Lifting Sequence Length Limitations of NLP Models using Autoencoders

Reza Marzban[a] and Christopher Crick[b]
Computer Science Department, Oklahoma State University, Stillwater, Oklahoma, U.S.A.

Keywords: Deep Learning, Natural Language Processing, Artificial Intelligence, Transformers.

Abstract: Natural Language Processing (NLP) is an important subfield within Machine Learning, and various deep learning architectures and preprocessing techniques have led to many improvements. Long short-term memory (LSTM) is the most well-known architecture for time series and textual data. Recently, models like Bidirectional Encoder Representations from Transformers (BERT), which rely on pre-training with unsupervised data and using transfer learning, have made a huge impact on NLP. All of these models work well on short to average-length texts, but they are all limited in the sequence lengths they can accept. In this paper, we propose inserting an encoder in front of each model to overcome this limitation. If the data contains long texts, doing so substantially improves classification accuracy (by around 15% in our experiments). Otherwise, if the corpus consists of short texts which existing models can handle, the presence of the encoder does not hurt performance. Our encoder can be applied to any type of model that deals with textual data, and it will empower the model to overcome length limitations.

1 INTRODUCTION

Deep learning is a machine learning (ML) technique designed as a very rough analogy to the neurons in a human's nervous system. Over the last decade, deep learning architectures have substantially outperformed other ML algorithms, especially on unstructured data, provided that we have a huge amount of data for training the model. Different deep learning architectures are designed to work with specific types of data. Convolutional Neural Networks (CNN) work well with image data, while Recurrent Neural Networks (RNN) work best with time-series or sequential data like texts. However, deep learning presents a number of challenges as well. Most importantly, it is often described as a black box: we throw some data at it and hopefully get good outputs. This is unhelpful if we want to optimize the network to improve performance. We need to understand what is happening inside, what kind of data is being created at each layer and passed to the next layer, and how can we make it work better.

Natural language processing (NLP) is a popular application for artificial intelligence, helping machines to communicate in human languages. Among various tasks, sentiment analysis is the basis of many NLP problems. This involves creating a model that learns the semantic value of each word and classifying a text according to the overall mood valence. Our goal is first to find or create the best possible network for this specific task, and then generalize the technique for a wider variety of NLP tasks.

One variety of RNN is the Long Short-term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) network. Such architectures include a memory model that makes them a popular choice for sequential or time-series problems such as interacting with textual data. LSTM networks attempt to improve on the methods of naive RNN models, but we know that LSTM structures still have many limitations. They do not work well enough when the number of words, or the variability in word count between texts, is high. Recently, we have seen an explosion in the diversity of NLP models, and researchers have devised models like BERT (Devlin et al., 2018) to boost performance by pre-training models with a huge amount of data, while also incorporating other techniques such as attention. BERT has been extremely successful in NLP research, but in common with LSTM approaches, it does not handle texts with high word counts very well. BERT, specifically, is designed only to handle texts of 512 words or fewer.

In this paper, we focus on overcoming this maximum sequence length limitation by introducing an

[a] https://orcid.org/0000-0003-2762-1432
[b] https://orcid.org/0000-0002-1635-823X

Marzban, R. and Crick, C.
Lifting Sequence Length Limitations of NLP Models using Autoencoders.
DOI: 10.5220/0010239502280235
In *Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2021)*, pages 228-235
ISBN: 978-989-758-486-2

encoder layer after the embedding layer of a model. Most modern NLP models use Word2Vec (Mikolov et al., 2013) embeddings to represent words. Our encoder uses the output of the embedding layer applied to a text with a large word count and reduces its dimensionality to a size compatible with a network such as BERT. We have tested our new encoder technique on several models such as LSTM and BERT. This paper is not about fine tuning our models and achieve the highest accuracy on different datasets. Our goal is to study the effect of inserting our encoder into different models on various datasets and support our hypothesis: including an engineered encoder layer helps overcome the maximum sequence length limitation of NLP models.

Our experiment on the LSTM model involved creating and comparing three models and evaluating their performance. Our first baseline model was a one-dimensional CNN, while the second one was an LSTM with 128 internal nodes. We compared the performance of these models with our approach, where we added a convolutional autoencoder to the LSTM network to reduce the input dimensionality, then fed the reduced-size data to the LSTM layer. We also tested our encoder on the BERT model. In order to achieve that, first we created a smaller version of BERT, then compared its performance with and without our encoder. Creating a smaller BERT model helped us to expedite our experiment as the original BERT must pre-train for several weeks using several tensor processing units (TPUs). After successfully creating our small BERT model, we tested it our datasets.

The results show that in both cases, inserting the new autoencoder layer increases the number of network parameters very modestly (*e.g.* in our reduced-size BERT, by less than one percent), and does not increase training time significantly. On the other hand, it disposes of the word length limitation and improves classification performance significantly if the data contains long text sequences. If the corpus only contains short texts, adding our encoder does not significantly change a model's behavior. Thus, the encoder never hurts performance, but improves it when long texts are encountered.

2 RELATED WORKS

NLP has long been an important application area for artificial intelligence and machine learning. NLP (Hirschberg and Manning, 2015; Goldberg, 2016) is the science of teaching a machine to understand and produce content based on human languages. Recently, there has been a huge improvement in NLP tasks with the help of deep learning. Artificial neural networks have exceeded traditional machine learning algorithms in many different fields, like machine vision, and NLP has benefited likewise.

Although deep learning has boosted performance, it has also introduced new problems. Researchers have therefore focused on developing new architectures and improving current ones. One such architecture is LSTM (Hochreiter and Schmidhuber, 1997), which is a special case of Recurrent Neural Network introduced in 1997. LSTMs are often used for time series data as they are equipped with an internal memory that can remember important data from previous time steps. They can also decide which data to ignore or forget when the model determines they are not material to its output.

NLP is also categorized as time-series data, where each word in a sentence is analogous to a time step. One of the basic yet difficult problems in NLP is text sentiment analysis (Venugopalan and Gupta, 2015), which deals with the computational study of a text's author's emotions, opinions, and sentiments expressed in textual data. The problem is usually cast as a binary classification that separates negative texts from positive ones. In order to get a good result in many NLP tasks, we need to transform our data from a textual format to a numerical one (Collobert et al., 2011). These numerical values convey the semantics of each word, so the model can understand their relationships. One-hot encoding is one way of doing so, but it requires a very large sparse matrix which is not desirable due to memory limitation. Another method is to use word embeddings as suggested by Bojanowski (Bojanowski et al., 2017), and the most common approach is to employ Word2Vec (Mikolov et al., 2013) to create these embeddings. Word embedding (Kusner et al., 2015) uses an unsupervised learning algorithm to assign a vector with a custom length to each word in the training set. There exist a number of pre-trained, readily-available word vector datasets; one of the most famous is Stanford's Glove (Pennington et al., 2014), which was trained on Wikipedia.

Another type of neural network is the CNN which is believed to work best on images, but many researchers have nevertheless applied CNNs to textual data with excellent performance (Kim, 2014; Conneau et al., 2016). Many papers have compared LSTM and CNN architectures on different kinds of data to determine which is most efficient and appropriate for particular tasks. Yin (Yin et al., 2017) compared CNNs and LSTMs on NLP tasks and claimed that LSTMs work better in most cases. Some authors have tried a mixture of both in order to improve performance (Wang et al., 2016; Sainath et al., 2015). CNNs can also be used as the basis of an autoencoder (Zhu and Zabaras, 2018). Such architectures learn to distinguish important features in order to reduce the dimensionality of the input.

More recently, a new wave of models have boosted performance in NLP tasks, starting with Vaswani (Vaswani et al., 2017). These models create a new type of encoder called a transformer which is based on attention. Transformers have been used to create much more advanced language models like BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019) and GPT3 (Brown et al., 2020). BERT (Devlin et al., 2018) is a state-of-the-art model built upon transformers that are pretrained on a huge amount of unlabeled data for several weeks. BERT can be fine-tuned for any NLP task for a few epochs and its performance is much better than preceding models. Several other scientists have tried to further improve BERT by tuning hyper-parameters and also pretraining for a longer time on more data (Sun et al., 2019; Liu et al., 2019). Yang (Yang et al., 2019) has created a model called XLNet, and claims to overcome some of BERT's limitations. Longformer (Beltagy et al., 2020) has tried to overcome the sequence length limitation in the attention-based transformers, but this technique can only be applied on transformer architectures. We need a technique that can be applied to all models that handle textual data.

Although this field has seen a great deal of rapid improvement, there are still many holes in our knowledge of deep learning networks. We do not know how these models perform when applied to long texts, or when the length variance between different texts within a training corpus is high. It seems that LSTMs and BERT work particularly well on textual data, but is this true for all forms of such data? Can they memorize the information necessary for a decision when the text in question is article-length, rather than the size of a tweet? Can they be adapted to any kind of textual data with any size? In most NLP models, there is a hard limitation on the maximum number of words or tokens that each entry can have. In this paper, we overcome that limitation, as there might be many cases when performing sentiment or other textual analysis at the level of paragraphs or even longer text lengths would be helpful. Current models will truncate texts that are longer than their maximum size, but by doing so, they lose a lot of information.

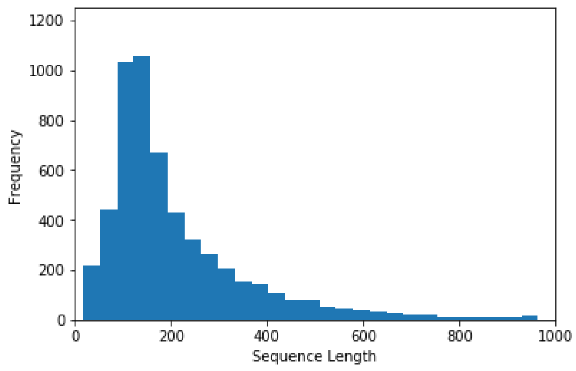

Figure 1: IMDb sequence length distribution.

3 TECHNICAL DESCRIPTION

3.1 Dataset Preprocessing

In order to demonstrate our approach, we have used a broad range of data of various types. The IMDb Large Movie Review Dataset (Maas et al., 2011) contains very long movie review texts classified in binary fashion according to sentiment. This allows us to test the performance of every model on very long sequences. As Figure 1 shows, most of the data consists of sequence lengths of greater than 100 words (tokens).

In addition to the IMDb, we used a dataset of Amazon Kindle book reviews and user ratings. This problem again belongs to sentiment analysis, but rather than a binary classification, it has categorical ratings with 5 classes. In addition, we used a large dataset of Stack Overflow questions and tags, where each question has a label or a tag that specifies which language or concept the question is about. It has 20 possible tags. The mean sequence length in Stack Overflow dataset is the largest. Our final dataset is the 20 Newsgroup dataset (Mitchell, 1999) that contains full text news articles classified into twenty possible categories. In both the Stack Overflow and 20 Newsgroup datasets, we only use texts that have at least 128 tokens (a.k.a. words) to observe the effect of very long texts on our models.

These four datasets were used in testing versions of our model and the various comparison benchmarks. We tested the original BERT model, and a simplified BERT version we created, on the IMDb and Stack Overflow datasets.

In the cleaning phase, we removed all numerical values, symbols and stop words. We also removed words that did not appear in our entire corpus at least twice and words of only one or two characters, and converted the whole corpus to lower case. The models expect an input vector of fixed length, but different sentences and paragraphs obviously have different lengths. In order to solve this problem, we choose a number that is close to the maximum of the length of all sentences in the whole dataset, ignoring a few outliers which are unusually long in comparison to other records. We pad the sentences with zero values at the end of records that have fewer words than our threshold. In order to create the embedding, we used the Stanford GloVe pre-trained embedding trained on Wikipedia. Word2Vec embeds words within vectors of consistent but arbitrary dimensionality, and in this case, we chose 100 for the embedding vector size.

These numbers represent the semantic relationships among words. As a result, we expect the distance between vectors of "Boston" and "Massachusetts" to be near the distance between "Austin" and "Texas". This will help our model to understand the overall meaning of each sentence. This matrix would be the preprocessed input of our future models. As we train the larger overall model, we can train the embedding at the same time, so that the embedding representation is updated after each iteration to represent the words more accurately, or we can fix the embedding weights in advance, so that the model can concentrate on training other layers' weights.

3.2 Encoder Technical Details

Our encoder is designed to reduce the length of the input sequence, as some of the models have hard limits on maximum sequence length. The goal of this encoder is to reduce the dimensionality of input so that larger texts can be fed into popular models. Currently, if a model gets an input with larger sequence length than allowed, it will simply truncate the sequence and ignore potentially useful data in the latter part of the text. The architecture of the encoder is very simple, consisting of two layers. The first layer is a 1-dimensional convolutional layer and the second one is a 1-dimensional max-pooling layer. The number of filters in our convolutional layer is equal to the length of the embedding vector used to represent words, which was equal to 100 in our case. The filter length, as well as the pool size in the pooling layers, are hyperparameters that can be tuned; in our case, they were set to 5 and 2 respectively. Our encoder is applied right after the embedding layer. If the output of the embedding layer has the shape of $m * k$, in which m is the number of words, and k is the length of the embedding vector for each word, the output of encoder layer is calculated by Equation 1.

$$(m,k) \rightarrow \left(\frac{m-f+1}{s}, k\right) \quad (1)$$

In Equation 1, the left side is the input shape and the right side is the output shape. f is the length of filters and s is the pool size in the pooling layer. Notice that the depth of input does not change at all, as we intend to reduce the number of sequence steps (words) in our input and not the size of the embedding. If we needed a smaller embedding length, we could simply specify it in the embedding layer. Initializing these hyperparameters according to our specific model, the output shape is shown in Equation 2.

$$(m,100) \rightarrow \left(\frac{m-4}{2}, 100\right) \quad (2)$$

Thus, if the embedding output's shape is $(512,100)$ (512 words, each represented by 100 numerical values), then after passing through our encoder, its shape will be $(254,100)$. The output will represent 254 important features to be found within the entire text, instead of merely truncating the input text. The output of our encoder can be fed into the model instead of the embedding output. This architecture allows the use of texts twice as long as the original models can handle, but this 2:1 scaling is not required. If further reduction is needed, more pooling or more layers can be added as necessary.

The hyper-parameters in both our encoder and models were selected by randomly sampling the parameter space and selecting for decent performance.

3.3 Testing Encoder on LSTM

In order to evaluate our encoder's performance in an LSTM context, we created three models and tested these three on different datasets. Each of these models varies in the structure and most of the hyper-parameters, but the following hyper-parameters are the same for all three: the learning rate and learning rate decay are both 0.001, we used the Adam optimizer, the batch size was 32, the number of epochs was 20, we applied an L2 regularizer to the last layer of each model, and used the cross entropy loss function for all models (binary cross entropy for movie review data, categorical cross entropy for Amazon Kindle reviews, Stack Overflow and 20 Newsgroup). We kept all of these hyper-parameters consistent in order to conduct a fair experiment in comparing these models, so that our results are only dependent on the architecture of our network.

Our first model is CNN. The details of each layer are presented in Figure 2. We have included the input shape and number of parameters according to our movie review data because of the input shape and number of parameters in the embedding layer change according to the data. As a result, we do not include

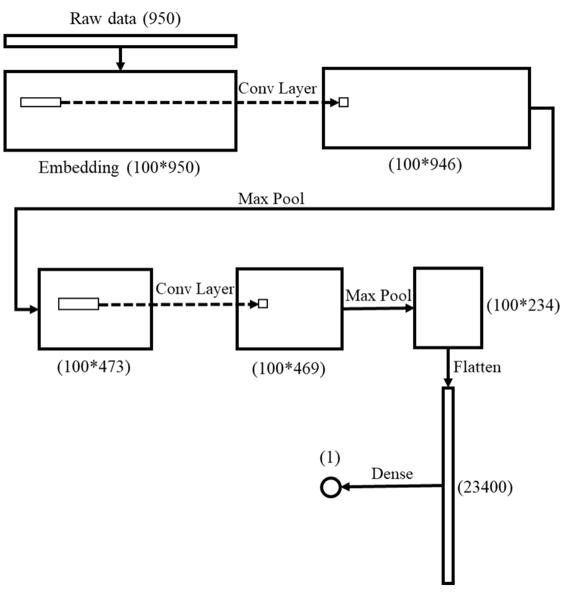

Figure 2: CNN Model.

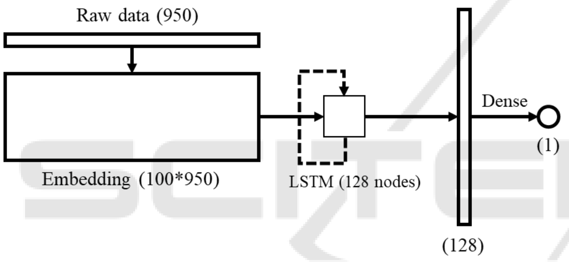

Figure 3: LSTM Model.

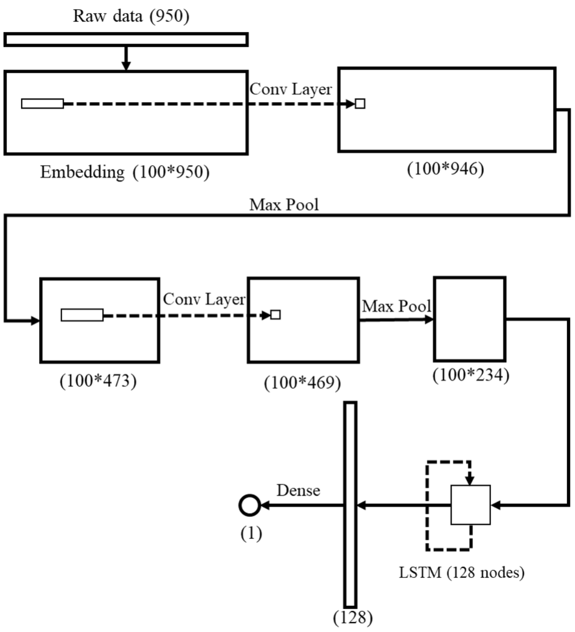

Figure 4: Encoder-LSTM Model.

the embedding layer parameters in our model's total parameters count, as it is the same for all models, and it varies based on data. Our CNN model has 123,601 total parameters to be trained, with two one-dimensional convolutional layers, both with 100 filters of size 5 and stride 1, and with RELU as the activation function. Both max-pooling layer window sizes are 2.

Our LSTM model has 128 nodes and 117,377 parameters ignoring the embedding layer weights, as shown in Figure 3. The original output of this layer has the shape of (?,950,128) but we only send the last output data to the next layer which has the shape of (?,128) instead of the full sequence of data (the question mark in the shapes represents batch size). We have used backward LSTM due to the fact that most of the sentences are padded with zero at the end, and if we do not use backward LSTM, the zeroes will cause the system to forget important information.

The final model we describe is our encoder-LSTM model, which uses the two encoder layers specified before and then uses an LSTM to predict according to the encoder-shortened text. The first part of this model is identical to our CNN model; we just removed the flatten layer and added a 128-node LSTM layer before the dense layer. Notice that the last layer of these models is fully connected with a single output node and sigmoid activation function. When applied to the multi-class datasets, the number of output nodes is increased to be equal to the number of classes, activated with the softmax function.

3.4 Testing Encoder on BERT

In order to check the efficiency of our encoder in BERT, we created an implementation of BERT from scratch and then inserted the encoder in the proper place. As the original base BERT has many parameters, and it takes it several weeks to pre-train on a huge amount of data, we created a smaller version of BERT and tested and compared its performance with and without an encoder. In order to create a smaller BERT, we reduced the hidden layer size from 768 to 200, the intermediate size from 3072 to 800, and the number of hidden layers from 12 to 6. The implementation details of Base BERT, Small BERT and Small BERT with encoder can be observed in Table 1.

After creating the small BERT, we inserted the encoder after the embedding preprocessing and just before post-processing. The original BERT has a maximum sequence length limitation of 512, which remains the same in the small BERT. Small BERT is much faster than the original Base BERT as it has reduced the number of parameters by 65% but the performance was not affected significantly. It helped us to develop and test models much faster. Even after introducing the encoder into the model, the number of parameters just increased by around 1%. As we did not have access to the original data that BERT was pre-trained on, which was a mixture of book corpora and Wikipedia, we trained our small BERT models on a much smaller dataset consisted of 50 book texts. Both decreasing the size of the model and size of the pre-training data have affected our accuracy a little, but it was not important to our hypothesis. We used small BERT as an environment to check the efficiency of our encoder.

We trained the small BERT with encoder and without encoder on 50 books data for 2,000 epochs and then used these two pre-trained models as the baseline for our fine-tuning tasks. We fine-tuned these models on two sets of data: the IMDb and Stack Overflow datasets. The Amazon Kindle and and 20 Newsgroup datasets were sufficiently dissimilar to the training corpus used by the small BERT network that they did not provide useful results. Each fine-tuning was done with 3 epochs. We also downloaded the original pre-trained base BERT model and fine-tuned it on the same data to compare the results.

4 EXPERIMENTAL RESULTS

4.1 Performance on LSTM

After training our models on the training sets for 20 epochs, we evaluated each of them with test sets. We ran each model on each data set twice, once with embedding layer training and once with untrainable embeddings to check the effect. Table 2 shows their test accuracy over our four different datasets. For the movie reviews, the random prediction baseline is 50% (as it is a binary classification), whereas it is 20% (5 classes) in the Amazon review dataset and 5% (20 possible tags) in both the Stack Overflow and 20 newsgroup datasets. The Stack Overflow and 20 newsgroup datasets only contain texts of 128 words or longer. Out of all of the experiments performed, our encoder-LSTM model achieves better accuracy in seven out of eight cases (a CNN works better in one instance).

Table 1: BERT models comparison.

BERT version	Base	Small	Small with Encoder
Hidden size	768	200	200
Intermediate size	3072	800	800
Attention heads	12	10	10
Hidden layers	12	6	6
Use encoder	No	No	Yes
Pre-train data	Book Corpus + Wikipedia	50 Books	50 Books
Total Parameters	178,566,653	81,326,847	81,927,447

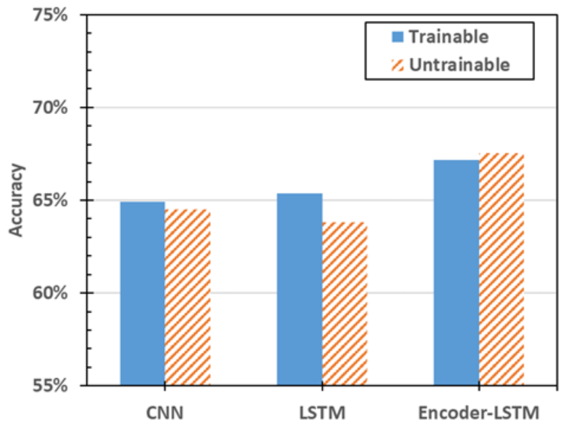

Figure 5: Stack Overflow dataset test accuracy.

Figures 5, 6 and Table 2 show an interaction effect between the model and the embedding. In 9 cases out of 12, the models work better with embedding layer training. In addition, there is a huge main effect from the model factor. The LSTM network does not add much accuracy, while in comparison to the CNN, it takes much longer for the LSTM to be trained. On the other hand, when we are using our encoder with LSTM, it increases the accuracy.

4.2 Performance on BERT

We used the IMDb and Stack Overflow datasets to test the efficiency of our encoder in the BERT model. We tested all datasets on three models: the original base BERT, the small BERT that we created and the small

Table 2: LSTM accuracy comparison.

Data set	Embedding	**CNN**	**LSTM**	**encoder LSTM**
Stack Overflow	*Trainable*	64.90	65.38	**67.17**
	Untrainable	64.51	63.82	**67.55**
Movie Review	*Trainable*	86.26	85.50	**87.02**
	Untrainable	**86.27**	86.02	83.81
Amazon Review	*Trainable*	53.07	53.22	**53.37**
	Untrainable	50.13	50.91	**51.69**
20 News group	*Trainable*	48.32	38.29	**50.23**
	Untrainable	42.26	35.12	**45.84**

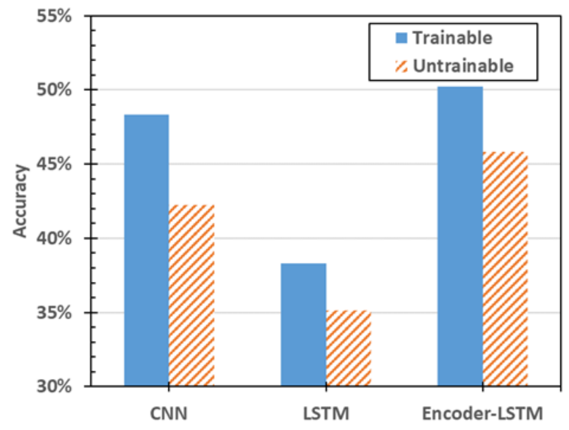

Figure 6: 20 newsgroup dataset test accuracy.

Table 3: BERT accuracy comparison.

Data set	Max Length	Base BERT	Small BERT	Small BERT - encoder
IMDB	512	84.98%	78.16%	76.26%
StackOverflow	128	10.49%	10.59%	25.20%
StackOverflow	512	81.08%	19.61%	27.65%

BERT with encoder. The test accuracy of these models is reported in Table 3. In the IMDb dataset, the random prediction baseline is 50%, but in the Stack Overflow dataset, it is 5% as we have 20 possible classes.

Table 3 shows that introducing the encoder into BERT does not affect the accuracy significantly in IMDb dataset. Certainly the performance of small BERT compared to original BERT is significantly lower, but that was expected due to the fact that small BERT is both a much smaller model (discussed in the technical description) and is pre-trained on a much smaller dataset. The small BERT helped us to check if the encoder can be inserted into BERT or not.

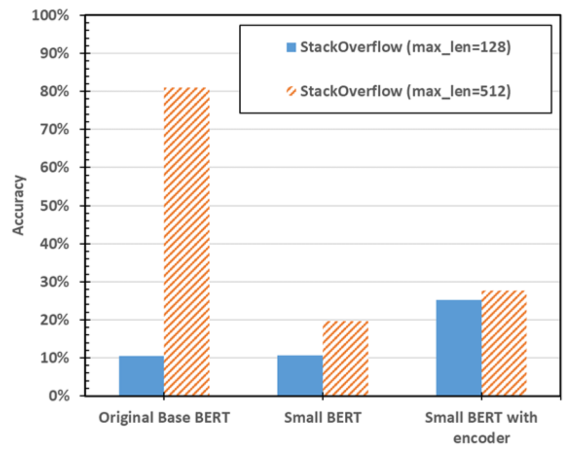

Figure 7: Effect of encoder on BERT accuracy in Stack Overflow dataset.

In our Stack Overflow dataset, when the maximum length is set to 128, the performance of the original base BERT and our small BERT are identical (see Table 3 and Figure 7), but we have a significant improvement after introducing our encoder. This is due to the fact that this dataset contains some very long texts. Also the problem is more complicated than a normal sentiment analysis as we need to find the best tag among 20 possible labels. When the maximum length is set to 512, the performance of the original base BERT is very high; on the other hand, our encoder still improves the performance of small BERT.

4.3 Result Analysis

According to the experiment results presented in this paper, we have shown that our encoder can be used as a tool for any type of NLP model to overcome the maximum sequence length limitation. We have tested our encoder on two popular NLP models, LSTM and BERT, and observed that if the dataset does not contain long texts, inserting the encoder into these models does not affect accuracy. However, if the corpus contains longer textual data, it will improve accuracy significantly. In addition to that, as the number of parameters in these models increases very little (around 1%), it does not affect running time. Each encoder layer reduces the sequence length approximately to half of its original size (the exact value can be calculated by Equation 1); if we need to decrease it further, we can use more encoder layers stacked on top of each other (in the LSTM experiment we used two layers of encoder while for BERT we used just one layer).

5 CONCLUSION AND FUTURE WORKS

LSTMs were designed to overcome some of the limitations that RNNs encounter with time-series data, and they usually succeed in outperforming the older architecture. BERT achieved a huge improvement in all types of textual data problems by using transfer learning. These two are among the best models designed for NLP tasks, but both share a maximum sequence length limitation. In other words, neither is capable of appropriately processing long texts. We have devised a technique to overcome this limitation by creating an encoder layer that will reduce the dimensionality of the input. Results showed that it provides us with the ability to process longer texts and improve accuracy.

We are planning to investigate different encoder architectures and initializations to determine which is most beneficial. In addition, we want to use the same technique on other types of time-series data and measure the usefulness of the encoder technique in length reduction for non-textual data.

REFERENCES

Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.

Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. (2016). Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420.

Hirschberg, J. and Manning, C. D. (2015). Advances in natural language processing. *Science*, 349(6245):261–266.

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

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kusner, M., Sun, Y., Kolkin, N., and Weinberger, K. (2015). From word embeddings to document distances. In *International conference on machine learning*, pages 957–966.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mitchell, T. (1999). The 20 newsgroup dataset.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Sainath, T. N., Vinyals, O., Senior, A., and Sak, H. (2015). Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4580–4584. IEEE.

Sun, C., Qiu, X., Xu, Y., and Huang, X. (2019). How to fine-tune bert for text classification? *arXiv preprint arXiv:1905.05583*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Venugopalan, M. and Gupta, D. (2015). Exploring sentiment analysis on twitter data. In *2015 Eighth International Conference on Contemporary Computing (IC3)*, pages 241–247. IEEE.

Wang, J., Yu, L.-C., Lai, K. R., and Zhang, X. (2016). Dimensional sentiment analysis using a regional cnn-lstm model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 225–230.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Yin, W., Kann, K., Yu, M., and Schütze, H. (2017). Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*.

Zhu, Y. and Zabaras, N. (2018). Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447.