of text. Then we create a word frequency map, and then we discard words that are infrequent. In the end, we add special words such as padding to the dictionary. Finally, the vocabulary would have word-to-integer mappings and integer-to-word mappings. As different datasets might have different words, we build different vocabulary for each dataset.

5.3 Word Embeddings

In each neural network, we have a word embedding layer, that maps the integer with which the word is associated to a 300-dimensional vector. This 300-dimensional vector contains weights associated with the word. This can be either learned through language models like CBOW or Skip Gram, or while training the particular classification model. In our case, we first trained the embeddings using CBOW or Skip Gram and then used those trained embeddings in GRU and compared the results. In all the other experiments we are starting with random weight.

6 Implementaion

All the code is implemented in python. We have used PyTorch to implement the neural network models. We have used ADAM optimizer from *torch.optim* for optimization.

6.1 Word2Vec

Both the Word2Vec model have 3 layers. First, is the embedding layer from *torch.nn*. We pass the words through it and get the word embedding. In the case of CBOW, we sum up the embeddings of n words. In Skip Gram, we have represented the data as one-to-one mapping. So, there is no need to sum anything. Then we pass it through a linear layer and apply sigmoid on it. This will give us the logits based on which we calculate the loss and backpropagate.

6.2 Reccurent Neural Network

To get a better understanding of how these neural networks are built, we have built the RNN from linear layers and we did not use the nn.RNN module. Here we embed the words in a sentence, take the embedding of the first word and the initial hidden layer, and concatenate both of them. Then we pass them through a linear layer, this would give us the next hidden layer. We take this hidden layer and concatenate it with the next input word and do the same thing till we reach the end. In the end, we take the output and pass it through the *hidden_to_output* layer and apply the sigmoid function on it. Then calculate the loss and backpropagate.

6.3 Gates Recurrent Unit

Here, we embed the words and pass the list of embeddings to *nn.GRU* module to do the forward pass. Then we take the last output and pass it through the *hidden_to_output* layer and apply the sigmoid function on it. Then calculate the loss and backpropagate.

6.4 Long Short Term Memory Networks

Here as well, we embed the words and pass the list of embeddings to *nn.LSTM* module to do the forward pass. There are a lot of hyperparameters that we can pass, such as whether we want multi-layer LSTM, or whether it should be bi-directional or not. We have kept it simple and not used bi-directional LSTMs.

6.5 Model Interpretabilty

We have used Captum AI for model interpretability. This provides a library to visualize text. Here we need to set the gradients to zero and do a forward pass first. Then using word embeddings and the gradient graph it would interpret the models and report the word importance.

7 Experimentation and Results

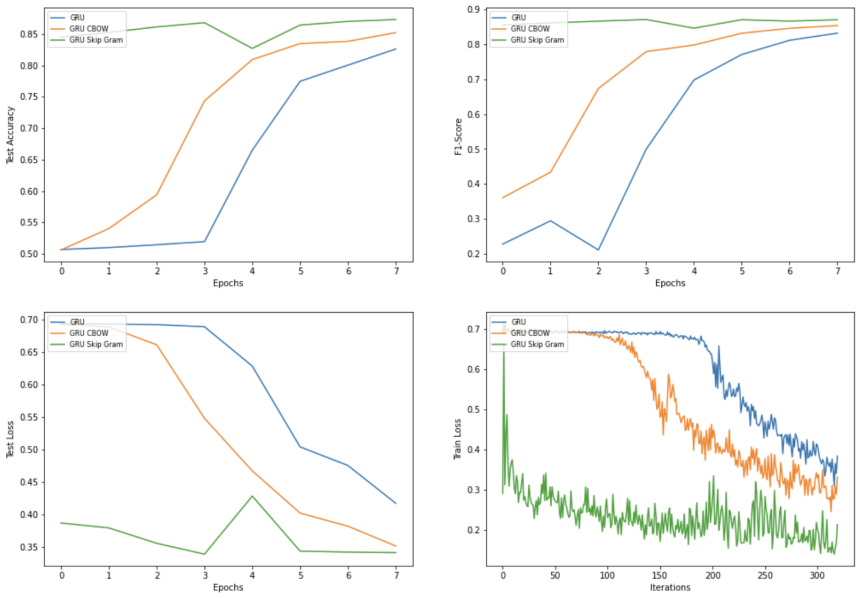

Figure 1 No pre-trained embedding vs CBOW vs Continuous Skip Gram Models for GRU. Here we can see that if the words are pre-trained then the model learns much more quickly. The final accuracy is the same as we learn the language model using the same text. So, we will not encounter any new relation that the classification model cannot learn.

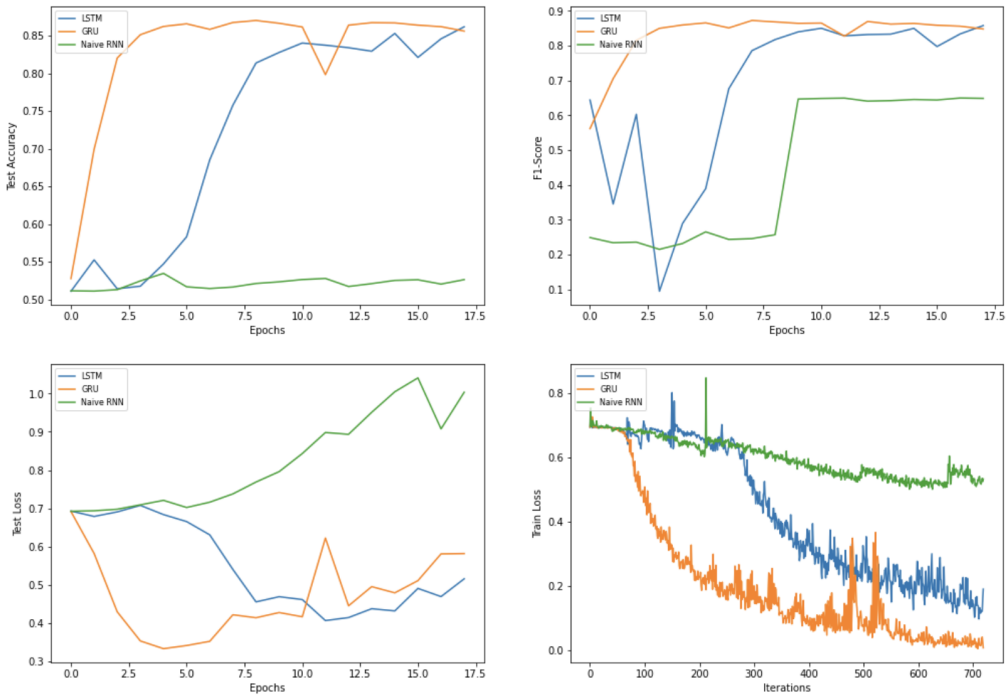

Figure 2 Sentimental analysis for IMDB Dataset. GRU Learns faster than LSTM but has the same accuracy. We are running more epochs than required, so there is some entropy after the model reaches the maximum accuracy.

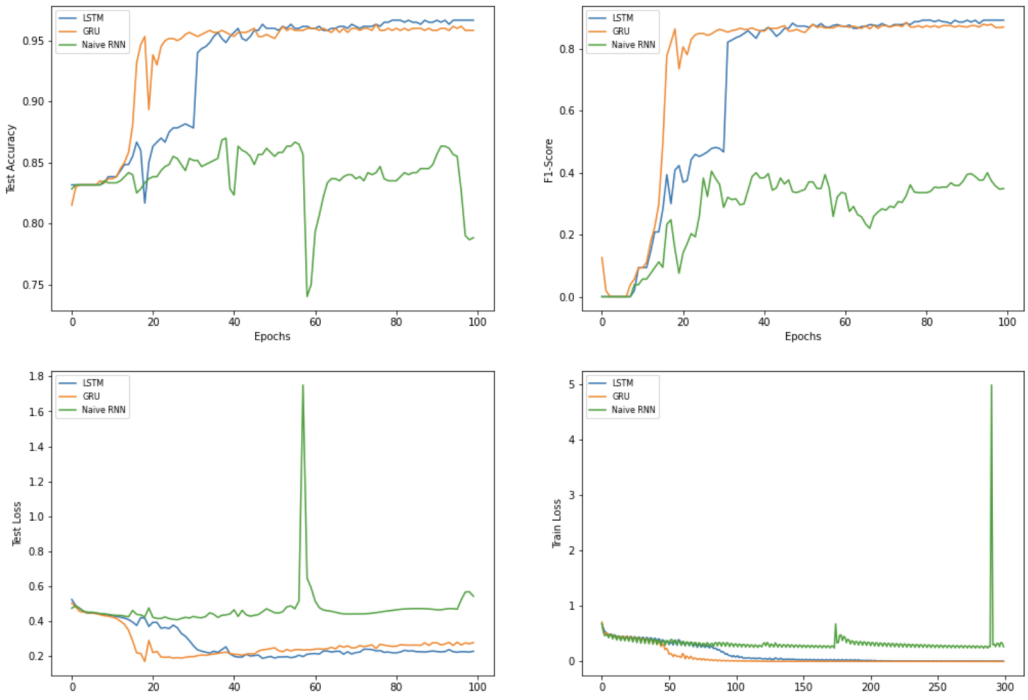

Figure 3 Classifying Emails as Spam or Not Spam. As these are small datasets, GRU performs the best.

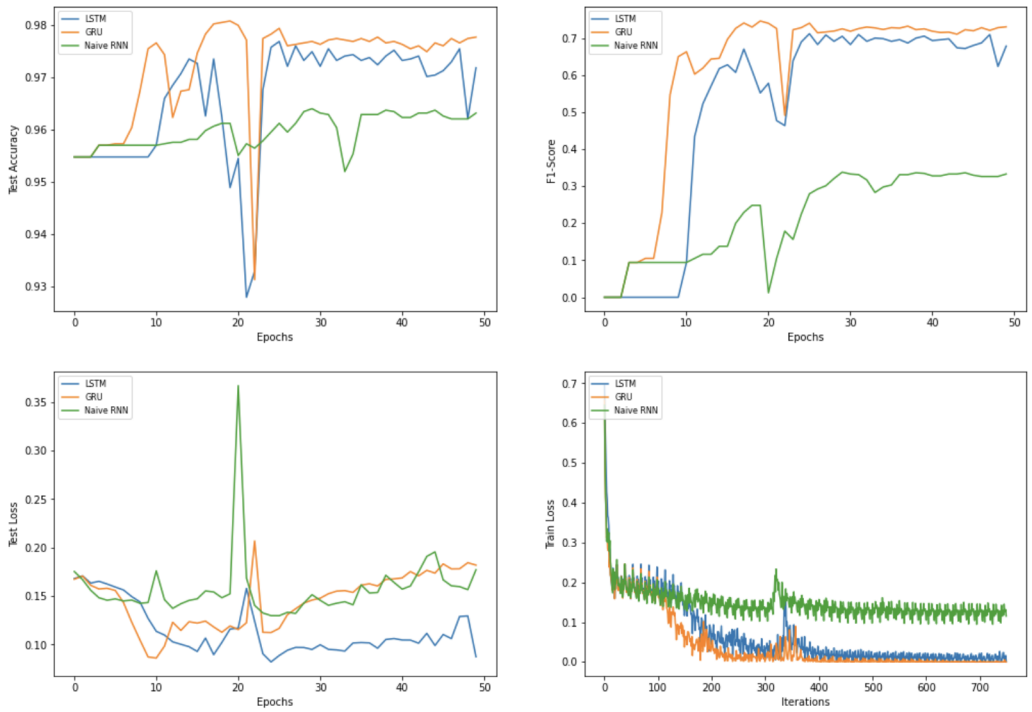

Figure 4 Classifying Job descriptions as fake or not. Here we are running more epochs than required, and we can see it overfitting to training data as the dataset is small.

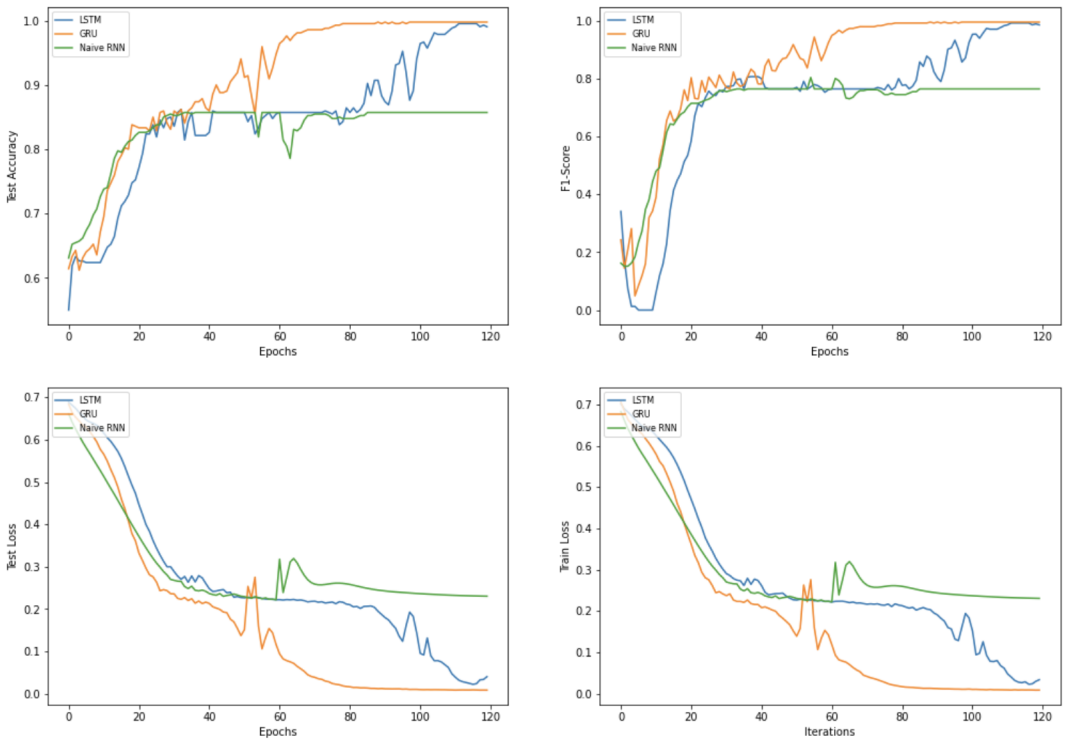

Figure 5 Classifying News articles as fake or not. Here as well, GRU is the fastest to reach the Accuracy, where as LSTM takes a while.

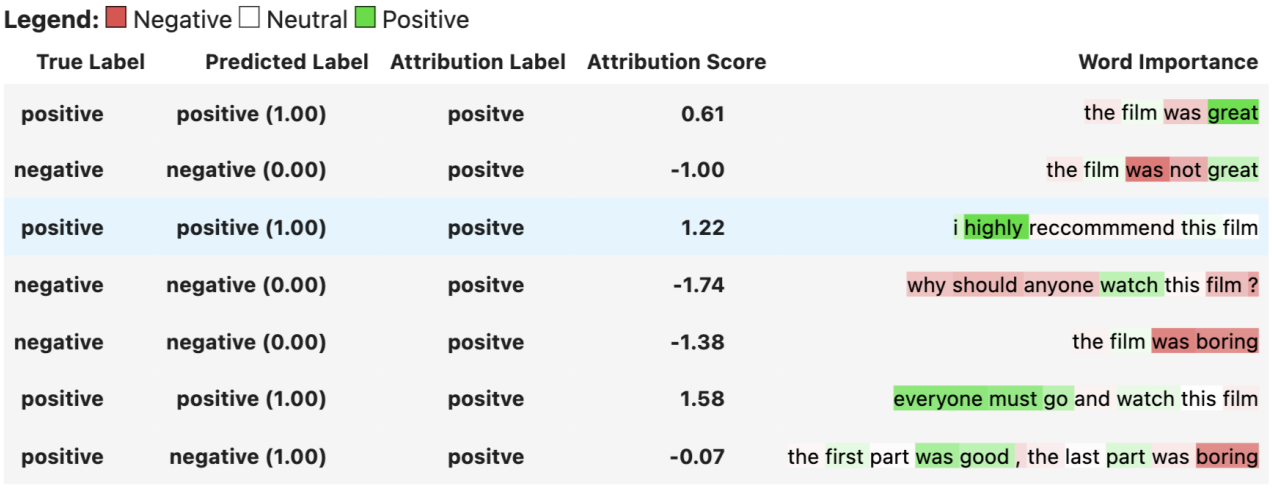

Figure 6 Interpreting the IMDB review classification model. We are taking GRU as an example and interpreting it on the IMDB review dataset. Here, we can see that it understands 'was great' and 'was not great'. Traditional bag of words would have failed to detect these features. Also, sentence 4 does not have any polarizing words like good, or bad. But still, it identifies it as a negative review. (Note. We have tried this out for a different number of epochs and some are better than others. If we stop too soon or too late then we see that it does work as well. Also, the number of epochs required would depend on the initial weights. As they are randomly assigned, deciding the number of epochs beforehand is a bit hard.)

8 Summary and Conclusion

In summary, we started off with building unsupervised language models like Continuous bag of Words and Continuous Skip Gram to learn similar words. Then we compared them using GRU as the classification model. And saw that they help train the model faster. And if we had trained on a large enough dataset like WikiText, then we would have seen an improvement in accuracy as well.
Then we compared 3 different neural network classification models. 1. Fully connected recurrent neural networks, Long Short-Term Memory Networks, and Gated Recurrent Unit Networks on 4 different datasets. As all of these datasets are smaller with sizes less than 50MB. We saw that GRU was able to reach the optimum much quicker when compared to the other two. And in most cases, LSTM and GRU yielded the same accuracy given enough epochs to train. We also, saw that Fully connected RNNs did not yield as high accuracy as LSTM and GRU as it had faced vanishing and exploding gradient problems. Finally, we interpreted GRU on IMDB dataset and inferred on 'How it classifies a review?'.

9 Future work

1. Currently, we have only tested this for binary classification problems. We would like to add support for multi-class problems as well. And then further regression problems. This would not be a hard challenge as it just requires some minor modifications to the output layer.

2. We would like to build and test more neural networks such as Convolutional Neural Networks, Transformers, etc.

3. We would like to implement more complex language models such as glove and train them on huge datasets like WikiText and get the word embedding from it. And using those embedding run the classification test again to see if we could improve the accuracy of the result even more.

References

[1] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

[2] Cho, Kyunghyun, et al. "On the properties of neural machine translation: Encoder-decoder approaches." arXiv preprint arXiv:1409.1259 (2014).

[3] Liu, Pengfei, Xipeng Qiu, and Xuanjing Huang. "Recurrent neural network for text classification with multi-task learning." arXiv preprint arXiv:1605.05101 (2016).

[4] Luan, Yuandong, and Shaofu Lin. "Research on text classification based on CNN and LSTM." 2019 IEEE international conference on artificial intelligence and computer applications (ICAICA). IEEE, 2019.

[5] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[6] Kokhlikyan, N., Miglani, V., Martin, M., Wang, E., Alsallakh, B., Reynolds, J., Melnikov, A., Kliushkina, N., Araya, C., Yan, S. and Reblitz-Richardson, O., 2020. Captum: A unified and generic model interpretability library for pytorch. arXiv preprint arXiv:2009.07896.

[7] Colas, Fabrice, and Pavel Brazdil. "Comparison of SVM and some older classification algorithms in text classification tasks." IFIP International Conference on Artificial Intelligence in Theory and Practice. Springer, Boston, MA, 2006.

[8] Su, J., Zhang, H. (2006, July). A fast decision tree learning algorithm. In Aaai (Vol. 6, pp. 500-505).