Sensors & Transducers, Vol. 249, Issue 2, February 2021, pp. 110-118

www.sensorsportal.com

Sensors & Transducers

Published by IFSA Publishing, S. L., 2021
http://www.sensorsportal.com

Experimental Comparison of Transformers and Reformers for Text Classification

*** Roghayeh Soleymani, Julien Beaulieu and Jérémie Farret**
Inmind Technologies Inc., Montreal, Canada
Tel.: + 1(514) 871-0470
* E-mail: esoleymani@inmindtechnologies.com

Received: 30 November 2020 /Accepted: 15 January 2021 /Published: 28 February 2021

Abstract: In this paper, we present experimental analysis of Transformers and Reformers for text classification applications in natural language processing. Transformers and Reformers yield the state-of-the-art performance and use attention scores for capturing the relationships between words in the sentences which can be computed in parallel on GPU clusters. Reformers improve Transformers to lower time and memory complexity. We will present our evaluation and analysis of applicable architectures for such improved performances. The experiments in this paper are done in Trax on Mind in a Box with four different datasets and under different hyperparameter tuning. We observe that Transformers achieve better performance than Reformers in terms of accuracy and training speed for text classification. However, Reformers allow the training of bigger models which would otherwise cause memory failures with Transformers.

Keywords: Natural Language Processing, Text Classification, Transformers, Reformers, Trax, Mind in a Box.

1. Introduction

Text classification is one of the important natural language processing (NLP) tasks that is applicable in real-world problems such as topic tagging, question answering, spam detection, sentiment analysis, news categorization, etc. Text classification or tagging can be used in the Fog layer computations to make text data deep [1] or to enrich data. This task includes an NLP step immediately after data acquisition, in Fog layer or in the cloud to organize and structure data. This strategy makes organizing, accessing, transmitting, and processing the data more efficient.

Designing text classification models with classical machine learning methods follows a two-step process: feature extraction and classification. Feature extraction consists of defining some hand-crafted feature vectors and designing high quality features. This step requires strong domain knowledge, tedious feature engineering and analysis, and may not be transferable between different applications.

Since 2012, deep learning models are being applied to different tasks in the world of machine learning and therefore in NLP. Neural network architectures used for text classification include recurrent neural networks (RNNs), convolutional neural networks (CNNs), Capsule Nets, Transformers, Reformers, and more (see the review by Minaee, *et al.* [2]). Most of these models learn feature representations automatically with less hand-engineering and allow for knowledge transfer between different applications of NLP.

In this paper, we focus on Transformers and Reformers that both rely on attention mechanisms for relating different positions of a single sequence in parallel and yield state-of-the-art results in many NLP tasks [2-4]. These models are useful to train language models on Open Data and perform transfer learning on

the application on hand (e.g. Deep Data and Data enrichment as a target application). Attention mechanism allows sequential processing of text to compute a representation of the sequence and capture meaningful relationships between words in a corpus. Transformers allow for much more parallelization than CNNs and RNNs and make it possible to efficiently train very big models on large amounts of data on GPU clusters [2]. Reformers improve the efficiency of Transformers to train larger models on larger corpuses [4].

The experiments presented in the original Reformer paper [4] are performed only on generative and language model training tasks including imagenet64, enwik8-64K, and WMT 2014 English-to-German translation task. In this paper, we extend our experiment on text classification problems including News Classification, and Sentiment Classification compared to the published conference version [5]. We carry out the experiments on four datasets and use different settings to compare the models in terms of performance over training time. The experiments are done using Trax library, an end-to-end library for deep learning by Google Brain team with JAX backend [6].

We use for this study the integrated solution Mind in a Box Catalyst™, an On Premise and Fog computing device that features up to 4 GeForce RTX 2080 Ti for accelerated AI training and inference in its legacy version, and now up to 4 Nvidia Tesla V100 Cards.

2. Background

Transformers originally proposed by Vaswani, *et al.* [3] are used for building language models to learn contextual text representations. These language models can then be used for text generation and classification. Each Transformer block consists of a masked multi-head attention module, followed by a normalization layer and a position-wise feed forward layer (see [3] and Fig. 1). The attention module estimates the level of correlation between each word/character with the other elements using an attention vector. Then, it takes the weighted sum of the elements in this vector as the prediction of the target.

Transformers are more efficient than RNNs for sequential processing of text in parallel, however they are still memory-inefficient to train on long sequences and large models. Attempts to achieve more efficient versions of the Transformer model's self-attention mechanism include augmenting self-attention with persistent memory, adaptive attention span, factorized sparse representation of attention, and product-key attention [2]. The latest state of the art attempt is reported as Reformer model [4].

Reformer model [4] was proposed to solve the memory problem with Transformer models training using approximate attention computation based on locality-sensitive hashing and reversible layers [4]. With Locality Sensitive Hashing (LSH), nearby word embedding vectors get the same hash with high probability and the vectors that have the same hash are assigned to the same LSH bucket. Then LSH buckets are converted to sorted chunks (batches) to allow parallel processing. However, LSH buckets may be uneven in size which makes batching difficult. To tackle this, the batch size is set such that they may contain nearby hashes. Then attention is applied only to these batches and neighbor hash vectors rather than the whole input vector as being done with Transformers (see Fig. 2 and [4] for further details). In addition to LSH, Reformers use reversible layers mechanism, meaning instead of storing all the activations in memory, it recomputes the input of each layer on demand during the back propagation process. Two sets of activations are stored for each layer, one captures changes to the first layer and the other captures the last layer and then they are subtracted (see [4] and [7] for further details). This model has been reported to be memory and time efficient to train on long sequences and large models.

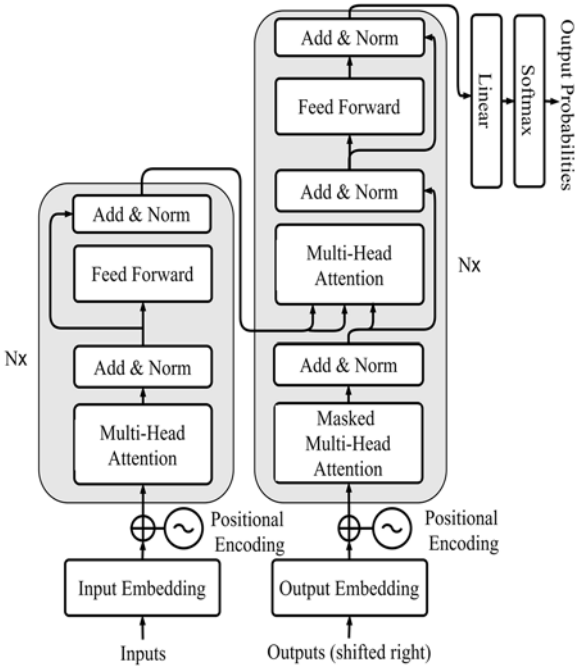


Fig. 1. The Transformer - model architecture [3].

3. Applicable Architectures

The experimental results in this paper rely on the integrated solution, Mind in a Box Catalyst™ with 4 GPU units, for balanced and accelerated training and inference capabilities in various text classification models (see Fig. 3). In this architecture, the objective is to integrate a pretrained language model on Open Data to learn the natural language structure. This model can read, understand, translate, summarize text with near human performance. Then this language model is used to do transfer learning for the target application of NLP on Big Data and generate Deep Data.

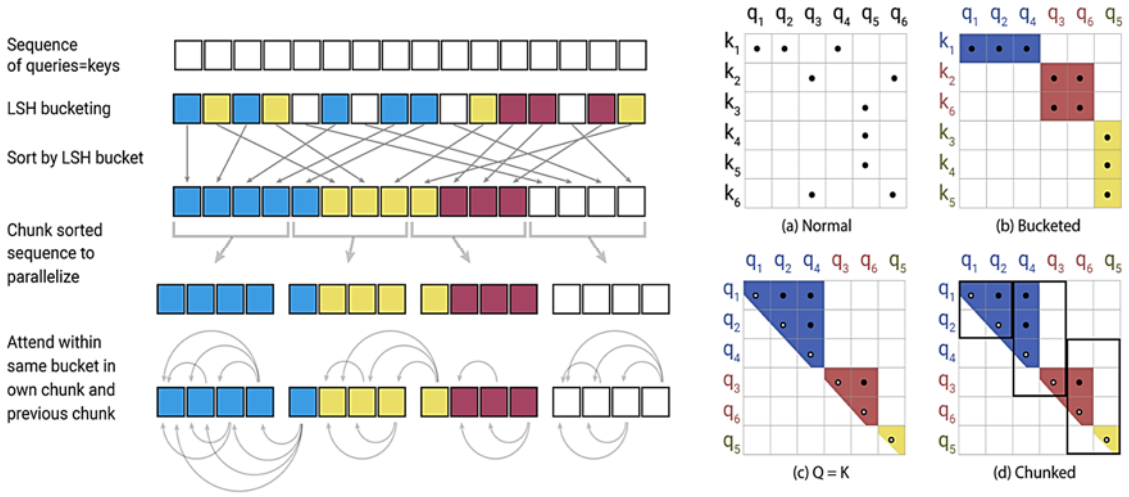


Fig. 2. Simplified depiction of LSH Attention showing the hash-bucketing, sorting, and chunking steps and the resulting causal attentions. (a-d) Attention matrices for these varieties of attention [4].

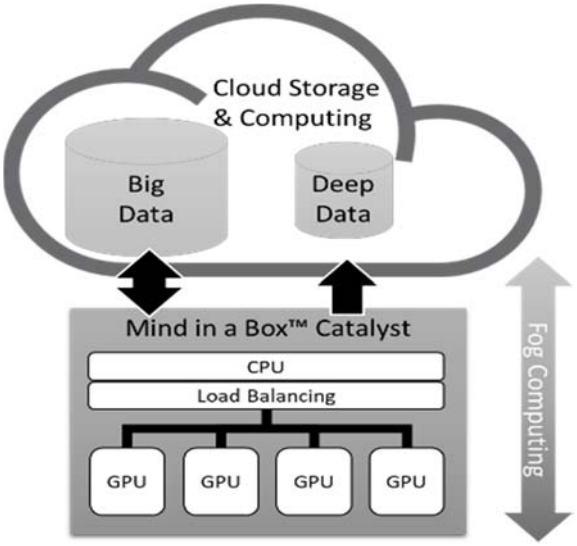


Fig. 3. On Premise/Fog AI functional architecture diagram.

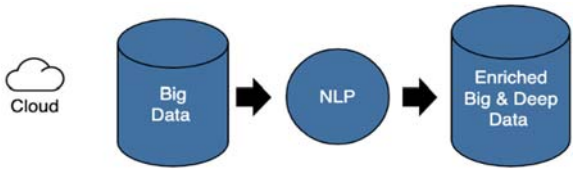


Fig. 4. Using NLP (text classification) in the cloud level to enrich big data.

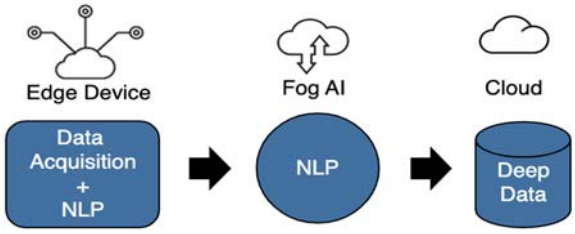


Fig. 5. Using NLP (text classification) on the edge level after data acquisition and before sending it to the fog layer. The data can be further analyzed with NLP (text classification) on the fog level before being sent to the cloud.

The objective sought through this experimental protocol and its underlying architecture is to assess the optimization and acceleration potential of the various models and libraries, to seek real time and / or high-performance computing capabilities. The technical common denominator here is an integration with the Python scripting language, which is currently extensively used to operate many Edge and Fog AI architectures, including the Fog AI appliance used for this experiment.

The applicable architectures for the performance considerations supported in our results are many. But the prime objective is to assess Hybrid Computing Deep Data generation strategies and in particular optimization of the choice of libraries and models depending on the data to be analyzed. Thus, relevant architecture would be Cloud, Edge and Fog AI architectures applying NLP techniques to generate Deep text Data and / or Enriched Big text Data, as illustrated in Fig. 4 and Fig. 5.

Fig. 4 shows a protocol to apply NLP algorithms to large amount of data (Big Data) to obtain Deep Data. This means that irrelevant information is removed, and data is stored in a structured and organized manner. In addition, NLP text classification algorithms can be applied to add labels or tags to the stored data to enrich it and improve its quality for the analysis applied in the next steps of processing the data.

If data is not yet stored as Big data in the cloud, a more efficient protocol can be used for its acquisition and storage. In Fig. 5, such protocol is presented where, data is processed with NLP in the edge device immediately after acquisition, and then sent to the fog layer. In Fog layer, further text processing and classification is performed on the data and when it is sent to the cloud for storage, it is structured and organized and only the relevant information is being

transmitted and stored. With this architecture, the computation is distributed over all three layers of the network and the computation load is reduced in cloud level which in turn reduces the transmission load and latency.

4. Experiments and Results

The experiments presented in the original Reformer paper [4] are performed only on generative tasks and no results of experiments have been presented for comparing Reformers with Transformers on text classification task. In this paper, four datasets are used as Open Data for the experiments on text classification tasks. They are as follows: IMDB and Yelp for binary sentiment classification, Allocine for large-scale sentiment analysis in French, and AGNews for four-class topic classification problem.

For the experiments, we build a Reformer and a Transformer encoder for binary and multiclass classification. A caveat to mention for this implementation is that the Reformer model does not use LSH Attention as referenced in the original Reformer paper. Rather, we use a memory efficient self-attention layer implemented by the Trax team. This efficient self-attention layer has two main differences when compared with the standard self-attention layer:

1. The query weights Q and the key weights K are shared in the dot-production attention which lowers memory consumption.

2. Attention masking is implemented by associating an integer – typically the sequence position - with each query and key vector and defining a function to compute attention masks from this information. The standard attention in comparison is less memory efficient because it instantiates $O(n^2)$-size attention masks.

Overall, the Reformer we use is still much more memory efficient than a Transformer encoder model because of the points mentioned above but also because it uses reversible layers. Reversible layers allow the model to save memory on GPUs and TPUs since we can re-compute inputs for backward propagation, instead of having to store them during the forward propagation. This alone saves a great deal of memory.

Different settings are used to tune the hyperparameters of Transformers and Reformers including optimizers (Adam and AdaFactor [8]), number of layers, number of attention heads, attention type in Reformers, the effect of vocabulary size, the number of features in the encoder inputs (model dimension), and maximum length of the sequences.

It is worth noting that, the length of attention key and value vectors that exists only for Reformer is set to 64.

In experiments of this paper, the performance is compared in terms of minimum testing cross entropy (Min CE), maximum test accuracy (per batch of text) and training time in seconds.

4.1. Adam and Adafactor Optimizers

We start the experiments by comparing the Adam and Adafactor optimizers. In Table 1, results are shown for comparing 2 layers of Transformer and Reformer encoder blocks and 32 attention heads ended with a dense layer and a softmax layer.

The results show that Adafactor performs better than Adam optimizer for both Transformer and Reformers. Therefore, for the rest of the experiments we use AdaFactor optimizer.

Table 1. Results of experiments with Transformers (TR) and Reformers (RF) (2 layers of TR/FR encoder blocks and 32 attention heads, a dense layer and a softmax layer) to compare Adam and AdaFactor optimization algorithms.

Dataset	Optimiser	Train Steps		Dropout rate		Train CE		Test CE		Test Accuracy		Train Time (sec)		Inference Time (sec)	
		TR	RF	TR	RF	TR	RF	TR	RF	TR	RF	TR	RF	TR	RF
IMDB	**Adam**	3000	5000	0.3	0.3	0.308	0.377	0.347	0.395	0.833	0.868	**1491.872**	1084.734	**2.42**	**3.84**
IMDB	**AdaFactor**	3000	5000	0.3	0.3	**0.246**	**0.321**	**0.324**	**0.362**	**0.90**	**0.885**	1540.54	**1068.767**	2.726	3.853
Yelp	**Adam**	5000	10000	0.2	0.2	0.323	**0.271**	0.308	0.276	0.871	0.886	558.18	2684.084	**2.329**	**3.711**
Yelp	**AdaFactor**	5000	10000	0.2	0.3	**0.212**	0.277	**0.202**	**0.219**	**0.920**	**0.910**	**533.50**	**2158.91**	2.405	3.792
AGNews	**Adam**	10000	10000	0.2	0.2	0.271	**0.245**	0.245	0.234	0.893	0.918	1201.45	**2192.94**	**2.454**	4.087
AGNews	**AdaFactor**	10000	10000	0.2	0.2	**0.209**	0.251	**0.219**	**0.203**	**0.910**	**0.935**	**1146.337**	2307.161	2.569	**3.912**

4.2. Attention Type in Reformers

Since we have the option of using standard self-attention layers as well as a more memory efficient implementation, we want to test if using a more efficient model will have an impact on our classification task's performance. We trained the Reformer model using two different approaches.

1. Alternating between memory efficient self-attention layers, as well as standard self-attention layers, for a total of six layers. We will referrer to this as a model having hybrid self-attention layers.

2. Using six memory-efficient-only self-attention layers. This approach will be referred to as efficient self-attention layers.

For these two approaches, we compared results using both 8k and 32k vocabulary sizes.

Results in Table 2 shows that:

- In almost all cases, it did not make a notable difference in accuracy to use either hybrid or efficient self-attention layers.
- In terms of training time, the impact was negligible, and on average efficient self-attention layers trained slightly faster.

In conclusion it is more beneficial to use the efficient version if memory is a concern since it will not impact performance or training time.

4.3. Number of Attention Head

In Table 3, we analyze the performance of the Transformers and Reformers when changing the number of attention heads. As shown, the number of attention heads tested are 6, 16 and 32.

These results show that increasing the number of attention heads improved the performance of these classifiers because it allows for better capturing the underlying context of the text.

Table 2. Results of experiments with Reformers (RF) to compare the effect of attention type on performance.

Dataset	Model	Vocab	D_Model	Encloder layers	Number of heads	Max length	Attention type	Training time	Max accuracy	Min CE
IMDB	Reformer	8k	512	6	6	512	hybrid	3168	87.27	0.33
	Reformer	8k	512	6	6	512	efficient	3024	**87.99**	**0.32**
	Reformer	32k	512	6	6	512	hybrid	3211	**88.44**	**0.30**
	Reformer	32k	512	6	6	512	efficient	3150	85.94	0.33
Yelp	Reformer	8k	512	6	6	512	hybrid	2997	91.02	0.21
	Reformer	8k	512	6	6	512	efficient	2917	**92.32**	0.17
	Reformer	32k	512	6	6	512	hybrid	3057	**93.83**	0.16
	Reformer	32k	512	6	6	512	efficient	3147	93.44	0.16
Allocine	Reformer	32k	512	6	6	512	hybrid	3048	93.3	0.18
	Reformer	32k	512	6	6	512	efficient	3090	**93.67**	0.18
AGNews	Reformer	8k	512	6	6	512	hybrid	2640	90.78	0.24
	Reformer	8k	512	6	6	512	efficient	2552	**90.94**	0.25
	Reformer	32k	512	6	6	512	hybrid	2593	**92.97**	0.22
	Reformer	32k	512	6	6	512	efficient	2657	92.73	0.21

Table 3. Results of experiments with Transformers (TR) and Reformers (RF) to compare the effect of the number of attention heads on performance.

Dataset	Model	Vocab	D_Model	Encloder layers	Number of heads	Max length	Attention type	Training time	Max accuracy	Min CE
IMDB	Reformer	8k	512	6	6	512	hybrid	3168	87.27	0.3323
	Reformer	8k	512	6	16	512	hybrid	5400	87.27	0.3232
	Reformer	8k	512	6	32	512	hybrid	6120	**88.44**	0.29
	Transformer	8k	512	6	6	512		2324.8	87.42	0.31
	Transformer	8k	512	6	16	512		2647	85.50	0.33
	Transformer	8k	512	6	32	512		2476	**89.06**	0.27
Yelp	Reformer	8k	512	6	6	512	hybrid	2997.6	91.02	0.21
	Reformer	8k	512	6	16	512	hybrid	4500	91.8	0.23
	Reformer	8k	512	6	32	512	hybrid	3385	**92.50**	0.19
	Transformer	8k	512	6	6	512		2425	**92.42**	0.19
	Transformer	8k	512	6	16	512		2627	90.0	0.23
	Transformer	8k	512	6	32	512		2388	92.27	0.20
Allocine	Reformer	8k	512	6	6	512	hybrid	3048	**93.3**	0.18
	Reformer	8k	512	6	16	512	hybrid	4440	92.11	0.20
	Reformer	8k	512	6	32	512	hybrid	3152	92.58	0.19
	Transformer	8k	512	6	6	512		2333	92.66	0.19
	Transformer	8k	512	6	16	512		2591	**95.08**	0.17
	Transformer	8k	512	6	32	512		1796	93.12	0.16
AGNews	Reformer	8k	512	6	6	512	hybrid	2640	90.78	0.24
	Reformer	8k	512	6	16	512	hybrid	3601	90.63	0.26
	Reformer	8k	512	6	32	512	hybrid	2322	**91.33**	0.28
	Transformer	8k	512	6	6	512		1860	90.94	0.25
	Transformer	8k	512	6	16	512		1911	90.94	0.26
	Transformer	8k	512	6	32	512		1469	**91.25**	0.24

4.4. Effect of Vocabulary Size

We are interested in analyzing the effect of pretrained vocabulary size on classification performance. All vocabularies used are subword implementations. 8k.subword and 32k.subword are both built using Tensor2Tensor's Subword Text Encoder class. Subword Text Encoder from Tensor2Tensor works similarly to a Byte Pair Encoding (BPE) tokenizer, but on top of splitting to subwords, it also does tokenization of spaces and punctuation. In other words, unlike BPE, it also

encodes the spaces – or absence of spaces – so that the raw sequence is fully reproducible. This is not the case with BPE tokenizer.

Results of experiments in Table 4 shows that:

- The 32k subword vocabulary does not consistently provide better results than the 8k subword vocabulary for all the datasets.
- Training time is usually slightly higher for 32k subword but not significantly.

Table 4. Results of experiments with Transformers and Reformers to compare the effect of vocab on performance.

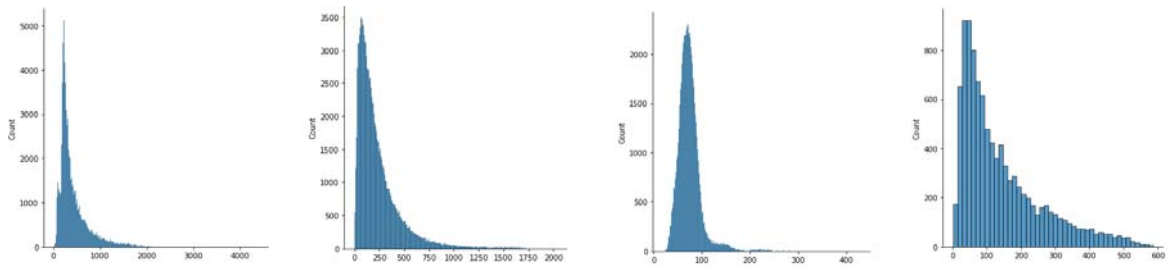
Dataset	Model	Vocab	D_Model	Encloder layers	Number of heads	Max length	Attention type	Training time	Max accuracy	Min CE
IMDB	Reformer	8k	512	6	6	512	hybrid	3168	**87.27**	0.33
	Reformer	32k	512	6	6	512	hybrid	6048	83.75	0.33
	Reformer	8k	512	6	6	512	hybrid	3211	88.44	0.30
	Reformer	32k	512	6	6	512	hybrid	10260	**89.14**	0.30
	Transformer	8k	512	6	6	512		2325	87.42	0.31
	Transformer	32k	512	6	6	512		2351	**87.94**	0.27
	Transformer	8k	512	6	6	512		2451	**89.69**	0.27
	Transformer	32k	512	6	6	512		3182	88.44	0.31
Yelp	Reformer	8k	512	6	6	512	hybrid	2998	**0.91**	0.21
	Reformer	32k	512	6	6	512	hybrid	5123	0.90	0.23
	Reformer	8k	512	6	6	512	hybrid	3057	**93.83**	0.16
	Reformer	32k	512	6	6	512	hybrid	10740	91.64	0.18
	Transformer	8k	512	6	6	512		2425	92.42	0.19
	Transformer	32k	512	6	6	512		2265	**92.66**	0.23
	Transformer	8k	512	6	6	512		2373	**93.98**	0.16
	Transformer	32k	512	6	6	512		2350	93.13	0.18
AGNews	Reformer	8k	512	6	6	512	hybrid	2640	**90.78**	0.24
	Reformer	32k	512	6	6	512	hybrid	8700	89.50	0.30
	Reformer	8k	512	6	6	512	hybrid	2657	**92.73**	0.21
	Reformer	32k	512	6	6	512	hybrid	9180	91.56	0.24
	Transformer	8k	512	6	6	512		1860	90.94	0.25
	Transformer	32k	512	6	6	512		1921	**91.72**	0.26
	Transformer	8k	512	6	6	512		1981	92.5	0.23
	Transformer	32k	512	6	6	512		1920	**92.97**	0.23

4.5. Effect of Maximum Length of the Sequences

Since the Reformer model is more memory efficient, we wanted to test whether increasing the maximum length of the sequences that will be read has an impact on performance.

We experimented with both shorter and longer sequence lengths (256, 512, 1024) and analyzed results comparing the Reformer encoder model with the Transformer encoder.

We also compared the maximum length hyperparameter with the average length of the sequences for each dataset to see if there is a correlation between the two. Table 5 shows the results and Fig. 6 shows the histogram of maximum length of the sequences in all datasets used in this paper. The average sequence length for IMDB dataset is 416 and the results show that a maximum length of 512 clearly performed better than lengths 256 and 1024 for both the Reformer and the Transformer model. The average sequence length for Yelp is 230. For Yelp dataset there was no clear pattern since sequence length of 1024 performed better for the Reformer model, and sequence length 512 performed the best for the Transformer model. The average sequence length for Allocine is 141. In this case, 256 also performed better on average than lengths 512 or 1024. The average sequence length for AGnews is 78. In this case, a maximum length of 256 performed better than 512 or 1024.



(a) IMDB (b) Yelp (c) Allocine (d) AGNews

Fig. 6. Sequence length distribution of the text samples in datasets.

Table 5. Results of experiments with Transformers and Reformers to compare the effect of max length on performance.

Dataset	Model	Vocab	D_Model	Encloder layers	Number of heads	Max length	Attention type	Training time	Max accuracy	Min CE
IMDB	Reformer	8k	512	6	6	256	hybrid	1633	83.20	0.44
	Reformer	8k	512	6	6	512	hybrid	3168	**87.27**	0.33
	Reformer	8k	512	6	6	1024	hybrid	2520	84.34	0.38
	Transformer	8k	512	6	6	256		1139	83.83	0.47
	Transformer	8k	512	6	6	512		2325	**87.42**	0.31
	Transformer	8k	512	6	6	1024		1918	79.69	0.46
Yelp	Reformer	8k	512	6	6	256	hybrid	1769	91.09	0.18
	Reformer	8k	512	6	6	512	hybrid	2998	91.02	0.21
	Reformer	8k	512	6	6	1024	hybrid	2777	**92.42**	0.23
	Transformer	8k	512	6	6	256		1297	91.95	0.22
	Transformer	8k	512	6	6	512		2425	**92.42**	0.19
	Transformer	8k	512	6	6	1024		2134	91.33	0.22
Allocine	Reformer	32k	512	6	6	256	hybrid	1888	**93.23**	0.16
	Reformer	32k	512	6	6	512	hybrid	3048	93.2	0.18
	Reformer	32k	512	6	6	1024	hybrid	2929	92.63	0.19
	Transformer	32k	512	6	6	256		1557	**93.05**	0.18
	Transformer	32k	512	6	6	512		2333	92.66	0.19
	Transformer	32k	512	6	6	1024		2312	91.51	0.20
AGNews	Reformer	8k	512	6	6	256	hybrid	2474	**91.71**	0.25
	Reformer	8k	512	6	6	512	hybrid	2640	90.78	0.24
	Reformer	8k	512	6	6	1024	hybrid	2592	91.41	0.24
	Transformer	8k	512	6	6	256		1921	**92.11**	0.26
	Transformer	8k	512	6	6	512		1860	90.94	0.25
	Transformer	8k	512	6	6	1024		1887	90.78	0.27

4.6. Effect of the Model Dimension

We experiment with 256, 512 and 1024 as the number of features so that each representation of subwords of our vocabulary contains more information. Based on the results in Table 6, depending on the dataset, and the model used, a dimension of 512 or 1024 will yield a better performance and the dimension of size 256 perform worse on average.

Table 6. Results of experiments with Transformers and Reformers to compare the effect of model dimension on performance.

Dataset	Model	Vocab	D_Model	Encloder layers	Number of heads	Max length	Attention type	Training time	Max accuracy	Min CE
IMDB	Reformer	8k	512	6	6	512	hybrid	3168	87.27	0.33
	Reformer	8k	1024	6	6	512	hybrid	6048	83.75	0.33
	Reformer	32k	512	6	6	512	hybrid	3211	88.44	0.30
	Reformer	32k	1024	6	6	512	hybrid	10260	89.14	0.30
	Transformer	8k	512	6	6	512		2325	87.42	0.31
	Transformer	8k	1024	6	6	512		2351	87.94	0.27
	Transformer	32k	512	6	6	512		2451	89.69	0.27
	Transformer	32k	1024	6	6	512		3182	88.44	0.31
Yelp	Reformer	8k	512	6	6	512	hybrid	2998	0.91	0.21
	Reformer	8k	1024	6	6	512	hybrid	5123	90.23	0.23
	Reformer	32k	512	6	6	512	hybrid	3057	93.83	0.16
	Reformer	32k	1024	6	6	512	hybrid	10740	91.64	0.18
	Transformer	8k	512	6	6	512		2425	92.42	0.19
	Transformer	8k	1024	6	6	512		2266	92.66	0.20
	Transformer	32k	512	6	6	512		2373	93.98	0.16
	Transformer	32k	1024	6	6	512		2350	93.13	0.18
Allocine	Reformer	8k	512	6	6	512	hybrid	3048	93.3	0.18
	Reformer	8k	1024	6	6	512	hybrid	7740	93.2	0.18
	Transformer	32k	512	6	6	512		2333	92.66	0.19
	Transformer	32k	1024	6	6	512		3097	92.42	0.177
AGNews	Reformer	8k	512	6	6	512	hybrid	2640	90.78	0.24
	Reformer	8k	1024	6	6	512	hybrid	8700	89.50	0.30
	Reformer	32k	512	6	6	512	hybrid	2657	92.73	0.21
	Reformer	32k	1024	6	6	512	hybrid	9180	91.56	0.24
	Transformer	8k	512	6	6	512		1860	90.94	0.25
	Transformer	8k	1024	6	6	512		1921	91.72	0.26
	Transformer	32k	512	6	6	512		1981	92.5	0.23
	Transformer	32k	1024	6	6	512		1920	92.97	0.23

4.7. Best Performance of Transformers and Reformers

In Table 7, we present the top 3 performance of Transformers and Reformers among all the experiments that were done under different hyper-parameter settings. In the case of IMDB, Yelp, and Allocine datasets, Transformers show the best performance in terms of accuracy. For AGNews however, the best performance is achieved by both Transformers and Reformers in terms of accuracy and in this case, the Reformer takes less time to train.

Table 7. Results of experiments with Transformers and Reformers to compare their top three performance.

Dataset	Model	Vocab	D_Model	Encloder layers	Number of heads	Max length	Attention type	Training time	Max accuracy	Min CE
IMDB	Reformer	8k	512	6	32	512	hybrid	6120	88.44	0.29
	Reformer	32k	512	6	6	512	hybrid	3211	88.44	0.30
	Reformer	8k	512	6	12	512	hybrid	10260	89.14	0.30
	Transformer	32k	1024	6	12	512		3182	88.44	0.31
	Transformer	8k	512	6	6	512		2476	89.06	0.27
	Transformer	32k	512	6	6	512		2451	**89.69**	0.27
Yelp	Reformer	32k	512	6	6	512	hybrid	2999	93.2	0.17
	Reformer	32k	512	6	6	512	efficient	3147	93.44	0.16
	Reformer	32k	512	6	6	512	hybrid	3057	93.83	0.16
	Transformer	32k	1024	6	12	512		2365	93.13	0.19
	Transformer	32k	512	6	6	512		2373	93.98	0.16
	Transformer	8k	512	8	6	512		3172	**95.16**	0.12
Allocine	Reformer	32k	512	6	6	256	hybrid	1888	93.23	0.16
	Reformer	32k	512	6	6	512	hybrid	3048	93.3	0.18
	Reformer	32k	512	6	6	256	efficient	3090	93.67	0.18
	Transformer	32k	512	6	6	256		1557	93.05	0.18
	Transformer	8k	512	6	32	512		1796	93.12	0.16
	Transformer	8k	512	6	16	512		2591	**95.08**	0.17
AGNews	Reformer	8k	512	6	6	256	hybrid	2474	91.71	0.25
	Reformer	32k	512	6	6	512	hybrid	2657	92.73	0.21
	Reformer	32k	512	6	6	512	efficient	2593	**92.97**	0.22
	Transformer	32k	512	6	6	512		1981	92.5	0.23
	Transformer	32k	512	6	6	512		2450	92.89	0.21
	Transformer	32k	512	8	6	1024		**1920**	**92.97**	0.23

5. Conclusions

In this paper, we compared Transformers and Reformers for text classification application. The results showed that increasing the number of attention heads improved the performance of both Transformers and Reformers because it allows for better capturing the underlying context of the text. Also, we observed that higher number of features results in better performance and setting the maximum length of sequences closer to the average length of sequence in the dataset results in better performance. In literature, Reformers have shown to outperform Transformers in terms of memory and time efficiency at the similar level of accuracy on language model training and text generative tasks. However, the experiments in this paper showed that Transformers might be preferred to Reformers for Text classification application with short text sequences and smaller models. The benefit of Reformers versus Transformers was observed to be the possibility of training larger models. Therefore, Reformers may still be a better option for training our language model on big corpus Open Data and do transfer-learning for the applications in hand. Benchmarking text classification using the language models based on Reformers and Transformers for the text classification is the future work related to this article.

Acknowledgements

This work was supported by the National Research Council through their Industrial Research Assistance Program (IRAP) and abide by their experimental research requirements.

References

[1]. Deep text analytics. (https://www.poolparty.biz/what-is-deep-text-analytics/)

[2]. S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, J. Gao, Deep learning based text classification: A comprehensive review, *arXiv*:2004.03705, 2020.

[3]. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in *Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS' 2017),* 2017, pp. 5998–6008.

[4]. N. Kitaev, L. Kaiser, A. Levskaya, Reformer: The efficient transformer, in *Proceedings of the*

International Conference on Learning Representations, 2020.

[5]. R. Soleymani, J. Farret, Text Classification with Transformers and Reformers for Deep Text Data, in *Proceedings of the 2nd International Conference on Advances in Signal Processing and Artificial Intelligence (ASPAI' 2020)*, Berlin, Germany, November 2020, pp. 239-243.

[6]. Trax, an end-to-end library for deep learning by Google Brain team. (https://github.com/google/trax).

[7]. A. N. Gomez, M. Ren, R. Urtasun, R. B. Grosse, The reversible residual network: Backpropagation without storing activations, in *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA*, 2017, pp. 2214-2224.

[8]. N. Shazeer, M. Stern, Adafactor: Adaptive learning rates with sublinear memory cost, *arXiv*:1804.04235, 2018.