# *Multiheaded Structured Self-Attention Using Neural Averaging Layer*

Kaushal Shetty
*Computer Science Engineering*
*Sri Jayachamrajendra College of Engineering*
Mysore, India
kaushalshetty@outlook.com

Sandhya Subramani
*Telecommunication Engineering*
*BMS College of Engineering*
Bengaluru, India
sandhyasubramani14@gmail.com

Vaishnavi Sridhar
*Computer Science Engineering*
*BMS Institute of Technology*
Bengaluru, India
vaishnavi.rao2@gmail.com

***Abstract*—The proposed Attention based Artificial Neural Network model aimed to provide higher prediction accuracy as well as faster training period for classification tasks by designing a Learnt Sentence Embedding that captured the different themes present in a sentence into a single embedding. This was performed by means of a Neural Averaging Layer that employed a weighted architecture to combine the various themes from the multiple heads of the attention layer output. The model also incorporated positional encoding to the word representations of the input in order to accommodate for flexible length in sentences during prediction. The model was implemented in PyTorch on a 16 GB RAM processor, and the efficacy was tested on the IMDB Movie Reviews Sentiment Dataset and the Reuters Newswire Topics Classification Dataset. The attention weights obtained from the model also helped visualize and interpret the model performance, as a heat map plotting of the attention distribution across the text corpus. Upon comparison with the base model and other Deep Learning model architectures, the proposed model was found to result the best accuracy as well as the quickest training time, while being computationally more expensive.**

***Keywords—Deep Learning; Artificial Neural Networks; Natural Language Processing; Supervised Learning; Attention Mechanism; Neural Averaging Layer;***

## I. INTRODUCTION

Sentence representation is an essential component in all Natural Language Processing (NLP) tasks such as Sequence Modelling [1], Language Modelling [2], Machine Translation [3], and even Classification. While there has been significant advancement in the development of semantically meaningful word embeddings [4], sentence embeddings are still being explored, and yet to produce satisfactory representations. Similar to word embeddings, sentence embeddings aim to encode phrases into fixed length dense vectors that can substantially improve the processing of text data.

The sentence embedding representations can be derived either through unsupervised learning or while training a model specifically in a supervised manner to solve a given task. While the embeddings from supervised training have been found to give better prediction results, embeddings like Skip-Thought [5] vectors and FastSent [6] that were generated by unsupervised algorithms are observed to be more generic and therefore usable in variety of domain agnostic applications.

Yet, these embeddings come with their own disadvantages such as the Skip-Thought vectors requiring the sentences to be ordered in a semantically meaningful way, making it unusable in applications that require social media text processing, and fastSent sacrificing word order for efficiency rendering it ineffective in use cases like language translation.

## II. LITERATURE SURVEY

The advent of Attention[7][8][9] mechanism has resolved these issues to an extent. Attention allows for the accurate classification and translation of long sentences by assigning higher weights to the important words at prediction time, thereby partially overcoming information loss.

However, the sentence embedding obtained from the attention models are still ineffective because the multiple heads obtained from the attention layer are either un-weighted averaged or max pooled and sent to the final dense layer. This results in the loss of context of the different themes present in the various heads.

Moreover, none of the mentioned architectures take into account the variations in sentence length. The models employ a fixed length embedding initialized at the time of model training based on the training dataset. During prediction, if the new corpus has a sentence with words lesser than the initialization, padding is done to maintain the dimensionality. Though, if the sentence is longer in length, the end parts get truncated resulting in a loss of information. It therefore results in an optimization issue as well, of the determination of the appropriate embedding size from just the training data.

All the above factors contribute towards the representation of the sentence embedding which in turn directly impacts the prediction accuracy of the model. It is therefore imperative to design an architecture that caters to:

1) The syntax and semantics of the text
2) The variable length of the sentence
3) The different themes in a given sentence

The proposed model addresses the above while also taking into consideration other parameters such as training time, model complexity and compute power.

978-1-7281-0345-7/18/$31.00 ©2018 IEEE

## III. Architecture of Proposed Classification Model

The proposed model comprised of four characteristic functional blocks, namely the Embedding Layer, Attention Layer, Neural Averaging Layer and the Output Layer, as outlined by Fig. 1.

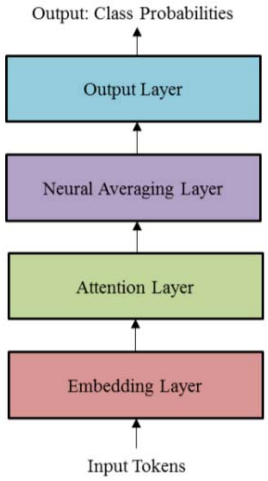


Fig. 1. High Level Architecture

The input tokens were first sent to the Embedding Layer which encoded their respectable positions into their trainable word embedding representations. These embeddings were then sent to the Attention Layer which ensured that the relevant words were attended to, by assigned higher weights to them. The output of the Attention Layer, a 2D sentence embedding that constituted multiple themes, needed to be appropriately encoded into a single 1D embedding without information loss. This was performed with the help of the Neural Averaging Layer. It was finally sent to the Output Layer that consisted of a dense layer which gave as outputs, the *softmax* probabilities for each class.

### A. Embedding Layer

Fig. 2. illustrates the concept of Positional Embedding at the individual token level, which was used to create the final word representation.

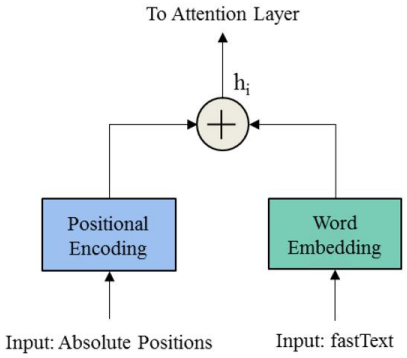


Fig. 2. Positional Embedding of Each Unit

Positional embedding was employed in order to integrate the time component into the word representation as a way to understand the sequence of words. Either the absolute or relative positions could be used. The embedding consists of the integration of two parts:

*1) Positional Encoding (PE):* The proposed model utilized the following sine and cosine functions to take into account, information about the tokens relative positions in the sequence.

$$PE_{(pos,\ 2i)} = sin(pos/10000^{2i/u}) \quad (1)$$

$$PE_{(pos,\ 2i+1)} = cos(pos/10000^{2i/u}) \quad (2)$$

where *pos* refers to the position, *i* is the dimension and *u* is the number of dimensions

*2) Word Embedding (WE):* The word representation embeddings from fastText [10] were used because of their superior syntactic and semantic performance over GloVe [11] embeddings.

Thus, for each input token *i*, fastText was used in conjunction with positional encoding through a summer to arrive at the final embedding $h_i$, as depicted below:

$$h_i = PE + WE \quad (3)$$

where $h_i$ is the embedding for each word *i* in sequence.

### B. Attention Layer

The attention layer was designed as shown in Fig. 3.

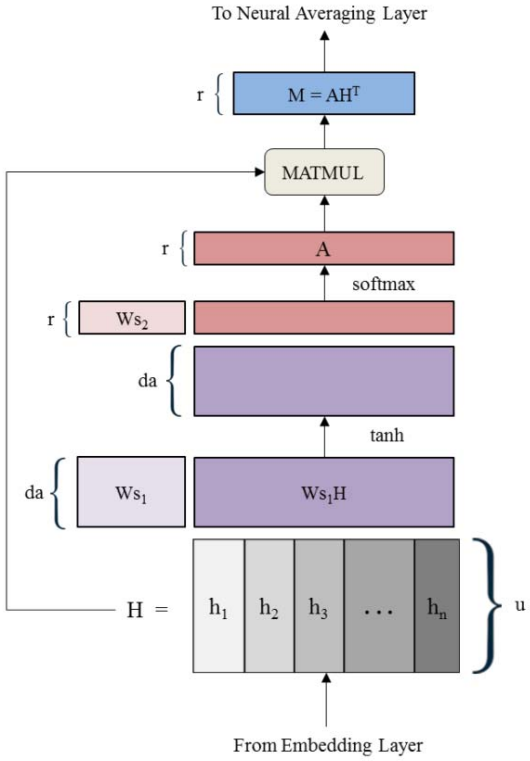


Fig. 3. Attention Layer

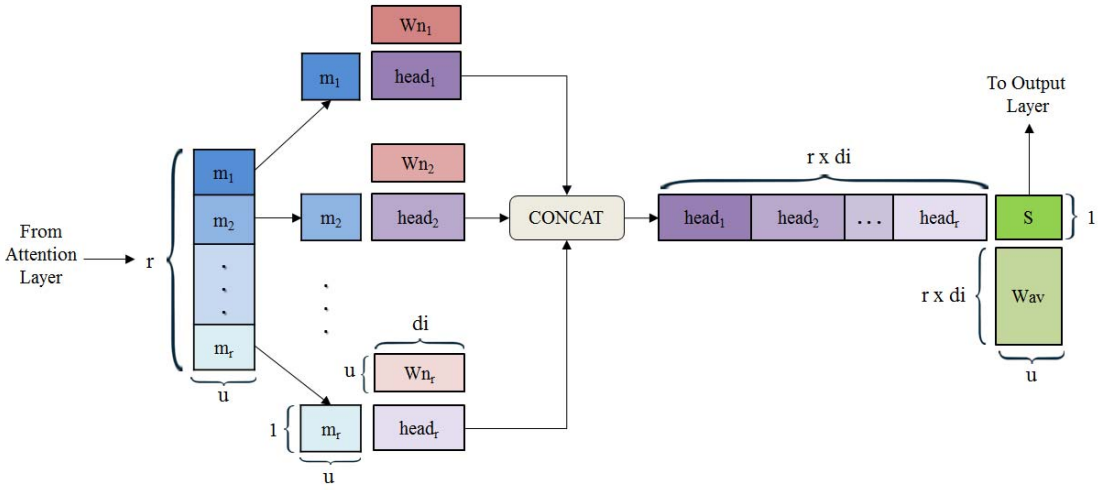


Fig. 4. Neural Averaging Layer

The final embedding output from the Embedding Layer was considered to be $H$, where $H$ was of dimension $(u \times n)$, where $n$ referred to the number of words and $u$ referred to the embedding dimension of each word. Thus,

$$H = (h_1, h_2, h_3, \ldots, h_n) \quad (4)$$

It was multiplied with weight matrix $Ws_1$ of shape $(da \times u)$ to give the product $Ws_1H$, which in turn had a dimension $(da \times n)$. To this, the *tanh* activation function was applied element-wise to get $tanh(Ws_1H)$, which also had the same dimensionality $(da \times n)$. Furthermore, this product was sent through another neural network with weight matrix $Ws_2$ having dimension $(r \times da)$ to determine the multiple heads or themes that the documents presented. It gave the output $Ws_2tanh(Ws_1H))$ that had a shape $(r \times n)$, where $r$ represented the number of themes. *Softmax* was performed along the first dimension to result in $A$, where:

$$A = softmax(Ws_2tanh(Ws_1H)) \quad (5)$$

Then, the initial embedding output $H$ was multiplied with the newly derived $A$ to give the 2D sentence embedding with $r$ multiple heads, denoted by $M$ as shown below:

$$M = AH^T \quad (6)$$

This 2D sentence embedding which has a shape $(r \times u)$ was sent as input to the Neural Averaging Layer.

### C. Neural Averaging Layer

The proposed Neural Averaging Layer, as shown in Fig. 4., employed a neural weighted approach to obtain a single vector embedding from the 2D sentence embedding, as opposed to a simple averaging or max pooling mechanism. This was performed by incorporating yet another artificial neural network to learn the various themes generated from each head present in the output of the Attention Layer, $M$. This approach ensured that the final sentence embedding was a learnt sentence embedding containing the weighted information from all the heads.

Thus, each row of M was multiplied by a unique weight matrix with its subsequent head concatenated with the other output heads to give the 1D vector, as shown below:

$$head_i = m_iWn_i \quad (7)$$

$$Hd = concat(head_1, head_2, \ldots, head_r) \quad (8)$$

In the above equations, $i$ represents each row of $m$, $m_i \in M$, $Wn_i$ is each unique neural weight matrix of dimension $(u \times di)$, and Hd has dimension $(1 \times (r \times di))$. Finally, the learnt sentence embedding $S$ was obtained by passing the 1D vector $Hd$ through a new weight matrix $Wav$ of shape $((r \times di) \times u)$.

$$S = HdWav \quad (9)$$

$S$ has the dimensionality $(1 \times u)$ and can be referred to as the Neural Averaged Sentence Embedding.

### D. Output Layer

The Output Layer consists of a softmax function that was applied to the product of $S$ and a weight matrix $Wop$ having dimensions $(u \times no\_of\_predictor\_classes)$, along with the inherent bias. The final output was a list of probabilities for all classes.

$$Output = softmax(SWop + bias) \quad (10)$$

## IV. Implementation

The model training was performed on a single 16GB RAM processor and implemented in PyTorch [12] to allow for easier debugging while making use of dynamic computational graphs.

### A. Algorithm

The algorithm used for training is as depicted in Fig. 5. Each forward pass was fed in with a sequence of tokens as input, using which the fastText embeddings were indexed and the positional embeddings were obtained. These in turn were summed together to derive the final embedding. This final

embedding was then sent as input to a dense layer with *tanh* activation followed by yet another dense layer with *softmax* activation across the first dimension, in order to get the attention weights. Both these layers have no bias. The attention weights were then multiplied with the final embeddings to achieve a 2D representation of the sentence. Each row of the matrix was further projected across different dimensions, and subsequently by concatenation, re-projected back into the initial dimension to get the 1D sentence embedding. This was passed through a final dense layer with *log softmax* activation.

During training, the Adam optimizer with a learning rate of 0.01 was used. The Negative Log Likelihood (NLL) loss function was applied to solve the classification problem. Hence, in each epoch, batches of input tokens passed through the model and generated as output, a probability distribution across the classes. The NLL loss was back-propagated to update the weights.

```
procedure forward(T)
        abs_pos ← get_absolute_pos(T)
        pos_emb ← get_pos_emb(abs_pos)
        wrd_emb ← get_wrd_emb(T)
        H ← wrd_emb + pos_emb
        A ← softmax(Ws2tanh(Ws1H))
        M ← AH^T
        r ← number_of_hops
        for i = 1, . . . , r; do
                head_i ← m_i Wn_i
        Hd ← concat(head1, head2, head3, . . . , headr)
        S ← HdWav
        return log_softmax(SWop + bias)

procedure training( )
        epoch ← n
        criterion ← NLLLoss( )
        optim ← Adam(lr = 0.01)
        model ← Model( )
        T ← [t1, t2, t3, . . . , tn]
        for e = 1, . . . , epoch; do
                pred ← model.forward(T)
                loss ← criterion(true, pred)
                optim.zero_grad( )
                loss.backward( )
                optim.step( )
        return model
```

Fig. 5. Model Training Pseudo Code

## B. Datasets

The performance of the model was tested on the IMDB Movie Reviews Sentiment Classification Dataset [13] and the Reuters Newswire Topics Classification Dataset [14]. Both datasets exhibit high variance in the inter-corpus sentence structure – comprising of a good mix of short and long sentences and with paragraphs ranging from 16 words to 364 words – making them very good benchmarks for evaluating the success of the model. Both datasets were split into their respective Train and Test datasets at a 6:4 ratio.

The visualizations of the model's performance were created by extracting and plotting the attention weights of each sample. While neural networks are generally considered uninterpretable black box models, the attention weights of the model enabled the user to understand the distribution of attention paid to each word, allowing for easier debugging and correction in the case of words being wrongly attended to.

*1) IMDB Movie Reviews Sentiment Dataset:* The dataset consists of 25,000 user reviews of movies on IMDB, labelled by sentiment. The aim of the task was to classify the movies as positive and negative based on these sentiments.

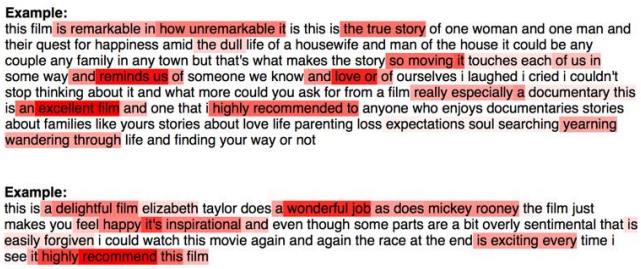


Fig. 6. IMDB Dataset: Prediction of Positive Sentiment Class

Two examples of user reviews that the model classified as positive sentiment are shown in Fig. 6. The intensity of the highlighted colour shows the importance of the words that determined the classification. Similarly, Fig. 7. presents examples of reviews that were identified to portray negative sentiments towards the movies.

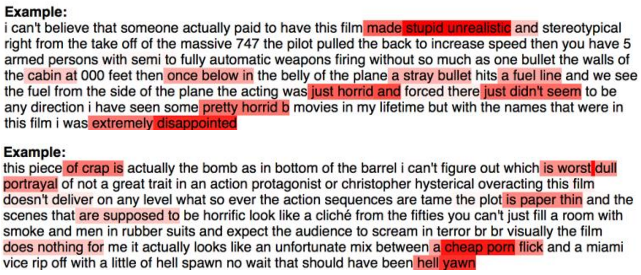


Fig. 7. IMDB Dataset: Prediction of Negative Sentiment Class

*2) Reuters Newswire Topics Classification Dataset:* The Reuters dataset poses a multi-class classification problem. It contains 11,228 datapoints which are newswires from a wide range of topics labelled under 46 different classes pertaining to business, agriculture, arts and entertainment, science and technology, sports, and more. The aim of the task was to accurately segregate the newswires into their classes.

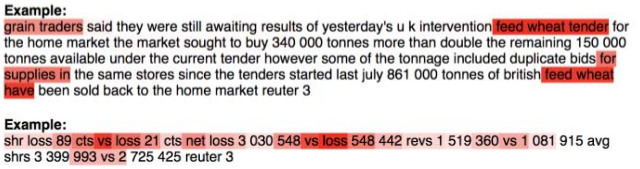


Fig. 8. Multi-class Prediction from Reuters Dataset

Fig. 8. illustrates two different cases of classification from the Reuters dataset, one related to agriculture, and the other belonging to stock price fluctuations, both accurately categorized into their respective classes based on the increased attention weights given to the highlighted words.

## V. Results

### A. IMDB Dataset

The result of model training on the IMDB dataset was inferred from Fig. 9 and Fig. 10. It was observed that the model displayed relatively high Train Accuracy right from the initial few epochs. While this would usually imply that the model is over-fitting, on printing the Cross-validation Score it was noted the Test Accuracy was also nearly at par with the Train Accuracy. Epoch 4 gave the highest Validation Accuracy, after which, on continuing the model training till Epoch 10, it was observed that the Cross–validation Score began to steadily decrease, indicative of the normal behavior of the Machine Learning model.

```
Using TensorFlow backend.
Using settings: {'epochs': 4, 'use_regularization': True, 'C': 0.03, 'clip': Tru
e, 'use_embeddings': False, 'attention_hops': 10}
Using model settings {'batch_size': 512, 'vocab_size': 20000, 'timesteps': 200,
'lstm_hidden_dimension': 50, 'd_a': 100}
Running EPOCH 1
avg_loss is
 0.5326
[torch.DoubleTensor of size 1]

Accuracy of the model 0.7794407894736842
Running EPOCH 2
avg_loss is
 0.3328
[torch.DoubleTensor of size 1]

Accuracy of the model 0.9024876644736842
Running EPOCH 3
avg_loss is
 0.2505
[torch.DoubleTensor of size 1]

Accuracy of the model 0.9387952302631579
Running EPOCH 4
avg_loss is
 0.1895
[torch.DoubleTensor of size 1]

Accuracy of the model 0.9657483552631579
Cross validation score: 0.92822345589018
```

Fig. 9. IMDB Dataset: Training Results

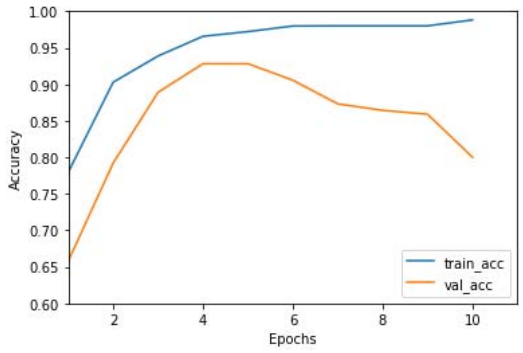


Fig. 10. IMDB Dataset: Training and Cross-validation Variation with Epochs

Therefore, it was tabulated that the proposed model achieved its highest Test Accuracy of 92.82% on the IMDB dataset with just 4 epochs.

### B. Reuters Dataset

The performance of the model on the Reuters dataset is shown in Fig. 11. The model was observed to achieve a notably high accuracy even for a dataset with 46 classes.

```
Running EPOCH 2
avg_loss is
 1.2424
[torch.FloatTensor of size 1]

Accuracy of the model 0.7306985294117647
Running EPOCH 3
avg_loss is
 0.8479
[torch.FloatTensor of size 1]

Accuracy of the model 0.8172104779411765
Running EPOCH 4
avg_loss is
 0.6377
[torch.FloatTensor of size 1]

Accuracy of the model 0.8701746323529411
Running EPOCH 5
avg_loss is
 0.4891
[torch.FloatTensor of size 1]

Accuracy of the model 0.9063648897058824
Running EPOCH 6
avg_loss is
 0.3973
[torch.FloatTensor of size 1]

Accuracy of the model 0.92578125
Running EPOCH 7
avg_loss is
 0.3299
[torch.FloatTensor of size 1]

Accuracy of the model 0.9403722426470589
Running EPOCH 8
avg_loss is
 0.2858
[torch.FloatTensor of size 1]

Accuracy of the model 0.9485294117647058
Running EPOCH 9
avg_loss is
 0.2565
[torch.FloatTensor of size 1]

Accuracy of the model 0.9512867647058824
Cross validation accuracy: 0.8445627282091234
F1 Score: 0.8728953855076038
```

Fig. 11. Reuters Dataset: Training Results

Fig. 12. shows that Validation Accuracy was on an accelerated rise till Epoch 9, where it achieved a relatively high Cross–validation Score of 84.46% and an F1 score of 0.87. The model was further trained till Epoch 15 to check for increased Test Score, however the highest Test Accuracy was still noted at Epoch 9. The model, in its expected behavior portrayed lowered values as the training continued.

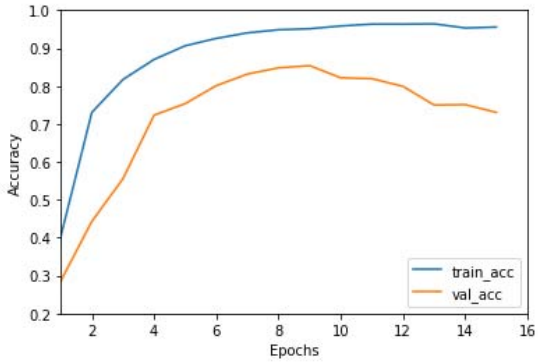


Fig. 12. Reuters Dataset: Training and Cross-validation Variation with Epochs

### C. Performance Evaluation Across Models

When the proposed model architecture was compared against the various other models tabulated in Table I, for both the datasets, the proposed neural averaged sentence representation helped predict better. While the Hierarchical Attention model gave a Validation Accuracy of 90.04% for the IMDB dataset, the highest value amongst its peers, the proposed model delivered an even higher prediction accuracy of 92.82%. The former model achieved its highest accuracy at Epoch 11, while the proposed model was at its zenith by Epoch 4, in less than half the time duration. Similarly, while the Structured Self-Attention model reported an accuracy of 82.66% on the Reuters dataset, the proposed model reported an accuracy of 84.46%. All the contrasted models were trained with the same batch size to maintain uniformity.

TABLE I. COMPARISON OF ACCURACY WITH EXISTING MODELS

| Model | Cross Validation Accuracy (%) | |
|---|---|---|
| | ***IMDB Dataset*** | ***Reuters Dataset*** |
| BiLSTM with fastText embedding + MLP | 86.253 | 82.11 |
| GRU with fastText | 86.82 | 81.48 |
| RNN | 84.67 | 80.27 |
| Unigram bigram TFIDF + SVM | 81.12 | 72.25 |
| Unigram bigram TFIDF + MLP | 83.22 | 76.63 |
| BiLSTM with Max Pooling | 88.92 | 81.73 |
| Hierarchical Attention | 90.04 | 81.52 |
| Structured Self Attention | 89.33 | 82.66 |
| This paper | 92.82 | 84.46 |

It is to be noted that the improved performance was also in part because of the incorporation of positional embedding into the system. Positional embedding allowed the model to capture the complete length of the input corpus during both training and prediction, as opposed to the remaining models where padding was performed to maintain the length of the corpus. This ensured that none of the information got truncated while embedding.

Furthermore, positional embedding also helped negate the latency of the model caused due to the additional neural network layers. Thus, while the training time for an individual epoch was approximately the same for the Structured Self-Attention model and the proposed model, the overall training time was significantly lower for the Neural Averaged model as the text was learnt in much lesser epochs. The rest of the models however, even post pruning, displayed varied training time and accuracy scores, none comparable to the efficiency of the proposed model.

However, the Neural Averaged model was also found to be computationally more expensive than its counterparts, due to the employment of a more complex neural network architecture. The RNN model had the simplest architecture and yet produced the least desirable results. The BiLSTM with Max Pooling model, while resulting in good validation accuracy, took the longest time to train. Thus, it is imperative to arrive at the model most suited for the use-case, depending on the most important parameter consideration for that scenario.

## VI. CONCLUSION

This paper presents the implementation of an improved Structured Self-attention Deep Learning model for the task of classification. It incorporated positionally embedded input tokens into an Attention based architecture that used yet another Artificial Neural Network in its structure to learn the context from the multiheads better. The model resulted in a prediction accuracy of 92.82% and 84.46% for the benchmark datasets of IMDB and Reuters respectively, scoring significantly higher than its counterparts. This increased performance can be attributed to the ability of the model to predict well on test data that has a corpus size even larger than the train data. The overall model training time was also found to be less than half in comparison to its base model, while being computationally more expensive. The proposed model can be applied to a variety of NLP based supervised learning tasks that necessitate high prediction accuracy.

## REFERENCES


[1] I. Sutskever, O. Vinyals, and Q. VV Le., "Sequence to sequence learning with neural networks". In Advances in Neural Information Processing Systems, pages 3104–3112, 2014.

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate". CoRR, abs/1409.0473, 2014.

[3] K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation." CoRR, abs/1406.1078, 2014.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory." Neural computation, 9(8):1735–1780, 1997

[5] R. Kiros,et al, "Skip thought vectors" arXiv:1506.06726v1

[6] F. Hill, K. Cho and K. and A. Karhonen, "Learning distributed representations of sentences from unlabelled data". arXiv:1602.03483

[7] Z. Lin et al "A structured self attentive sentence embedding" arXiv:1703.03130v1

[8] A. Vaswani, et al, "Attention is all you need" arXiv:1706.03762v5

[9] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola1 and E. Hovy, "Hierarchical attention networks for document classification" Proceedings of NAACL-HLT 2016, pages 1480–1489.

[10] fastText website, https://fasttext.cc/

[11] GloVe website, https://nlp.stanford.edu/projects/glove/

[12] GitHub PyTorch webpage, https://github.com/pytorch/pytorch

[13] Keras dataset webpage, https://keras.io/datasets/#imdb-movie-reviews-sentiment-classification

[14] Keras dataset webpage, https://keras.io/datasets/#reuters-newswire-topics-classification

## ABOUT THE AUTHORS

[1] 

Kaushal Shetty at the time of writing this paper is an AI Applied Research Engineer working in the financial services industry. He received his B.E in Computer Science Engineering from Sri Jayachamrajendra College of Engineering, Mysore, in 2016. His areas of interest are Deep Learning, Natural Language Generation and Artificial Intelligence.

[2] 

Sandhya Subramani at the time of writing this paper is an AI Applied Research Engineer working in the financial services industry. She received her B.E in Telecommunication Engineering from BMS College of Engineering, Bengaluru, in 2016. Her areas of interest are Natural Language Processing, Object Oriented Programming and Artificial Intelligence.

[3] 

Vaishnavi Sridhar at the time of writing this paper is an Associate Software Engineer in Machine Learning and is working in the finance industry. She received her B.E in Computer Science Engineering from BMS Institute of Technology, Bengaluru, in 2016. Her areas of interest are Artificial Intelligence, Embedded Systems and Cognitive Computing.