AI6127 Project:
A Study on Linear Attention Mechanism for NLP tasks

Chan Li Long
School of Computer Science and Engineering
Nanyang Technological University
lchan025@e.ntu.edu.sg | G2202252L

Ong Wen Qing
School of Computer Science and Engineering
Nanyang Technological University
ongw0111@e.ntu.edu.sg | G2202256K

Abstract

The transformer has largely overtaken RNNs for NLP tasks as they scale better with data and the architecture is highly parallelizable. The original transformer uses the softmax attention which is $O(n^2)$ in computational and memory complexity. In this study, approximations of the original softmax attention is explored and the effects of swapping full softmax attention with linear attention is studied. Naively swapping the full attention with linear attention reduces accuracy by 6-9% but potentially increases inference speed by 416% (Input Length: 1024). LoRA[3] can be used to align a model that has its full attention layers swaped with linear attention layers. The task of text classification of amazon reviews is used for this study.

1 Introduction

The original softmax attention has a computational and memory complexity of $O(n^2)$ and this would mean that large pretrained models which is typically more performant will not be able to run on consumer GPUs with large input length. In this case n is the sentence length and an attention mechanism with $O(n)$ would greatly lower the barrier to entry for general NLP applications.

The purpose of this report is to document the study done on swapping full softmax attention with linear attention on a text classification task.

This report will first introduce a linear attention mechanism that approximates the softmax attention and how they can lower memory and computational complexity to $O(n)$. The experiment design and model architecture will then be explained. The results will show the effects of linear attention and remarks by the authors on the efficacy of linear attention will be made in the conclusion.

2 Linear Attention Mechanism

A few papers have attempted to reduce the $O(n^2)$ complexity to $O(n)$ using kernel function ϕ that approximates the softmax attention. What makes the original softmax attention slow is that the matrix $QK^T \in \mathbb{R}^{n \times n}$ is explicitly computed. One work [4] formulated softmax approximation in Equation 1 where $\text{elu}(x)$ is the exponential linear unit.

$$\begin{gathered} softmax\left(\frac{QK^T}{\sqrt{D}}\right)V \approx \phi(Q')\Big(\phi(K')^T V\Big) \\ \phi(x) = \text{elu}(x) + 1 \\ Q, K, V \in \mathbb{R}^{b \times n \times d} \end{gathered} \tag{1}$$

We can plot the error distribution between the full and kernel approximated softmax by giving a random query and key drawn from the gaussian distribution $Q, K \sim \mathcal{N}(0, 0.2I)$. The zero centered error distribution with small standard

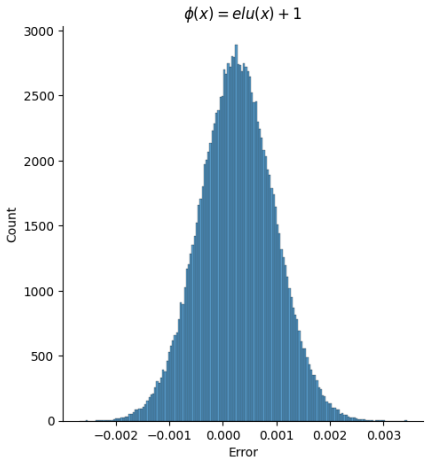

Figure 1: Error distribution between full softmax attention and its kernel approximation in Equation 1. The mean is 0.0003 and the standard deviation is 0.0007.

```python
a,b = torch.zeros((1,1024,128)),0.2*torch.ones((1,1024,128))
Q,K = torch.normal(a,b),torch.normal(a,b)
V = K

#================[FULL SM ATTENTION]=======================#
QK = torch.einsum("nld,nkd->nlk",Q,K)/math.sqrt(Q.shape[-1]) #+ mask_l
sm_QK = torch.softmax(QK,dim=-1)
sm_attn1 = torch.einsum("nlk,nkd->nld",sm_QK,V)

#================[APPROX ATTENTION]=======================#
feat_proj = lambda x: torch.nn.functional.elu(x)+1

Q_lin = feat_proj(Q)
K_lin = feat_proj(K)

Z = 1/(torch.einsum("nld,nd->nl", Q_lin, K_lin.sum(1))+1e-10)

KV = torch.einsum("nsd,nsm->nmd", K_lin, V)

V_lin2 = torch.einsum("nld,nmd,nl->nlm", Q_lin, KV, Z)
#================[ERROR CALCULATION]=======================#
error = (V_lin2-sm_attn1).flatten()
```

Figure 2: Code to recreate the softmax error distribution

deviation suggests that the kernel approximation could be naively swapped with the full softmax attention. We will test this hypothesis out in later sections of this report.

1000 Runs Speed Test | Token Length: 1024		
	Full Softmax	**Kernel Approx**
Time Taken (s)	8.88	2.66

Table 1: Time taken to run 1000 runs of attention calculation. Kernel approximation is 3.33 times faster than the full attention calculation.

In Table 2, a speed test is conducted to see how much faster the kernel approximation is compared to the full attention. In our test the kernel softmax approximation is 3.33 times faster than the full attention calculation. This main speedup is due to the $n \times n$ matrix not being explicitly calculated and that $\left(\phi(K')^T V\right) \in \mathbb{R}^{d \times d}$ is calculated first. Since for most long sequence cases $n >> d$, we expect a computational speedup when using kernel approximations for the calculation of attention.

3 Experiment and Model Architecture

This section will describe the experiments conducted in this report. The task will be to fit models on the Amazon Reviews dataset and the classification accuracy of sentiments will be used to compare Linear Attention with Full Attention. The Fast-Transformers library [4] is used and the addition of LoRA is added manually in pytorch.

Statistics on Amazon Reviews Dataset	
Median Sentence Length	66
Max Sentence Length	254
# Positive Class	2,000,000
# Negative Class	2,000,000
# Samples for Train	1,000,000
# Samples for Validation	400,000
# Vocab	50,000

Table 2: Dataset Statistics

Transformer Architecture	
Features	**Attributes**
Type	Encoder Only
# Parameters	14,382,338
# Encoder Layers	3
# Heads	8
Embedding Dimension	256
Feed Forward Dimension	512
Masking	No causal masking is used
Feature Aggregation (Class Head)	Max Pool

Training Setup	
Optimizer	AdamW
Learning Rate	1e-04
Weight Decay	1e-04
Epochs	30
Batch Size	64

Table 3: Left: Transformer Architecture used in experiments, Right: Training Configuration used for the experiments in this report.

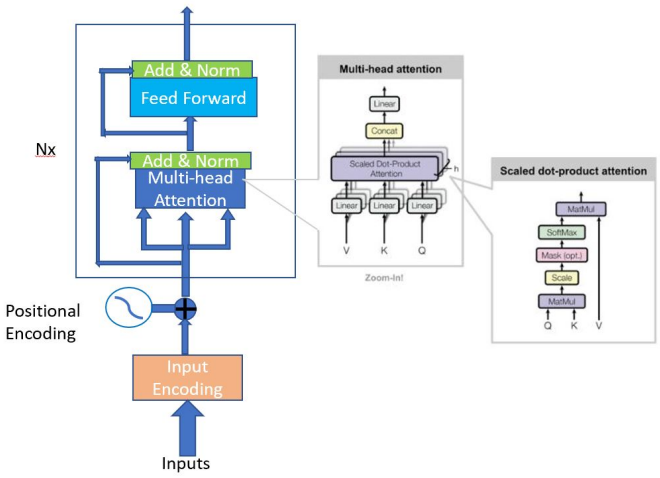

Figure 3: Transformer Architecture with detail illustration of the attention mechanism[5]

4 Results

Naive swapping of Full Attention to Linear Attention results in a decrease in accuracy due to accumulation of errors. Table 4 shows that naively swapping all three attention layers detailed in Table 3 without retraining will lead to a decrease in inference accuracy. It is hypothesized that the kernel approximation errors detailed in Equation 1 propagates throughout the model during the forward pass and leads to a decrease in accuracy if all attention mechanism are swapped.

Effects of big data: Training with larger dataset leads to higher test set performance. In table 4, the model that is trained with 3.6 million samples performs significantly better than models trained with just 1 million samples even though only 3/30 epochs are completed. This shows the ability of transformers to scale with data as expected from other experiments with large pretrained transformers like GPT [1].

Swapping later layers is better than swapping front layers. The model used in this project consist of stacking 3 sequential transformer encoders. In table 4, it is hypothesized that the decrease in accuracy is due to error getting propagated during the forward pass. In table 4 it can be seen that swapping full softmax attention with linear attention in the later layers will achieve better performance than swapping the front layers. We know that from other experiments in computer vision that the front layers processes fine grained information while the later layers processes coarse grained global information. One explanation for this result could be that the errors in processing fine grained information carries a heavier penalty on performance than errors in processing coarse grained information.

Using LoRA can help align swapped attention models with little cost. LoRA [3] is a method to fine tune a transformer model by learning a difference ΔW applied to the query,key,value and output projection matrices W_q, W_k, W_v, W_o of a typical self-attention layer. Crucially during the fine tuning step we only train ΔW which have a much smaller number of parameters to train as compared to the entire model. LoRA also provides the benefit of not ading additional parameters during inference as the old projection matrix can be fused with ΔW to form a new weight matrix that is of the same size as the original weights $W^{new} = W^{old} + \Delta W$. Table 4 shows that LoRA can help to recover the model's accuracy after swapping the full attention layers for linear attention. This might hint that it is possible to easily fine tune large language models with its attention mechanism swapped so that it can infer on sentences with very large length.

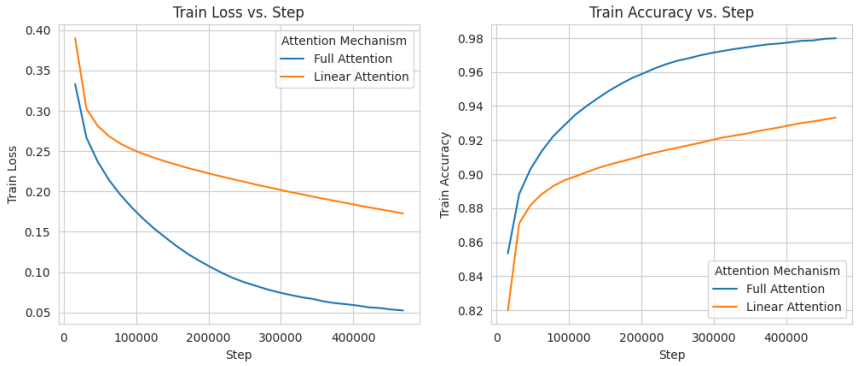

Figure 4: Train Loss and Train Accuracy for Full Attention Model (Blue Line) and Linear Attention Model (Orange Line) on the training dataset

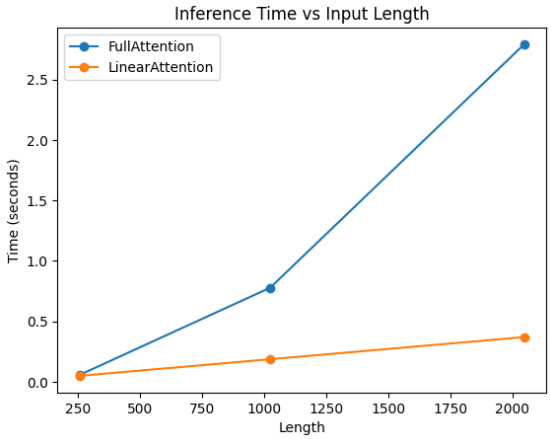

Figure 5: Time Inference Vs Input length for the Full Attention and Linear Attention

Evaluation Metrics to Analyze the Attention Mechanism Used		
	Full Attention	Linear Attention
Training Accuracy	93.3%	93.2%
Validation Accuracy	60.2%	59.5%

Table 4: Evaluation Metrics for Attention mechanism Analysis

Time inference speed between Full attention and linear attention(Length:1024)		
	Full Attention	Linear Attention
Time (s)	0.777	0.187

Table 5: The time inference speed between Full Attention and Linear Attention are 0.777 sec and 0.187 sec respectively. Linear attention model performance 416% faster than the Full attention model due to the reduction of the $O(n^2)$ complexity to $O(n)$ using kernel function ϕ that approximates the softmax attention show in section 2 of the report.

Test Set Accuracy | Swapping Attention		
	Attention Mechanism	
Model Trained With	**Linear Attention**	**Full Attention**
Linear Attention (1M Samples)	60.5%	57.9%
Full Attention (1M Samples)	57.9%	61.4%
Full Attention (3.6M Samples*)	62.2%	68.8%

Table 6: Results of naively swapping attention mechanism after training. Purpose is to measure the effects of swapping attention mechanisms after training. Model trained with 3.6M samples is only trained for 3 epochs due to limited GPU resources and time constraint.

Test Set Accuracy		
Model Trained With	**Swap 1st Layer**	**Swap first 2 Layers**
Full Attention (1M Samples)	59.2%	58.9%
Model Trained With	**Swap Last Layer**	**Swap Last 2 Layers**
Full Attention (1M Samples)	61.2%	60.2%

Table 7: Results of naively swapping only some layers with linear attention after training.

Test Accuracy when using LoRA			
Full	**Linear**	**Naive Swap**	**r = 32**
61.4%	60.5%	57.9%	60.4%
# Parameters to Train			
Retraining All		**LoRA r=32**	
14,382,338		196,608 (1.36%)	

Table 8: Test Set Accuracy. Full: Performance of Full softmax model. Linear: Performance of Linear attention model. Naive Swap: Swapping full attention layers with linear attention layers. r=32 : Swapping all layers with linear attention layers and retraining with LoRA for r=32. Using LoRA only requires training of 196,608 parameters (1.36%).

5 Conclusion

The purpose of this report is to document the study done on swapping full softmax attention with linear attention on a text classification task. In the experiments conducted in this report, naively swapping all attention mechanisms in the transformer encoder results in up to 9.5% decrease in performance but a 416+% increase in inference speed for a token length of 1024. It is also shown that swapping attention mechanisms in the later layers is better than swapping attention mechanism in the earlier layers as the earlier layers might be processing on fine grained information and that disturbing this fine grained transformation is more detrimental to performance than disturbing coarse grained transformation which typically happens in later layers.

Softmax kernel approximation will induce errors that propagate throughout the model and will reduce performance when full attention layers are naively swapped with linear attention layers. This error propagation is also mentioned in the performer paper [2] where the authors proposed another softmax approximation method that is backwards compatible with full softmax attention but requires retraining. The experiments in this report suggests that only learning a new small set of weights ΔW (using LoRA [3]) is required to align a model that has its full attention layers swapped with linear attention.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[3] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[4] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[5] Momina Qazi, Muhammad Usman Shahid Khan, and Mazhar Ali. Detection of fake news using transformer model. pages 1–6, 01 2020.