Efficient Attention

And the Chronicles of LoRA the Transformer

Sahil Chaudhary* Mrigank Pawagi*
Rohit Jorige* Nagasai Jajapuram*

WorthYourAttention

*Equal contribution.

Indian Institute of Science

12 April 2024

Introduction

A long long time ago... RNN's and LSTM's were the go-to model for sequence to sequence models. But they had a few flaws:

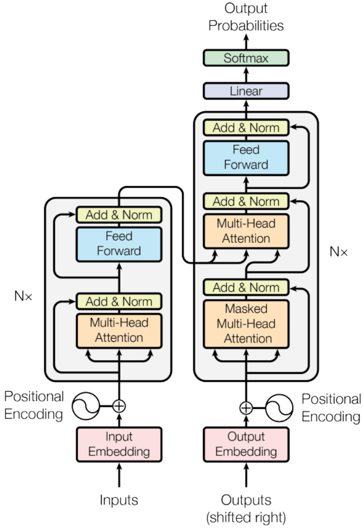
- Sequential Processing
- They do not accurately capture long term dependencies (not LSTM)
- They were slow
- Hard to parallelize
- Past information retained through past hidden states
- Vanishing gradients

Attention in Transformers

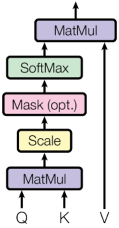
Overview of the architecture

Key Innovations in Transformers:

- **Non-sequential Processing:** Sentences are processed as a whole, not word by word.
- **Self Attention:** A new ′unit′ for computing word similarity scores within a sentence.
- **Positional Embeddings:** Replaces recurrence with fixed or learned weights encoding token position information.

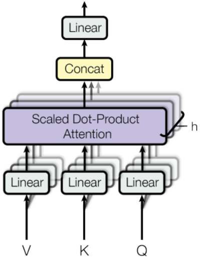


Scaled Dot-Product Attention



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right) V$$

Multi-Head Attention



$$\text{MultiHead}(Q, K, V) = \text{Concat}(Head_1, \ldots, Head_h) W_O$$

English to Kannada Translation

English Prompt	Kannada translation	Correct Kannada Translation
I cannot stand this smell	ನಾನು ಅಂತರ ಸಂಗತಿ ನಾನು ಕೊಡುವುದಿಲ್ಲ	ನಾನು ಈ ವಾಸನೆಯನ್ನು ಸಹಿಸುವುದಿಲ್ಲ
I am here.	ನಾನು ಕೇಳಿದ್ದೇನೆ	ನಾನು ಇಲ್ಲಿದ್ದೇನೆ
what should we do?	ಏನು ಮಾಡಬೇಕು?	-

Transformer Formal Definition

Let $x \in \mathbb{R}^{N \times F}$ denote a sequence of N feature vectors of dimensions F. A transformer is a function $T : \mathbb{R}^{N \times F} \rightarrow \mathbb{R}^{N \times F}$ defined by the composition of L transformer layers $T_1(\cdot), \ldots, T_L(\cdot)$, where

$$T_i(x) = f_i(A_i(x) + x)$$

Here $f_i(\cdot)$ is usually implemented using a feed-forward network, A_i is a self-attention function.

Formal Definition

Formally, the input sequence x is projected by three matrices $W_Q \in \mathbb{R}^{F \times D}$, $W_K \in \mathbb{R}^{F \times D}$ and $W_V \in \mathbb{R}^{F \times F}$ to corresponding representations Q, K and V. The output for all positions, $A_i(x) = V'$, is computed as follows,

$$Q = xW_Q,$$
$$K = xW_K,$$
$$V = xW_V,$$

The output $A_i(x)$ is computed as follows:

$$A_i(x) = V' = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V.$$

Linear Attention

Similarity Function

Attention on a closer look is nothing but a weighted average of the values, where the weight assigned to each value is computed by similarity function of the corresponding key.

$$A_i(x) = V' = \left[\frac{\sum_{j=1}^{N} sim(Q_k, K_j) V_j}{\sum_{j=1}^{N} sim(Q_k, K_j)} \right]_{k=1}^{N}$$

Linear Attention

Similarity Function

What are the properties we wish to have on similarity function?

1. sim:$\mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}_+$

Similarity Function

What are the properties we wish to have on similarity function?

1. sim:$\mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}_+$

Note that this includes all kernels $K(x, y) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}_+$

Linear Attention

Introduce Kernels

Thus, attention can be written as,

$$V' = \left[\frac{\sum_{j=1}^{N} K(Q_k, K_j) V_j}{\sum_{j=1}^{N} K(Q_k, K_j)} \right] \tag{1}$$

Thus, attention can be written as,

$$V' = \left[\frac{\sum_{j=1}^{N} K(Q_k, K_j) V_j}{\sum_{j=1}^{N} K(Q_k, K_j)} \right]$$

Let's look at each row of this matrix.

$$V'_k = \frac{\sum_{j=1}^{N} K(Q_k, K_j) V_j}{\sum_{j=1}^{N} K(Q_k, K_j)}$$

Linear Attention

Given a kernel with a feature representation $\phi(x)$, we can rewrite this as

$$V'_k = \frac{\sum_{j=1}^{N} \phi(Q_k)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_k)^T \phi(K_j)}$$

Linear Attention

Given a kernel with a feature representation $\phi(x)$, we can rewrite $eq^n - 3$ as follows,

$$V'_k = \frac{\sum_{j=1}^{N} \phi(Q_k)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_k)^T \phi(K_j)}$$

$$V'_k = \frac{\phi(Q_k)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_k)^T \sum_{j=1}^{N} \phi(K_j)}$$

Linear Attention

Given a kernel with a feature representation $\phi(x)$, we can rewrite $eq^n - 3$ as follows,

$$V_k' = \frac{\sum_{j=1}^{N} \phi(Q_k)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_k)^T \phi(K_j)}$$

$$V_k' = \frac{\phi(Q_k)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_k)^T \sum_{j=1}^{N} \phi(K_j)}$$

Since $\sum_{j=1}^{N} \phi(K_j) V_j^T$ and $\sum_{j=1}^{N} \phi(K_j)$ can be computed once and reused for every query, this formulation has $\mathcal{O}(N)$ time and memory complexity

Causal Masking

The transformer architecture can be used to efficiently train autoregressive models by masking the attention computation. i.e.

$$V'_k = \frac{\sum_{j=1}^{k} sim(Q_k, K_j)V_j}{\sum_{j=1}^{k} sim(Q_k, K_j)}$$

Causal Masking

The transformer architecture can be used to efficiently train autoregressive models by masking the attention computation. i.e.

$$V'_k = \frac{\sum_{j=1}^{k} \phi(Q_k)^T \phi(K_j) V_j^T}{\sum_{j=1}^{k} \phi(Q_k)^T \phi(K_j)}$$

$$V'_k = \frac{\phi(Q_k)^T \sum_{j=1}^{k} \phi(K_j) V_j^T}{\phi(Q_k)^T \sum_{j=1}^{k} \phi(K_j)}$$

Causal Masking

Let $S_k = \sum_{j=1}^{k} \phi(K_j) V_j^T$ and $Z_k = \sum_{j=1}^{k} \phi(K_j)$.

$$V_k' = \frac{\phi(Q_k) S_k}{\phi(Q_k) Z_k}$$

Note that, S_i and Z_i can be computed from S_{i-1} and Z_{i-1} in constant time.

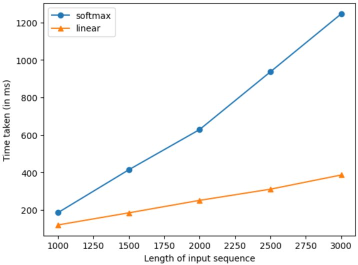
Causal Masking

This enables the transformer to be formulated as an RNN using the following recurrence relations. We set, and for all $i \geq 1$,

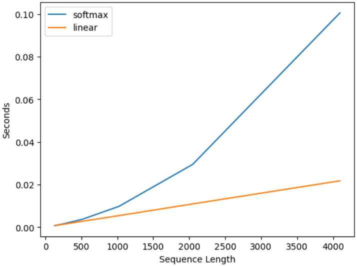
$$S_0 = 0$$
$$Z_0 = 0$$
$$S_i = S_{i-1} + \phi(K_i)V_i^T = S_{i-1} + \phi(x_i W_K)(x_i W_V)^T$$
$$Z_i = Z_{i-1} + \phi(K_i) = z_{i-1} + \phi(x_i W_K)$$

Experiments

Performance on random sequences



Time taken by attention layer



Time taken for one pass

Image Completion

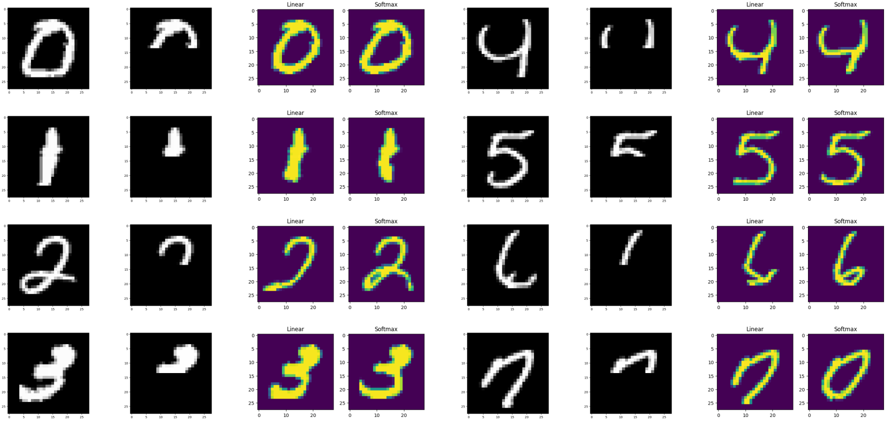


Image Completion

The linear transformer achieved faster image completion on occluded inputs in comparison to the softmax transformer.
The accuracies were 87% and 86.3% on images generated by the linear and the softmax transformers respectively.

Image Generation

Number of images	Linear	Softmax
100	33.45s	367.91s
200	67.76s	811.99s
300	76.46s	1248.12

Table: Time taken for image completion

1. Quadratic nature of self-attention is clearly visible in the plots shown above.
2. As expected, linear self-attention is way faster than softmax self-attention.

LoRA

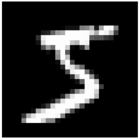
LoRA uses a simple property that, a matrix $M \in \mathbb{R}^{n \times m}$ can be rank decomposed as $M = UV$ where $U \in \mathbb{R}^{n \times r}$, $V \in \mathbb{R}^{r \times m}$ and r is the rank of M. This reduces the number of trainable parameters from mn to $r(n + m)$. Note that $r(n + m) \ll nm$ for small r.

Application of LoRA on Neural Networks

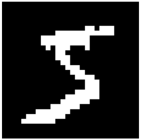
We have trained a feed forward neural network for an image classification task using the MNIST dataset. Our base model was composed of three linear layers which together had 55.1K trainable parameters.

Image Variations

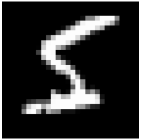
We create variations of the MNIST dataset to finetune and test the robustness of the model.



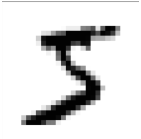
MNIST



Quantized MNIST



Rotated MNIST



Inverted MNIST

LoRA on Neural Networks

Base Model Performance on Various Datasets

After training, our base model achieved a test accuracy of approximately 93.2% on the original MNIST dataset. The performance of the model on modified datasets is presented in the table below.

Dataset	Accuracy
Original MNIST	93.2%
Quantized MNIST	85.58%
Rotated MNIST	12.38%
Inverted MNIST	5.52%

LoRA on Neural Networks

Fine-tuning the model on the variants of the MNIST dataset

We fine-tuned our base model on three variants.

Dataset Accuracy	Accuracy	Trainable Parameters
Quantized MNIST	93.57%	55.1K
Rotated MNIST	91.97%	55.1K
Inverted MNIST	76.41%	55.1K

LoRA on Neural Networks

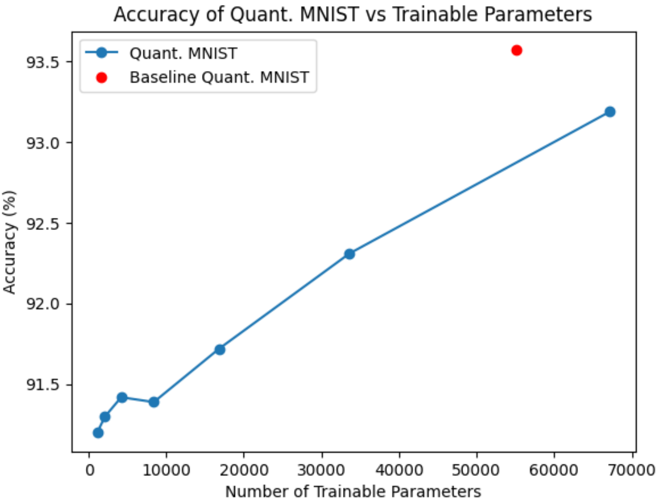
Applying LoRA during fine-tuning

We found that our fine-tuned models achieved accuracies comparable to their full fine-tuned counterparts with fewer trainable parameters.

r	Trainable Params	Quant. MNIST	Rot. MNIST	Inv. MNIST
1	1.1K	91.20%	37.53%	16.23%
2	2.1K	91.30%	49.01%	17.92%
4	4.2K	91.42%	69.10%	16.32%
8	8.4K	91.39%	77.49%	32.19%
16	16.8K	91.72%	86.95%	62.26%
32	33.6K	92.31%	89.50%	68.06%
64	67.2K	93.19%	90.41%	71.88%

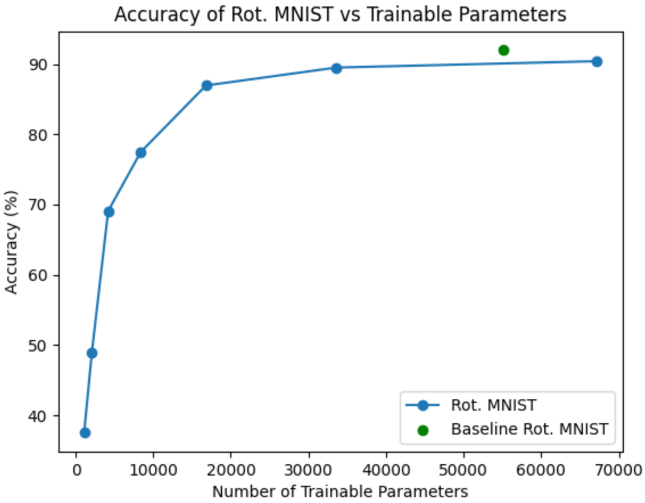
LoRA on Neural Networks

Applying LoRA during fine-tuning



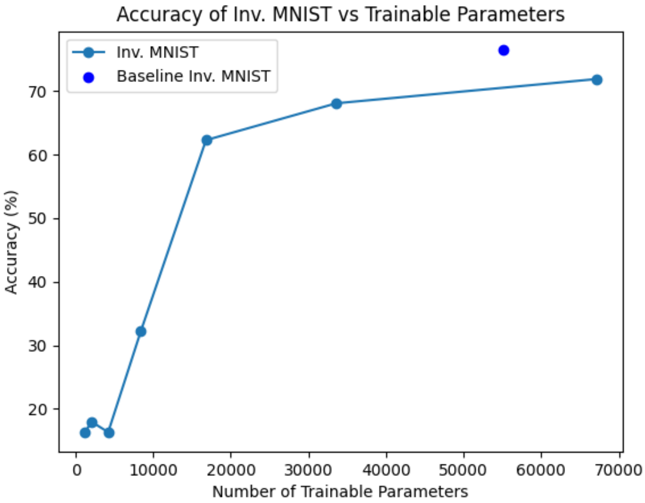
LoRA on Neural Networks

Applying LoRA during fine-tuning



LoRA on Neural Networks

Applying LoRA during fine-tuning



LoRA on Neural Networks

Training with LoRA from scratch

We trained the model from scratch with LoRA applied to the 3 layers in our neural network. The accuracy of the models are presented in the table below.

r	Trainable Params	MNIST	Quant. MNIST	Rot. MNIST	Inv. MNIST
1	1.1K	56.79%	23.50%	25.91%	22.21%
2	2.1K	71.90%	37.48%	43.96%	45.81%
4	4.2K	84.44%	64.87%	62.60%	69.67%
8	8.4K	89.12%	77.96%	82.39%	83.11%
16	16.8K	92.64%	88.2%	86.76%	87.38%
32	33.6K	93.98%	90.13%	90.62%	90.25%
64	67.2K	94.85%	91.66%	91.85%	86.01%

Thank You For Your Attention!

References