Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention

Angelos Katharopoulos [1 2] **Apoorv Vyas** [1 2] **Nikolaos Pappas** [3] **François Fleuret** [2 4 *]

Abstract

Transformers achieve remarkable performance in several tasks but due to their quadratic complexity, with respect to the input's length, they are prohibitively slow for very long sequences. To address this limitation, we express the self-attention as a linear dot-product of kernel feature maps and make use of the associativity property of matrix products to reduce the complexity from $\mathcal{O}\left(N^2\right)$ to $\mathcal{O}\left(N\right)$, where N is the sequence length. We show that this formulation permits an iterative implementation that dramatically accelerates autoregressive transformers and reveals their relationship to recurrent neural networks. Our *linear transformers* achieve similar performance to vanilla transformers and they are up to 4000x faster on autoregressive prediction of very long sequences.

1. Introduction

Transformer models were originally introduced by Vaswani et al. (2017) in the context of neural machine translation (Sutskever et al., 2014; Bahdanau et al., 2015) and have demonstrated impressive results on a variety of tasks dealing with natural language (Devlin et al., 2019), audio (Sperber et al., 2018), and images (Parmar et al., 2019). Apart from tasks with ample supervision, transformers are also effective in transferring knowledge to tasks with limited or no supervision when they are pretrained with autoregressive (Radford et al., 2018; 2019) or masked language modeling objectives (Devlin et al., 2019; Yang et al., 2019; Song et al., 2019; Liu et al., 2020).

However, these benefits often come with a very high computational and memory cost. The bottleneck is mainly caused by the global receptive field of self-attention, which processes contexts of N inputs with a quadratic memory and time complexity $\mathcal{O}\left(N^2\right)$. As a result, in practice transformers are slow to train and their context is *limited*. This disrupts temporal coherence and hinders the capturing of long-term dependencies. Dai et al. (2019) addressed the latter by attending to memories from previous contexts albeit at the expense of computational efficiency.

Lately, researchers shifted their attention to approaches that increase the context length without sacrificing efficiency. Towards this end, Child et al. (2019) introduced sparse factorizations of the attention matrix to reduce the self-attention complexity to $\mathcal{O}\left(N\sqrt{N}\right)$. Kitaev et al. (2020) further reduced the complexity to $\mathcal{O}\left(N \log N\right)$ using locality-sensitive hashing. This made scaling to long sequences possible. Even though the aforementioned models can be efficiently trained on large sequences, they do not speed-up autoregressive inference.

In this paper, we introduce the *linear transformer* model that significantly reduces the memory footprint and scales linearly with respect to the context length. We achieve this by using a kernel-based formulation of self-attention and the associative property of matrix products to calculate the self-attention weights (§ 3.2). Using our linear formulation, we also express causal masking with linear complexity and constant memory (§ 3.3). This reveals the relation between transformers and RNNs, which enables us to perform autoregressive inference orders of magnitude faster (§ 3.4). Our evaluation on image generation and automatic speech recognition demonstrates that *linear transformer* can reach the performance levels of transformer, while being up to three orders of magnitude faster during inference.

2. Related Work

In this section, we provide an overview of the most relevant works that seek to address the large memory and computational requirements of transformers. Furthermore, we discuss methods that theoretically analyze the core component of the transformer model, namely self-attention. Finally, we present another line of work that seeks to alleviate the softmax bottleneck in the attention computation.

[1]Idiap Research Institute, Switzerland [2]EPFL, Switzerland [3]University of Washington, Seattle, USA [4]University of Geneva, Switzerland. [*]Work done at Idiap. Correspondence to: Angelos Katharopoulos <firstname.lastname@idiap.ch>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

2.1. Efficient Transformers

Existing works seek to improve memory efficiency in transformers through weight pruning (Michel et al., 2019), weight factorization (Lan et al., 2020), weight quantization (Zafrir et al., 2019) or knowledge distillation. Clark et al. (2020) proposed a new pretraining objective called replaced token detection that is more sample efficient and reduces the overall computation. Lample et al. (2019) used product-key attention to increase the capacity of any layer with negligible computational overhead.

Reducing the memory or computational requirements with these methods leads to training or inference time speedups, but, fundamentally, the time complexity is still quadratic with respect to the sequence length which hinders scaling to long sequences. In contrast, we show that our method reduces both memory and time complexity of transformers both theoretically (§ 3.2) and empirically (§ 4.1).

Another line of research aims at increasing the "context" of self-attention in transformers. Context refers to the maximum part of the sequence that is used for computing self-attention. Dai et al. (2019) introduced Transformer-XL which achieves state-of-the-art in language modeling by learning dependencies beyond a fixed length context without disrupting the temporal coherence. However, maintaining previous contexts in memory introduces significant additional computational cost. In contrast, Sukhbaatar et al. (2019) extended the context length significantly by learning the optimal attention span per attention head, while maintaining control over the memory footprint and computation time. Note that both approaches have the same asymptotic complexity as the vanilla model. In contrast, we improve the asymptotic complexity of the self-attention, which allows us to use significantly larger context.

More related to our model are the works of Child et al. (2019) and Kitaev et al. (2020). The former (Child et al., 2019) introduced sparse factorizations of the attention matrix reducing the overall complexity from quadratic to $\mathcal{O}\left(N\sqrt{N}\right)$ for generative modeling of long sequences. More recently, Kitaev et al. (2020) proposed Reformer. This method further reduces complexity to $\mathcal{O}\left(N \log N\right)$ by using locality-sensitive hashing (LSH) to perform fewer dot products. Note that in order to be able to use LSH, Reformer constrains the keys, for the attention, to be identical to the queries. As a result this method cannot be used for decoding tasks where the keys need to be different from the queries. In comparison, *linear transformers* impose no constraints on the queries and keys and scale linearly with respect to the sequence length. Furthermore, they can be used to perform inference in autoregressive tasks three orders of magnitude faster, achieving comparable performance in terms of validation perplexity.

2.2. Understanding Self-Attention

There have been few efforts to better understand self-attention from a theoretical perspective. Tsai et al. (2019) proposed a kernel-based formulation of attention in transformers which considers attention as applying a kernel smoother over the inputs with the kernel scores being the similarity between inputs. This formulation provides a better way to understand attention components and integrate the positional embedding. In contrast, we use the kernel formulation to speed up the calculation of self-attention and lower its computational complexity. Also, we observe that if a kernel with positive similarity scores is applied on the queries and keys, linear attention converges normally.

More recently, Cordonnier et al. (2020) provided theoretical proofs and empirical evidence that a multi-head self-attention with sufficient number of heads can express any convolutional layer. Here, we instead show that a self-attention layer trained with an autoregressive objective can be seen as a recurrent neural network and this observation can be used to significantly speed up inference time of autoregressive transformer models.

2.3. Linearized softmax

For many years, softmax has been the bottleneck for training classification models with a large number of categories (Goodman, 2001; Morin & Bengio, 2005; Mnih & Hinton, 2009). Recent works (Blanc & Rendle, 2017; Rawat et al., 2019), have approximated softmax with a linear dot product of feature maps to speed up the training through sampling. Inspired from these works, we linearize the softmax attention in transformers. Concurrently with this work, Shen et al. (2020) explored the use of linearized attention for the task of object detection in images. In comparison, we do not only linearize the attention computation, but also develop an autoregressive transformer model with linear complexity and constant memory for both inference and training. Moreover, we show that through the lens of kernels, every transformer can be seen as a recurrent neural network.

3. Linear Transformers

In this section, we formalize our proposed *linear transformer*. We present that changing the attention from the traditional *softmax* attention to a feature map based dot product attention results in better time and memory complexity as well as a causal model that can perform sequence generation in linear time, similar to a recurrent neural network.

Initially, in § 3.1, we introduce a formulation for the transformer architecture introduced in (Vaswani et al., 2017). Subsequently, in § 3.2 and § 3.3 we present our proposed *linear transformer* and finally, in § 3.4 we rewrite the transformer as a recurrent neural network.

3.1. Transformers

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 $T_1(\cdot), \ldots, T_L(\cdot)$ as follows,

$$T_l(x) = f_l(A_l(x) + x). \tag{1}$$

The function $f_l(\cdot)$ transforms each feature independently of the others and is usually implemented with a small two-layer feedforward network. $A_l(\cdot)$ is the self attention function and is the only part of the transformer that acts across sequences.

The self attention function $A_l(\cdot)$ computes, for every position, a weighted average of the feature representations of all other positions with a weight proportional to a similarity score between the representations. 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 $W_V \in \mathbb{R}^{F \times M}$ to corresponding representations Q, K and V. The output for all positions, $A_l(x) = V'$, is computed as follows,

Original formula for Scaled dot-product softmax attention from "Attention is all you need" paper.

$$\begin{aligned} Q &= xW_Q, \\ K &= xW_K, \\ V &= xW_V, \\ A_l(x) = V' &= \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V. \end{aligned} \tag{2}$$

Note that in the previous equation, the softmax function is applied rowwise to QK^T. Following common terminology, the Q, K and V are referred to as the "queries", "keys" and "values" respectively.

Equation 2 implements a specific form of self-attention called softmax attention where the similarity score is the exponential of the dot product between a query and a key. Given that subscripting a matrix with i returns the i-th row as a vector, we can write a generalized attention equation for any similarity function as follows,

$$V'_i = \frac{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right)}. \tag{3}$$

Equation 3 is equivalent to equation 2 if we substitute the similarity function with $\text{sim}\left(q, k\right) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$.

3.2. Linearized Attention

The definition of attention in equation 2 is generic and can be used to define several other attention implementations such as polynomial attention or RBF kernel attention (Tsai et al., 2019). Note that the only constraint we need to impose to $\text{sim}\left(\cdot\right)$, in order for equation 3 to define an attention function, is to be non-negative. This includes all kernels $k(x, y) : \mathbb{R}^{2 \times F} \rightarrow \mathbb{R}_+$.

Given such a kernel with a feature representation $\phi\left(x\right)$ we can rewrite equation 2 as follows,

$$V'_i = \frac{\sum_{j=1}^{N} \phi\left(Q_i\right)^T \phi\left(K_j\right) V_j}{\sum_{j=1}^{N} \phi\left(Q_i\right)^T \phi\left(K_j\right)}, \tag{4}$$

and then further simplify it by making use of the associative property of matrix multiplication to

$$V'_i = \frac{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right)}. \tag{5}$$

The above equation is simpler to follow when the numerator is written in vectorized form as follows,

$$\left(\phi\left(Q\right)\phi\left(K\right)^T\right)V = \phi\left(Q\right)\left(\phi\left(K\right)^T V\right). \tag{6}$$

Note that the feature map $\phi\left(\cdot\right)$ is applied rowwise to the matrices Q and K.

From equation 2, it is evident that the computational cost of softmax attention scales with $\mathcal{O}\left(N^2\right)$, where N represents the sequence length. The same is true for the memory requirements because the full attention matrix must be stored to compute the gradients with respect to the queries, keys and values. In contrast, our proposed *linear transformer* from equation 5 has time and memory complexity $\mathcal{O}\left(N\right)$ because we can compute $\sum_{j=1}^{N} \phi\left(K_j\right) V_j^T$ and $\sum_{j=1}^{N} \phi\left(K_j\right)$ once and reuse them for every query.

3.2.1. Feature Maps and Computational Cost

For softmax attention, the total cost in terms of multiplications and additions scales as $\mathcal{O}\left(N^2 \max\left(D, M\right)\right)$, where D is the dimensionality of the queries and keys and M is the dimensionality of the values. On the contrary, for linear attention, we first compute the feature maps of dimensionality C. Subsequently, computing the new values requires $\mathcal{O}\left(NCM\right)$ additions and multiplications.

The previous analysis does not take into account the choice of kernel and feature function. Note that the feature function that corresponds to the exponential kernel is infinite dimensional, which makes the linearization of exact softmax attention infeasible. On the other hand, the polynomial kernel, for example, has an exact finite dimensional feature map and has been shown to work equally well with the exponential or RBF kernel (Tsai et al., 2019). The computational cost for a linearized polynomial transformer of degree 2 is $\mathcal{O}\left(ND^2M\right)$. This makes the computational complexity favorable when $N > D^2$. Note that this is true in practice since we want to be able to process sequences with tens of thousands of elements.

For our experiments, that deal with smaller sequences, we employ a feature map that results in a positive similarity

function as defined below,

$$\phi\left(x\right) = \text{elu}(x) + 1, \tag{7}$$

where $\text{elu}(\cdot)$ denotes the exponential linear unit (Clevert et al., 2015) activation function. We prefer $\text{elu}(\cdot)$ over $\text{relu}(\cdot)$ to avoid setting the gradients to 0 when x is negative. This feature map results in an attention function that requires $\mathcal{O}\left(NDM\right)$ multiplications and additions. In our experimental section, we show that the feature map of equation 7 performs on par to the full transformer, while significantly reducing the computational and memory requirements.

3.3. Causal Masking

The transformer architecture can be used to efficiently train autoregressive models by masking the attention computation such that the i-th position can only be influenced by a position j if and only if $j \leq i$, namely a position cannot be influenced by the subsequent positions. Formally, this causal masking changes equation 3 as follows,

$$V_i' = \frac{\sum_{j=1}^{i} \text{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{i} \text{sim}\left(Q_i, K_j\right)}. \tag{8}$$

Following the reasoning of § 3.2, we linearize the masked attention as described below,

$$V_i' = \frac{\phi\left(Q_i\right)^T \sum_{j=1}^{i} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{i} \phi\left(K_j\right)}. \tag{9}$$

By introducing S_i and Z_i as follows,

$$S_i = \sum_{j=1}^{i} \phi\left(K_j\right) V_j^T, \tag{10}$$

$$Z_i = \sum_{j=1}^{i} \phi\left(K_j\right), \tag{11}$$

we can simplify equation 9 to

$$V_i' = \frac{\phi\left(Q_i\right)^T S_i}{\phi\left(Q_i\right)^T Z_i}. \tag{12}$$

Note that, S_i and Z_i can be computed from S_{i-1} and Z_{i-1} in constant time hence making the computational complexity of linear transformers with causal masking linear with respect to the sequence length.

3.3.1. Gradient Computation

A naive implementation of equation 12, in any deep learning framework, requires storing all intermediate values S_i in order to compute the gradients. This increases the memory consumption by $\max\left(D, M\right)$ times; thus hindering the applicability of causal linear attention to longer sequences or deeper models. To address this, we derive the gradients of the numerator in equation 9 as cumulative sums. This allows us to compute both the forward and backward pass of causal linear attention in **linear time** and **constant memory**. A detailed derivation is provided in the supplementary material.

Given the numerator $\bar{V}_i$ and the gradient of a scalar loss function with respect to the numerator $\nabla_{\bar{V}_i}\mathcal{L}$, we derive $\nabla_{\phi(Q_i)}\mathcal{L}$, $\nabla_{\phi(K_i)}\mathcal{L}$ and $\nabla_{V_i}\mathcal{L}$ as follows,

$$\nabla_{\phi(Q_i)}\mathcal{L} = \nabla_{\bar{V}_i}\mathcal{L} \left(\sum_{j=1}^{i} \phi\left(K_j\right) V_j^T\right)^T, \tag{13}$$

$$\nabla_{\phi(K_i)}\mathcal{L} = \left(\sum_{j=i}^{N} \phi\left(Q_j\right) \left(\nabla_{\bar{V}_j}\mathcal{L}\right)^T\right) V_i\,, \tag{14}$$

$$\nabla_{V_i}\mathcal{L} = \left(\sum_{j=i}^{N} \phi\left(Q_j\right) \left(\nabla_{\bar{V}_j}\mathcal{L}\right)^T\right)^T \phi\left(K_i\right). \tag{15}$$

The cumulative sum terms in equations 9, 13-15 are computed in linear time and require constant memory with respect to the sequence length. This results in an algorithm with computational complexity $\mathcal{O}\left(NCM\right)$ and memory $\mathcal{O}\left(N \max\left(C, M\right)\right)$ for a given feature map of C dimensions. A pseudocode implementation of the forward and backward pass of the numerator is given in algorithm 1.

3.3.2. Training and Inference

When training an autoregressive transformer model the full ground truth sequence is available. This makes layerwise parallelism possible both for $f_l(\cdot)$ of equation 1 and the attention computation. As a result, transformers are more efficient to train than recurrent neural networks. On the other hand, during inference the output for timestep i is the input for timestep $i+1$. This makes autoregressive models impossible to parallelize. Moreover, the cost per timestep for transformers is not constant; instead, it scales with the square of the current sequence length because attention must be computed for all previous timesteps.

Our proposed *linear transformer* model *combines the best of both worlds*. When it comes to training, the computations can be parallelized and take full advantage of GPUs or other accelerators. When it comes to inference, the cost per time and memory for one prediction is constant for our model. This means we can simply store the $\phi\left(K_j\right) V_j^T$ matrix as an internal state and update it at every time step like a recurrent neural network. This results in inference **thousands of times faster** than other transformer models.

3.4. Transformers are RNNs

In literature, transformer models are considered to be a fundamentally different approach to recurrent neural networks. However, from the causal masking formulation in § 3.3 and the discussion in the previous section, it becomes evident that any transformer layer with causal masking can be written as a model that, given an input, modifies an internal state and then predicts an output, namely a Recurrent Neural Network (RNN). Note that, in contrast to Universal Transformers (Dehghani et al., 2018), we consider the recurrence with respect to time and not depth.

In the following equations, we formalize the transformer layer of equation 1 as a recurrent neural network. The resulting RNN has two hidden states, namely the attention memory s and the normalizer memory z. We use subscripts to denote the timestep in the recurrence.

Forming the linear attention in an RNN formula for equences

$$s_0 = 0, \tag{16}$$
$$z_0 = 0, \tag{17}$$
$$s_i = s_{i-1} + \phi\left(x_i W_K\right)\left(x_i W_V\right)^T, \tag{18}$$
$$z_i = z_{i-1} + \phi\left(x_i W_K\right), \tag{19}$$
$$y_i = f_l\left(\frac{\phi\left(x_i W_Q\right)^T s_i}{\phi\left(x_i W_Q\right)^T z_i} + x_i\right). \tag{20}$$

In the above equations, x_i denotes the i-th input and y_i the i-th output for a specific transformer layer. Note that our formulation does not impose any constraint on the feature function and it can be used for representing *any transformer* model, in theory even the ones using softmax attention. This formulation is a first step towards better understanding the relationship between transformers and popular recurrent networks (Hochreiter & Schmidhuber, 1997) and the processes used for storing and retrieving information.

Algorithm 1 Linear transformers with causal masking

function `forward` $(\phi(Q), \phi(K), V)$**:**
  $V' \leftarrow 0, S \leftarrow 0$
  for $i = 1, \ldots, N$ **do**
    $S \leftarrow S + \phi(K_i) V_i^T$ equation 10
    $\bar{V}_i \leftarrow \phi(Q_i) S$
  end
  return $\bar{V}$
end

function `backward` $(\phi(Q), \phi(K), V, G)$**:**
  `/* G is the gradient of the loss with respect to the output of forward */`
  $S \leftarrow 0, \nabla_{\phi(Q)}\mathcal{L} \leftarrow 0$
  for $i = 1, \ldots, N$ **do**
    $S \leftarrow S + \phi(K_i) V_i^T$
    $\nabla_{\phi(Q_i)}\mathcal{L} \leftarrow G_i S^T$ equation 13
  end
  $S \leftarrow 0, \nabla_{\phi(K)}\mathcal{L} \leftarrow 0, \nabla_V \mathcal{L} \leftarrow 0$
  for $i = N, \ldots, 1$ **do**
    $S \leftarrow S + \phi(Q_i) G_i^T$
    $\nabla_{V_i}\mathcal{L} \leftarrow S^T \phi(K_i)$ equation 15
    $\nabla_{\phi(K_i)}\mathcal{L} \leftarrow S V_i$ equation 14
  end
  return $\nabla_{\phi(Q)}\mathcal{L}, \nabla_{\phi(K)}\mathcal{L}, \nabla_V \mathcal{L}$
end

4. Experiments

In this section, we analyze experimentally the performance of the proposed *linear transformer*. Initially, in § 4.1, we evaluate the linearized attention in terms of computational cost, memory consumption and convergence on synthetic data. To further showcase the effectiveness of *linear transformers*, we evaluate our model on two real-world applications, image generation in § 4.2 and automatic speech recognition in § 4.3. We show that our model achieves competitive performance with respect to the state-of-the-art transformer architectures, while requiring significantly less GPU memory and computation.

Throughout our experiments, we compare our model with two baselines, the full transformer with softmax attention and the Reformer (Kitaev et al., 2020), the latter being a state-of-the-art accelerated transformer architecture. For the Reformer, we use a PyTorch reimplementation of the published code and for the full transformer we use the default PyTorch implementation. Note that for Reformer, we do not use the reversible layers, however, this does not affect the results as we only measure the memory consumption with respect to the self attention layer. In all experiments, we use **softmax** (Vaswani et al., 2017) to refer to the standard transformer architecture, **linear** for our proposed *linear transformers* and **lsh-X** for Reformer (Kitaev et al., 2020), where *X* denotes the hashing rounds.

For training the *linear transformers*, we use the feature map of equation 7. Our PyTorch (Paszke et al., 2019) code with documentation and examples can be found at `https://linear-transformers.com/`. The constant memory gradient computation of equations 13-15 is implemented in approximately 200 lines of CUDA code.

4.1. Synthetic Tasks

4.1.1. Convergence Analysis

To examine the convergence properties of *linear transformers* we train on an artifical copy task with causal masking. Namely, the transformers have to copy a series of symbols similar to the sequence duplication task of Kitaev et al. (2020). We use a sequence of maximum length 128 with 10

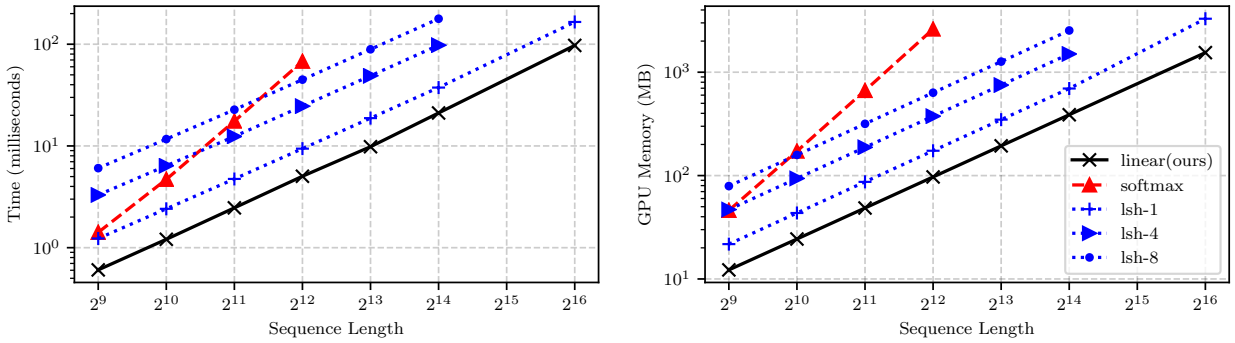

Figure 1: Comparison of the computational requirements for a forward/backward pass for Reformer (lsh-X), softmax attention and linear attention. Linear and Reformer models scale linearly with the sequence length unlike softmax which scales with the square of the sequence length both in memory and time.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

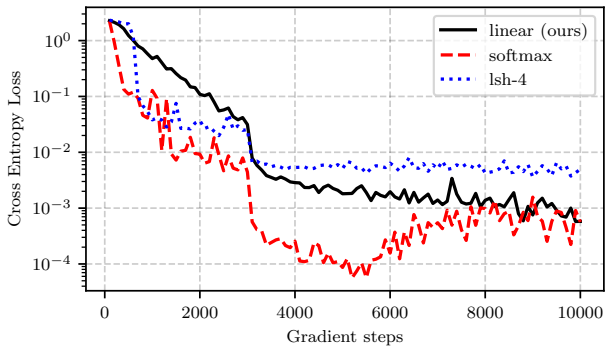

Figure 2: Convergence comparison of *softmax*, *linear* and *reformer* attention on a sequence duplication task. *linear* converges stably and reaches the same final performance as softmax. The details of the experiment are in § 4.1.

different symbols separated by a dedicated separator symbol. For all three methods, we train a 4 layer transformer with 8 attention heads using a batch size of 64 and the RAdam optimizer (Liu et al., 2019) with a learning rate of 10^{-3} which is reduced to 10^{-4} after 3000 updates. Figure 2 depicts the loss with respect to the number of gradient steps. We observe that linear converges smoothly and reaches a lower loss than lsh due to the lack of noise introduced by hashing. In particular, it reaches the same loss as softmax.

4.1.2. Memory and Computational Requirements

In this subsection, we compare transformers with respect to their computational and memory requirements. We compute the attention and the gradients for a synthetic input with varying sequence lengths $N \in \{2^9, 2^{10}, \ldots, 2^{16}\}$ and measure the peak allocated GPU memory and required time for each variation of transformer. We scale the batch size inversely with the sequence length and report the time and memory per sample in the batch.

Every method is evaluated up to the maximum sequence length that fits the GPU memory. For this benchmark we use an NVidia GTX 1080 Ti with 11GB of memory. This results in a maximum sequence length of 4,096 elements for softmax and 16,384 for lsh-4 and lsh-8. As expected, softmax scales quadratically with respect to the sequence length. Our method is faster and requires less memory than the baselines for every configuration, as seen in figure 1. We observe that both Reformer and linear attention scale linearly with the sequence length. Note that although the asymptotic complexity for Reformer is $\mathcal{O}\left(N \log N\right)$, $\log N$ is small enough and does not affect the computation time.

4.2. Image Generation

Transformers have shown great results on the task of conditional or unconditional autoregressive generation (Radford et al., 2019; Child et al., 2019), however, sampling from transformers is slow due to the task being inherently sequential and the memory scaling with the square of the sequence length. In this section, we train causally masked transformers to predict images pixel by pixel. Our achieved performance in terms of bits per dimension is on par with *softmax* attention while being able to generate images **more than 1,000 times faster** and with **constant memory per image** from the first to the last pixel. We refer the reader to our supplementary for comparisons in terms of training evolution, quality of generated images and time to generate a single image. In addition, we also compare with a faster softmax transformer that caches the keys and values during inference, in contrast to the PyTorch implementation.

4.2.1. MNIST

First, we evaluate our model on image generation with autoregressive transformers on the widely used MNIST dataset (LeCun et al., 2010). The architecture for this experiment comprises 8 attention layers with 8 attention heads each. We

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.

set the embedding size to 256 which is 32 dimensions per head. Our feed forward dimensions are 4 times larger than our embedding size. We model the output with a mixture of 10 logistics as introduced by Salimans et al. (2017). We use the RAdam optimizer with a learning rate of 10^{-4} and train all models for 250 epochs. For the reformer baseline, we use 1 and 4 hashing rounds. Furthermore, as suggested in Kitaev et al. (2020), we use 64 buckets and chunks with approximately 32 elements. In particular, we divide the 783 long input sequence to 27 chunks of 29 elements each. Since the sequence length is realtively small, namely only 784 pixels, to remove differences due to different batch sizes we use a batch size of 10 for all methods.

Table 1 summarizes the results. We observe that linear transformers achieve almost the same performance, in terms of final perplexity, as softmax transformers while being able to generate images more than 300 times faster. This is achieved due to the low memory requirements of our model, which is able to simultaneously generate 10,000 MNIST images with a single GPU. In particular, the memory is constant with respect to the sequence length because the only thing that needs to be stored between pixels are the s_i and z_i values as described in equations 18 and 19. On the other hand, both softmax and Reformer require memory that increases with the length of the sequence.

Image completions and unconditional samples from our MNIST model can be seen in figure 3. We observe that our linear transformer generates very convincing samples with sharp boundaries and no noise. In the case of image completion, we also observe that the transformer learns to use the same stroke style and width as the original image effectively attending over long temporal distances. Note that as the achieved perplexity is more or less the same for all models, we do not observe qualitative differences between the generated samples from different models.

4.2.2. CIFAR-10

The benefits of our linear formulation increase as the sequence length increases. To showcase that, we train 16 layer transformers to generate CIFAR-10 images (Krizhevsky et al., 2009). For each layer we use the same configuration as in the previous experiment. For Reformer, we use again 64 buckets and 83 chunks of 37 elements, which is approximately 32, as suggested in the paper. Since the sequence length is almost 4 times larger than for the previous experiment, the full transformer can only be used with a batch size of 1 in the largest GPU that is available to us, namely an NVidia P40 with 24GB of memory. For both the linear transformer and reformer, we use a batch size of 4. All models are trained for 7 days. We report results in terms of bits per dimension and image generation throughput in table 2. Note that although the main point of this experiment is not the final perplexity, it is evident that as the sequence length grows, the fast transformer models become increasingly more efficient per GPU hour, achieving better scores than their slower counterparts.

As the memory and time to generate a single pixel scales quadratically with the number of pixels for both Reformer and softmax attention, the increase in throughput for our linear transformer is even more pronounced. In particular, **for every image generated** by the softmax transformer, **our method can generate 4,460 images**. Image completions and unconditional samples from our model can be seen in figure 4. We observe that our model generates images with spatial consistency and can complete images convincigly without significantly hindering the recognition of the image category. For instance, in figure 4b, all images have successfully completed the dog's nose (first row) or the windshield of the truck (last row).

4.3. Automatic Speech Recognition

To show that our method can also be used for non-autoregressive tasks, we evaluate the performance of linear transformers in end-to-end automatic speech recognition using Connectionist Temporal Classification (CTC) loss (Graves et al., 2006). In this setup, we predict a distribution over phonemes for each input frame in a non autore-

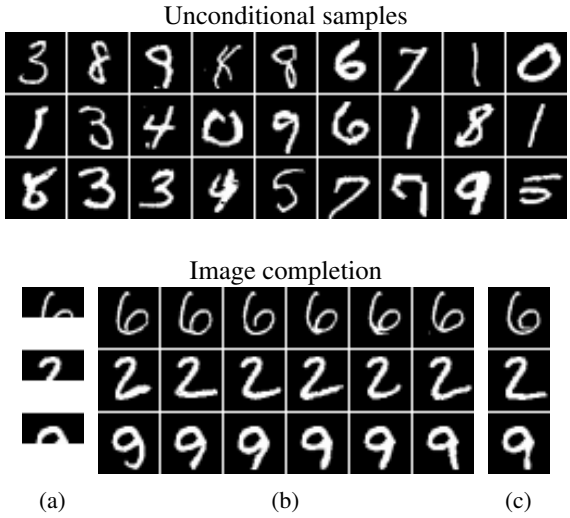

Figure 3: Unconditional samples and image completions generated by our method for MNIST. (a) depicts the occluded orignal images, (b) the completions and (c) the original. Our model achieves comparable bits/dimension to softmax, while having more than **300 times** higher throughput, generating **142 images/second**. For details see § 4.2.1.

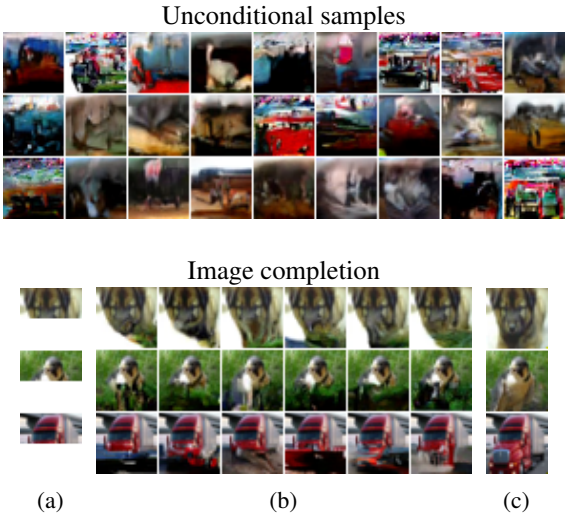

Figure 4: Unconditional samples and image completions generated by our method for CIFAR-10. (a) depicts the occluded orignal images, (b) the completions and (c) the original. As the sequence length grows linear transformers become more efficient compared to softmax attention. Our model achieves more than **4,000 times** higher throughput and generates **17.85 images/second**. For details see § 4.2.2.

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.

gressive fashion. We use the 80 hour WSJ dataset (Paul & Baker, 1992) with 40-dimensional mel-scale filterbanks without temporal differences as features. The dataset contains sequences with 800 frames on average and a maximum sequence length of 2,400 frames. For this task, we also compare with a bidirectional LSTM (Hochreiter & Schmidhuber, 1997) with 3 layers of hidden size 320. We use the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 10^{-3} which is reduced when the validation error stops decreasing. For the transformer models, we use 9 layers with 6 heads with the same embedding dimensions as for the image experiments. As an optimizer, we use RAdam with an initial learning rate of 10^{-4} that is divided by 2 when the validation error stops decreasing.

All models are evaluated in terms of phoneme error rate (PER) and training time per epoch. We observe that linear outperforms the recurrent network baseline and Reformer both in terms of performance and speed by a large margin, as seen in table 3. Note that the softmax transformer, achieves lower phone error rate in comparison to all baselines, but is significantly slower. In particular, *linear transformer* is more than $3\times$ faster per epoch. We provide training evolution plots in the supplementary.

5. Conclusions

In this work, we presented *linear transformer*, a model that significantly reduces the memory and computational cost of the original transformers. In particular, by exploiting the associativity property of matrix products we are able to compute the self-attention in time and memory that scales linearly with respect to the sequence length. We show that our model can be used with causal masking and still retain its linear asymptotic complexities. Finally, we express the transformer model as a recurrent neural network, which allows us to perform inference on autoregressive tasks thousands of time faster.

This property opens a multitude of directions for future research regarding the storage and retrieval of information in both RNNs and transformers. Another line of research to be explored is related to the choice of feature map for linear attention. For instance, approximating the RBF kernel with random Fourier features could allow us to use models pretrained with softmax attention.

Supplementary Material for Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention

A. Gradient Derivation

In the first section of our supplementary material, we derive in detail the gradients for causally masked linear transformers and show that they can be computed in linear time and constant memory. In particular, we derive the gradients of a scalar loss with respect to the numerator of the following equation,

$$V_i' = \frac{\phi\left(Q_i\right)^T \sum_{j=1}^{i} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{i} \phi\left(K_j\right)}. \tag{21}$$

The gradient with respect to the denominator and the fraction are efficiently handled by autograd. Without loss of generality, we can assume that Q and K already contain the vectors mapped by $\phi\left(\cdot\right)$, hence given the numerator

$$\bar{V}_i = Q_i^T \sum_{j=1}^{i} K_j V_j^T, \tag{22}$$

and $\nabla_{\bar{V}}\mathcal{L}$ we seek to compute $\nabla_Q\mathcal{L}$, $\nabla_K\mathcal{L}$ and $\nabla_V\mathcal{L}$. Note that $Q \in \mathbb{R}^{N\times D}$, $K \in \mathbb{R}^{N\times D}$ and $V \in \mathbb{R}^{N\times M}$. To derive the gradients, we first express the above equation for a single element without using vector notation,

$$\bar{V}_{ie} = \sum_{d=1}^{D} Q_{id} \sum_{j=1}^{i} K_{jd} V_{je} = \sum_{d=1}^{D}\sum_{j=1}^{i} Q_{id} K_{jd} V_{je}. \tag{23}$$

Subsequently we can start deriving the gradients for Q by taking the partial derivative for any Q_{lt}, as follows

$$\frac{\partial \mathcal{L}}{\partial Q_{lt}} = \sum_{e=1}^{M} \frac{\partial \mathcal{L}}{\partial \bar{V}_{le}} \frac{\partial \bar{V}_{le}}{\partial Q_{lt}} = \sum_{e=1}^{M} \frac{\partial \mathcal{L}}{\partial \bar{V}_{le}} \left(\sum_{j=1}^{l} K_{jt} V_{je} \right). \tag{24}$$

If we write the above equation as a matrix product of gradients it becomes,

$$\nabla_{Q_i}\mathcal{L} = \nabla_{\bar{V}_i}\mathcal{L} \left(\sum_{j=1}^{i} K_j V_j^T \right)^T, \tag{25}$$

proving equation 13 from the main paper. In equation 24 we made use of the fact that Q_{lt} only affects $\bar{V}_l$ hence we do not need to sum over i to compute the gradients. However, for K and V this is not the case. In particular, K_j affects all $\bar{V}_i$ where $i \geq j$. Consequently, we can write the partial derivative of the loss with respect to K_{lt} as follows,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial K_{lt}} &= \sum_{e=1}^{M}\sum_{i=l}^{N} \frac{\partial \mathcal{L}}{\partial \bar{V}_{ie}} \frac{\partial \bar{V}_{ie}}{\partial K_{lt}} = \sum_{e=1}^{M}\sum_{i=l}^{N} \frac{\partial \mathcal{L}}{\partial \bar{V}_{ie}} \frac{\partial \left(\sum_{d=1}^{D}\sum_{j=1}^{i} Q_{id} K_{jd} V_{je} \right)}{\partial K_{lt}} \\ &= \sum_{e=1}^{M}\sum_{i=l}^{N} \frac{\partial \mathcal{L}}{\partial \bar{V}_{ie}} Q_{it} V_{le}. \end{aligned} \tag{26}$$

As for Q we can now write the gradient in vectorized form,

$$\nabla_{K_i}\mathcal{L} = \left(\sum_{j=i}^{N} Q_j \left(\nabla_{\bar{V}_j}\mathcal{L} \right)^T \right) V_i, \tag{27}$$