Linearizing Transformers

Ejaaz Merali

PIQuIL

August 7th, 2020

The Trouble with Transformers

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

Computation of the attention function has horrible time and memory complexity.
In terms of the sequence length n it is $O(n^2)$. This gets costly as n gets very large, as it typically does in natural language processing (and sometimes in physics).

In this talk, we'll discuss modifications to the attention function that seek to reduce its time and memory complexity.

Outline

- Linformer
  - Self-Attention is Low Rank
  - Projecting Keys and Values into a lower-dimensional space
  - Additional Efficiency Improvements
  - Results
- Transformers are RNNs
  - Generalizing Attention
  - Linearizing Attention
  - How to implement Causal Masking
  - Expressing Transformers as RNNs
  - Results
- Can we combine the two methods?

Linformer

Recent paper by Wang et al. (2020) that reduces the time complexity of attention by making use of the fact that the *context mapping matrix*, P, is low rank.

$$P = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

On the next slide, we show plots of the cumulative distribution of the singular values of P, averaged over 10 thousand sentences.

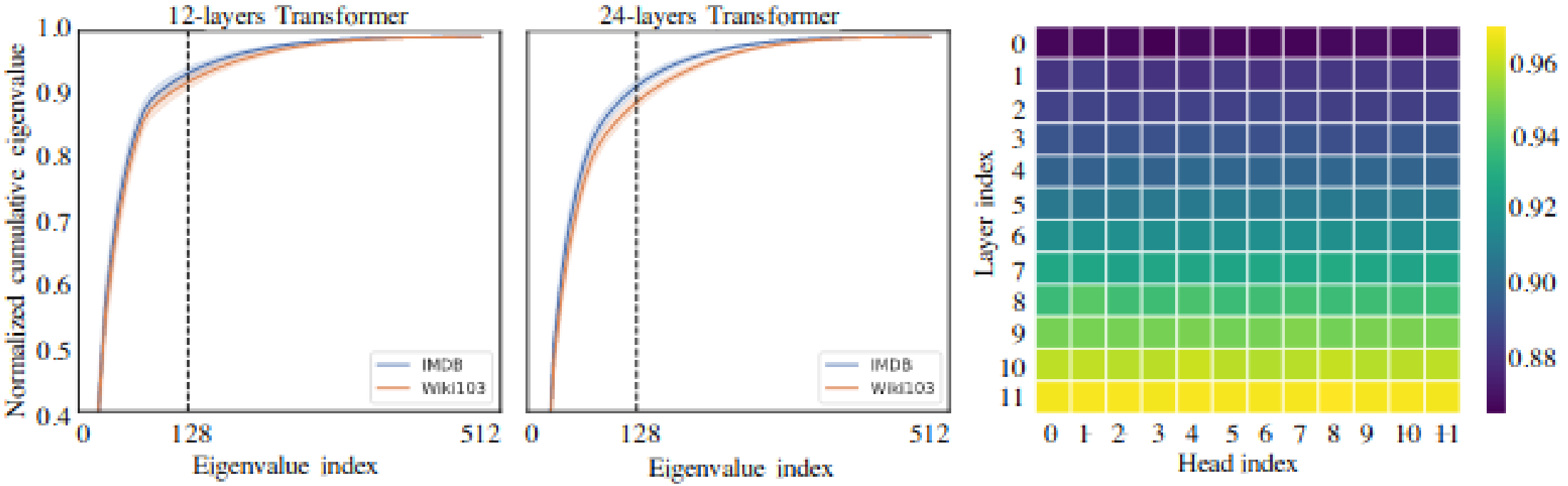


Figure 1: Left two figures are spectrum analysis of the self-attention matrix in pretrained transformer model (Liu et al., 2019) with $n = 512$. The Y-axis is the normalized cumulative singular value of context mapping matrix P, and the X-axis the index of largest eigenvalue. The results are based on both RoBERTa-base and large model in two public datasets: Wiki103 and IMDB. The right figure plots the heatmap of normalized cumulative eigenvalue at the 128-th largest eigenvalue across different layers and heads in Wiki103 data.

(Wang et al. 2020)

Self-Attention is Low Rank

Theorem 1: For any $Q, K \in \mathbb{R}^{n \times d_k}$, and $V \in \mathbb{R}^{n \times d_v}$, for any column vector v of V, there exists a low-rank matrix $\tilde{P} \in \mathbb{R}^{n \times n}$ such that:

$$\Pr(\|\tilde{P}v - Pv\| < \epsilon\|Pv\|) > 1 - o(1) \quad \text{and} \quad \text{rank}(\tilde{P}) = \Theta(\log(n))$$

which means that, for any ϵ we can construct a matrix $\tilde{P}$ (which has rank on the order of $\log(n)$) that has a high probability of approximating the action of P on a vector v.

This low-rank approximation to P constructed in the proof takes the form:

$$\tilde{P} = PR^T R$$

where $R \in \mathbb{R}^{k \times n}$ ($k < n$), with entries drawn from the normal distribution $N(0, 1/k)$.

Projecting Keys and Values to a Lower Dimensional space

Let $E, F \in \mathbb{R}^{k \times n}$. We'll use these matrices to project our keys and values into a lower dimensional space. Thus, our single-head attention becomes:

$$\text{Attention}(Q, EK, FV) = \text{softmax}\left(\frac{Q(EK)^T}{\sqrt{d_k}}\right)(FV)$$

Now our context mapping matrix P has shape $n \times k$ (with $k \ll n$), and the values matrix is effectively of size $k \times d_v$. We see that the operations above take time/space $O(nk) < O(n^2)$.

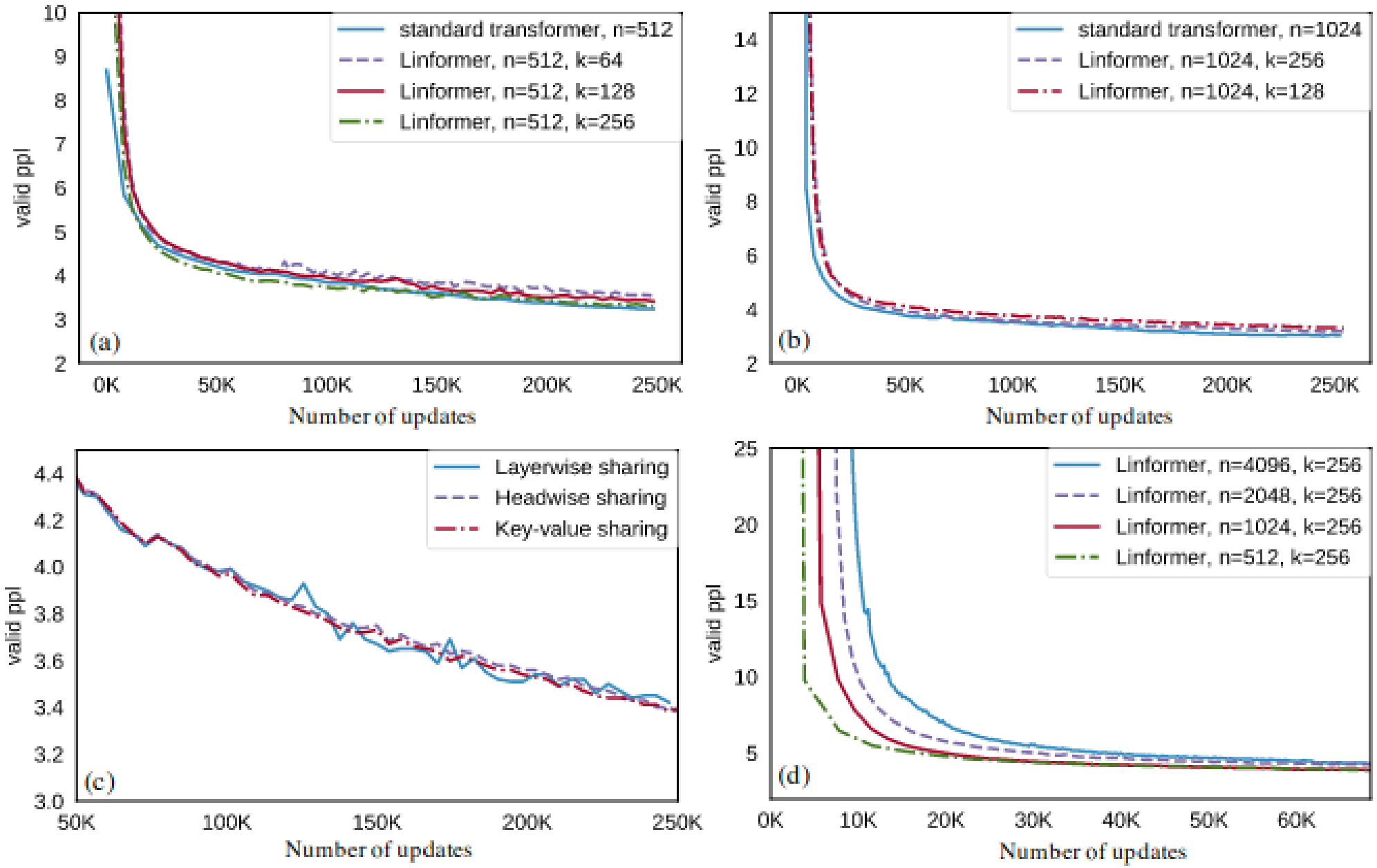
In a Multi-head Attention setting, we'd typically have separately trained E, F matrices for each attention head.

Additional Efficiency Improvements

The authors propose a few more tweaks to further optimize parameter count, performance, and efficiency.

1. **Headwise sharing**: share E, F across all heads in a MHA layer
2. **Key-value sharing**: Headwise sharing + using a single projection matrix instead of two $E = F$
3. **Layerwise sharing**: Key-value sharing + use the same projection matrix in *every* MHA layer

Results



(Wang et al. 2020)

Figure 3: Pretraining validation perplexity versus number of updates.

n	Model	SST-2	IMDB	QNLI	QQP	Average
512	Liu et al. (2019), RoBERTa-base	93.1	94.1	90.9	**90.9**	92.25
	Linformer, 128	92.4	94.0	90.4	90.2	91.75
	Linformer, 128, shared kv	**93.4**	93.4	90.3	90.3	91.85
	Linformer, 128, shared kv, layer	93.2	93.8	90.1	90.2	91.83
	Linformer, 256	93.2	94.0	90.6	90.5	92.08
	Linformer, 256, shared kv	93.3	93.6	90.6	90.6	92.03
	Linformer, 256, shared kv, layer	93.1	94.1	**91.2**	90.8	**92.30**
512	Devlin et al. (2019), BERT-base	92.7	93.5	91.8	89.6	91.90
	Sanh et al. (2019), Distilled BERT	91.3	92.8	89.2	88.5	90.45
1024	Linformer, 256	93.0	93.8	90.4	90.4	91.90
	Linformer, 256, shared kv	93.0	93.6	90.3	90.4	91.83
	Linformer, 256, shared kv, layer	93.2	**94.2**	90.8	90.5	92.18

Table 2: Dev set results on benchmark natural language understanding tasks. The RoBERTa-base model here is pretrained with same corpus as BERT.

(Wang et al. 2020)

length n	projected dimensions k				
	128	256	512	1024	2048
512	1.5x	1.3x	-	-	-
1024	1.7x	1.6x	1.3x	-	-
2048	2.6x	2.4x	2.1x	1.3x	-
4096	3.4x	3.2x	2.8x	2.2x	1.3x
8192	5.5x	5.0x	4.4x	3.5x	2.1x
16384	8.6x	7.8x	7.0x	5.6x	3.3x
32768	13x	12x	11x	8.8x	5.0x
65536	20x	18x	16x	14x	7.9x

length n	projected dimensions k				
	128	256	512	1024	2048
512	1.7x	1.5x	-	-	-
1024	3.0x	2.9x	1.8x	-	-
2048	6.1x	5.6x	3.6x	2.0x	-
4096	14x	13x	8.3x	4.3x	2.3x
8192	28x	26x	17x	8.5x	4.5x
16384	56x	48x	32x	16x	8x
32768	56x	48x	36x	18x	16x
65536	60x	52x	40x	20x	18x

Table 3: Inference-time efficiency improvements of the Linformer over the Transformer, across various projected dimensions k and sequence lengths n. Left table shows time saved. Right table shows memory saved.

(Wang et al. 2020)

Transformers are RNNs

By using a generalization of the attention function, Katharopoulos et al. (2020) were able to construct an alternative to softmax-attention which allows the computation to be done in linear time and space.

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

Generalizing Attention

The main idea is to view the softmax function as taking a "similarity score" between the queries and keys.

$$A(Q, K, V) = \frac{\sum_j \mathrm{sim}(Q_i, K_j) V_j}{\sum_j \mathrm{sim}(Q_i, K_j)}$$

where Q_i is the ith row of the matrix Q. We recover the softmax-attention by setting $\mathrm{sim}(q, k) = \exp(q^T k / \sqrt{(}d_k))$

Linearizing Attention

The only necessary constraint on the similarity function is that the output be non-negative. This includes all kernel functions:

$$k(x, y) : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^+$$

So, we take a kernel with feature representation $\phi(x)$:

$$[A(Q, K, V)]_i = \frac{\sum_j^n \phi(Q_i)^T \phi(K_j) V_j^T}{\sum_j^n \phi(Q_i)^T \phi(K_j)} \overset{(*)}{=} \frac{\phi(Q_i)^T \sum_j^n \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_j^n \phi(K_j)}$$

Due to the factorization in step $(*)$, we can cache the values of the sums over j and use them with multiple queries. As a result, we have linear time and space complexity in the sequence length n.

Short discussion on Complexity

The number of FLOPs for softmax attention scales as $O(n^2 \max(d_k, d_v))$, while the linearized attention we just derived scales as $O(ncd_v)$ where c is the output dimension of the feature map ϕ.

In the paper, they chose to use an elementwise feature map (thus giving $O(nd_kd_v)$ scaling):

$$\phi : \mathbb{R} \to \mathbb{R}^+$$

$$\phi(x) = \mathrm{elu}(x) + 1$$

where elu is the exponential linear unit (sort-of a smoothed ReLU)

$$\mathrm{elu}(x) = \begin{cases} x & \text{if } x > 0 \\ \exp(x) - 1 & \text{if } x \leq 0 \end{cases}$$

How to Implement Causal Masking (1/2)

Recall that, in order to preserve the autoregressive sampling of a generative Transformer or a decoder layer, we needed to apply masking to the first attention layer.

Instead of setting parts of the input of the similarity function to a masking value as before, we simply truncate the summation in the matrix multiplication between the context mapping matrix and the values matrix.

$$[A(Q, K, V)]_i = \frac{\phi(Q_i)^T \sum_j^i \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_j^i \phi(K_j)}$$

Define: $S_i = \sum_j^i \phi(K_j) V_j^T \in \mathbb{R}^{d_k \times d_v}$ and $Z_i = \sum_j^i \phi(K_j) \in \mathbb{R}^{d_k}$

How to Implement Causal Masking (2/2)

Then we have:

$$[A(Q, K, V)]_i = \frac{\phi(Q_i)^T S_i}{\phi(Q_i)^T Z_i}$$

We note that we can compute S_i and Z_i from S_{i-1} and Z_{i-1} in constant time (wrt the sequence length n). Causal masking didn't break the linear scaling.

Problem: a naive use of automatic differentiation for this masked attention function will require storing all intermediate values of S_i to compute the gradients. This would break the linear space scaling.

The authors derived the relevant gradients which allow performing the backward pass of causal linear attention in linear time and constant memory.

Expressing Transformers as RNNs

Notice that at each timestep, we are updating S and Z in order to produce our output.

This looks like an RNN with two hidden states.

$$\begin{cases} s_0, z_0 = 0, 0 \\ s_t = s_{t-1} + \phi(x_t W^{(K)})(x_t W^{(V)})^T \\ z_t = z_{t-1} + \phi(x_t W^{(K)}) \\ a_t = \frac{\phi(x_t W^{(Q)})^T s_t}{\phi(x_t W^{(Q)})^T z_t} \\ y_t = f_l\left(a_t + x_t\right) \end{cases}$$

where f_l is the Feed-Forward Net at the end of the Transformer layer l, and x_t is the input to the Transformer layer.

Multi-head Linear Attention as an RNN (not in the paper)

We easily see that a Multi-head Attention analogue of the linear Transformer would function as an RNN with $2h$ hidden states:

$$
\begin{cases}
s_0^i, z_0^i = 0, 0 \\
s_t^i = s_{t-1}^i + \phi(x_t W^{(K,i)})(x_t W^{(V,i)})^T \\
z_t^i = z_{t-1}^i + \phi(x_t W^{(K,i)}) \\
a_t^i = \frac{\phi(x_t W^{(Q,i)})^T s_t^i}{\phi(x_t W^{(Q,i)})^T z_t^i} \\
a_t = \mathrm{Concat}(a_t^1, a_t^2, \ldots, a_t^h) W^{(0)} \\
y_t = f_l\left(a_t + x_t\right)
\end{cases}
$$

Transformers are RNNs: Results

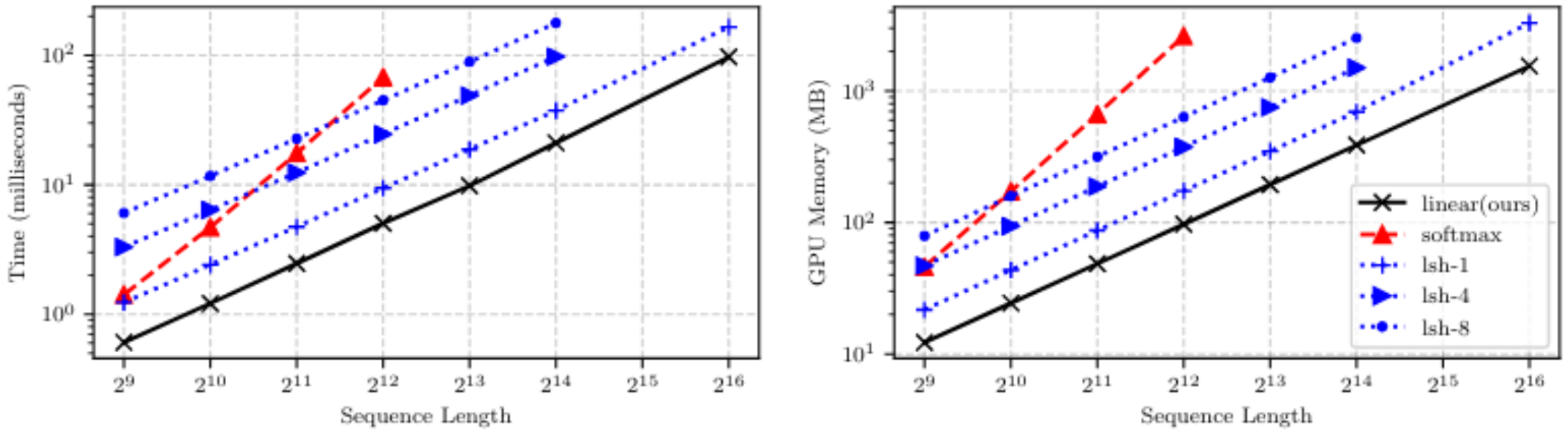


Figure 1: Comparison of the computational requirements for a forward/backward pass for Reformer (lsh-X), softmax attention and linear attention. Linear and Reformer models scale linearly with the sequence length unlike softmax which scales with the square of the sequence length both in memory and time. Full details of the experiment can be found in § 4.1.

(Katharopoulos et al. 2020)

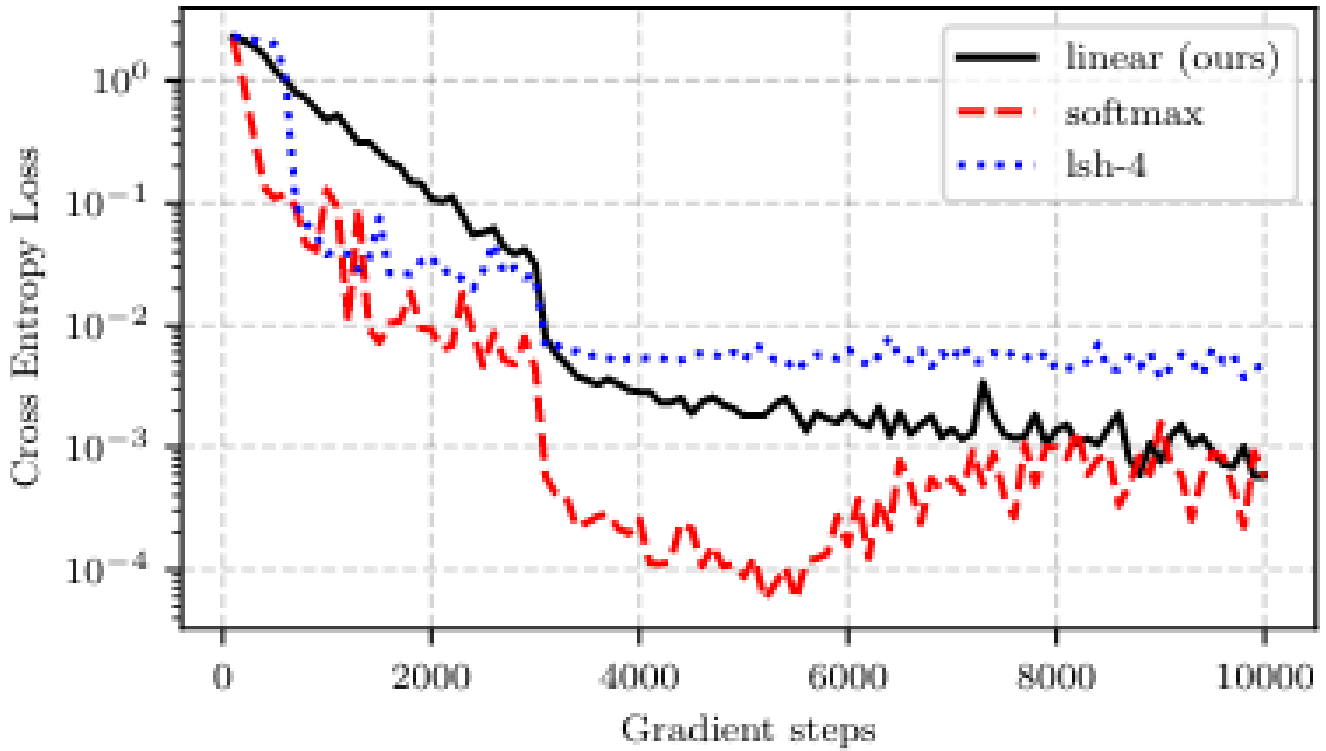


Figure 2: Convergence comparison of *softmax*, *linear* and *reformer* attention on a sequence duplication task. *linear* converges stably and reaches the same final performance as softmax. The details of the experiment are in § 4.1.

(Katharopoulos et al. 2020)

Method	Bits/dim	Images/sec	
Softmax	0.621	0.45	(1×)
LSH-1	0.745	0.68	(1.5×)
LSH-4	0.676	0.27	(0.6×)
Linear (ours)	0.644	**142.8**	**(317×)**

Table 1: Comparison of autoregressive image generation of MNIST images. Our linear transformers achieve almost the same bits/dim as the full softmax attention but more than 300 times higher throughput in image generation. The full details of the experiment are in § 4.2.1.

Method	Bits/dim	Images/sec	
Softmax	3.47	0.004	(1×)
LSH-1	3.39	0.015	(3.75×)
LSH-4	3.51	0.005	(1.25×)
Linear (ours)	3.40	**17.85**	**(4,462×)**

Table 2: We train autoregressive transformers for 1 week on a single GPU to generate CIFAR-10 images. Our linear transformer completes 3 times more epochs than softmax, which results in better perplexity. Our model generates images 4,000× faster than the baselines. The full details of the experiment are in § 4.2.2.

(Katharopoulos et al. 2020)

Method	Validation PER	Time/epoch (s)
Bi-LSTM	10.94	1047
Softmax	5.12	2711
LSH-4	9.33	2250
Linear (ours)	8.08	**824**

Table 3: Performance comparison in automatic speech recognition on the WSJ dataset. The results are given in the form of phoneme error rate (PER) and training time per epoch. Our model outperforms the LSTM and Reformer while being faster to train and evaluate. Details of the experiment can be found in § 4.3.

(Katharopoulos et al. 2020)

Can we combine the two methods?

Notice that in general, the feature map $\phi(x)$ can output a vector of a different size than its input.

Applying E to the output of applying ϕ on our keys, and F to our values as before. Our (unmasked) attention function then becomes:

$$[A(Q,K,V)]_i = \frac{\phi(Q_i)^T \sum_j^n (E\phi(K_j))(FV_j)^T}{\phi(Q_i)^T \sum_j^n \phi(K_j)}$$

Or, looking at just the numerator in matrix notation:

$$A(Q, K, V) \propto (\phi(Q))[(E\phi(K))^T (FV)]$$

where $E\phi(K)$ is a matrix of size $k \times d_k$, FV is of size $k \times d_v$, $\phi(Q)$ is $n \times d_k$.

The problem with this is that we can no longer guarantee that our similarity function is positive definite: $\text{sim}(q, k) = \phi(q)^T E\phi(k)$.

Again, once we've cached the value in the square brackets, the time and space complexity of computing attention, given a query matrix Q is $O(nd_k d_v)$.

So combining the two methods doesn't give any actual benefits in time or space complexity. In fact, it introduces new problems.

References

Katharopoulos, Angelos, et al. "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention." arXiv preprint arXiv:2006.16236 (2020).

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.

Wang, Sinong, et al. "Linformer: Self-Attention with Linear Complexity." arXiv preprint arXiv:2006.04768 (2020).