Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas and François Fleuret

ICML, July 2020

https://linear-transformers.com/

Funded by
Transformers are performant

Transformer models have demonstrated impressive performance on

- **NLP** (Vaswani et al., 2017; Devlin et al., 2019; Dai et al., 2019; Yang et al., 2019; Radford et al., 2019)
  - Neural Machine Translation
  - Question Answering
  - Textual Entailment

- Speech & audio processing (Sperber et al., 2018)
- Autoregressive image generation and general computer vision (Child et al., 2019; Parmar et al., 2019; Carion et al., 2020; Cordonnier et al., 2020)
Transformers are performant

Transformer models have demonstrated impressive performance on

- **NLP** (Vaswani et al., 2017; Devlin et al., 2019; Dai et al., 2019; Yang et al., 2019; Radford et al., 2019)
  - Neural Machine Translation
  - Question Answering
  - Textual Entailment
- **Speech & audio processing** (Sperber et al., 2018)
- **Autoregressive image generation and general computer vision** (Child et al., 2019; Parmar et al., 2019; Carion et al., 2020; Cordonnier et al., 2020)
Transformers are hard to scale

Self-attention computation and memory scales as $O(N^2)$ with respect to the sequence length.

A single self-attention layer in an NVIDIA GTX 1080 Ti
Our contributions in a nutshell

- A transformer model with linear complexity both for memory and computation during training
Our contributions in a nutshell

- A transformer model with \textit{linear complexity} both for memory and computation \textit{during training}

- A transformer model with \textit{linear computational complexity and constant memory} for \textit{autoregressive inference}
Our contributions in a nutshell

- A transformer model with \textit{linear complexity} both for memory and computation \textit{during training}
- A transformer model with \textit{linear computational complexity and constant memory} for \textit{autoregressive inference}
- Unravel the \textit{relation between transformers and RNNs}
Definition of a transformer

A. Katharopoulos

Transformers are RNNs.
Definition of a transformer

A. Katharopoulos

Transformers are RNNs
Definition of a transformer

A. Katharopoulos

Transformers are RNNs
The commonly used attention mechanism is the scaled dot product attention

\[ Q = XW_Q \]
\[ K = XW_K \]
\[ V = XW_V \]

\[ A_l(X) = V' = \text{softmax} \left( \frac{QK^T}{\sqrt{D}} \right) V \]
Self-Attention

The commonly used attention mechanism is the scaled dot product attention

\[ Q = XW_Q \]
\[ K = XW_K \]
\[ V = XW_V \]

\[ A_l(X) = V' = \text{softmax} \left( \frac{QK^T}{\sqrt{D}} \right) V \]
Self-Attention

The commonly used attention mechanism is the scaled dot product attention

\[
Q = XW_Q \\
K = XW_K \\
V = XW_V \\
A_i(X) = V' = \text{softmax}\left( \frac{QK^T}{\sqrt{D}} \right) V
\]

Quadratic complexity
Linear Attention

What if we write the self-attention using an \textbf{arbitrary similarity score}?

\[
V'_i = \frac{\sum_{j=1}^{N} \text{sim} (Q_i, K_j) V_j}{\sum_{j=1}^{N} \text{sim} (Q_i, K_j)}
\]
Linear Attention

What if this similarity is a kernel, namely \( \text{sim}(a, b) = \phi(a)^T \phi(b) \)?

\[
V'_i = \frac{\sum_{j=1}^{N} \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^{N} \text{sim}(Q_i, K_j)}
\]

Kernelization

\[
= \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)}
\]
Matrix products are associative which makes the attention computation $O(N)$ with respect to the sequence length.

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

Kernelization

$$= \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)}$$

Associativity property

$$= \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}$$
Causal Masking

Causal masking is used to efficiently train autoregressive transformers.
Causal Masking

Causal masking is used to efficiently train autoregressive transformers.

**Non-autoregressive**

\[
V_i' = \frac{\sum_{j=1}^{N} \text{sim} (Q_i, K_j) V_j}{\sum_{j=1}^{N} \text{sim} (Q_i, K_j)}
\]

**Autoregressive**

\[
V_i' = \frac{\sum_{j=1}^{i} \text{sim} (Q_i, K_j) V_j}{\sum_{j=1}^{i} \text{sim} (Q_i, K_j)}
\]
Causal Masking

Causal masking is used to efficiently train autoregressive transformers.

\[
V'_i = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}
\]

Non-autoregressive

\[
V'_i = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}
\]

Autoregressive
Causal Masking

Causal masking is used to efficiently train autoregressive transformers.

Non-autoregressive

\[ V'_i = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)} \]

Autoregressive

\[ V'_i = \frac{\phi(Q_i)^T \sum_{j=1}^{i} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{i} \phi(K_j)} \]
Causal Masking

Causal masking is used to efficiently train autoregressive transformers.

\[
V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}
\]

Non-autoregressive

\[
S = \frac{\sum_{j=1}^{N} \phi(K_j) V_j^T}{\sum_{j=1}^{N} \phi(K_j)}
\]

Autoregressive

\[
V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{i} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{i} \phi(K_j)}
\]

\[
S_i = \frac{\sum_{j=1}^{i} \phi(K_j) V_j^T}{\sum_{j=1}^{i} \phi(K_j)}
\]

Naive computation of \( S_i \) and \( Z_i \) results in quadratic complexity.
Autoregressive transformers can be written as a function that **receives an input** $x_i$, **modifies the internal state** $\{s_{i-1}, z_{i-1}\}$ and **predicts an output** $y_i$. 

![Diagram](https://via.placeholder.com/150)
Transformers are RNNs

Autoregressive transformers can be written as a function that receives an input $x_i$, modifies the internal state $\{s_{i-1}, z_{i-1}\}$ and predicts an output $y_i$. 

![Diagram of autoregressive transformers]
Transformers are RNNs

Autoregressive transformers can be written as a function that receives an input $x_i$, modifies the internal state $\{s_{i-1}, z_{i-1}\}$ and predicts an output $y_i$. 
Transformers are RNNs

Autoregressive transformers can be written as a function that receives an input $x_i$, modifies the internal state $\{s_{i-1}, z_{i-1}\}$ and predicts an output $y_i$.

Autoregressive inference with linear complexity and constant memory.
Practical implications

- Our *theoretical analysis holds for all transformers* even when using infinite dimensional feature maps
Practical implications

- Our theoretical analysis holds for all transformers even when using infinite dimensional feature maps.
- We need a simple finite dimensional feature map to speed up computation.
Practical implications

- Our theoretical analysis holds for all transformers even when using infinite dimensional feature maps.
- We need a simple finite dimensional feature map to speed up computation.
- We derive the gradients as cumulative sums which allows for a significant speed-up.
Experimental setup

Baselines
- Softmax transformer (Vaswani et al., 2017)
- LSH attention from Reformer (Kitaev et al., 2020)

Experiments
- Artificial benchmark for computational and memory requirements
- Autoregressive image generation on MNIST and CIFAR-10
- Automatic speech recognition on Wall Street Journal
Experimental setup

Baselines

▶ Softmax transformer (Vaswani et al., 2017)
▶ LSH attention from Reformer (Kitaev et al., 2020)

Experiments

▶ Artificial benchmark for computational and memory requirements
▶ Autoregressive image generation on MNIST and CIFAR-10
▶ Automatic speech recognition on Wall Street Journal
Experimental setup

Baselines
- Softmax transformer (Vaswani et al., 2017)
- LSH attention from Reformer (Kitaev et al., 2020)

Experiments
- Artificial benchmark for computational and memory requirements
- Autoregressive image generation on MNIST and CIFAR-10
- Automatic speech recognition on Wall Street Journal
Benchmark

A. Katharopoulos Transformers are RNNs
Benchmark

A. Katharopoulos Transformers are RNNs
Autoregressive image generation

Unconditional samples after 250 epochs on MNIST

- Ours (0.644 bpd)
- Softmax (0.621 bpd)
- LSH-1 (0.745 bpd)
- LSH-4 (0.676 bpd)

Unconditional samples after 1 GPU week on CIFAR-10

- Ours (3.40 bpd)
- Softmax (3.47 bpd)
- LSH-1 (3.39 bpd)
- LSH-4 (3.51 bpd)
Autoregressive image generation

Unconditional samples after 250 epochs on MNIST

- Ours (0.644 bpd)
- Softmax (0.621 bpd)
- LSH-1 (0.745 bpd)
- LSH-4 (0.676 bpd)

Unconditional samples after 1 GPU week on CIFAR-10

- Ours (3.40 bpd)
- Softmax (3.47 bpd)
- LSH-1 (3.39 bpd)
- LSH-4 (3.51 bpd)
Autoregressive image generation

Unconditional samples after 250 epochs on MNIST

- Ours (0.644 bpd)
- Softmax (0.621 bpd)
- LSH-1 (0.745 bpd)
- LSH-4 (0.676 bpd)

Unconditional samples after 1 GPU week on CIFAR-10

- Ours (3.40 bpd)
- Softmax (3.47 bpd)
- LSH-1 (3.39 bpd)
- LSH-4 (3.51 bpd)
Autoregressive image generation

MNIST

Images / second

10^0 10^1 10^2

softmax lsh-1 ours

100 10^1 10^2

CIFAR-10

Images / second

10^0 10^1 10^{-1} 10^{-2}

softmax lsh-1 ours

10^{-2} 10^{-1} 10^0
Autoregressive image generation

MNIST

Images / second

10^0
10^1
10^2

softmax lsh-1 ours

10^0 10^-1 10^-2

CIFAR-10

Images / second

10^0
10^-1
10^1

softmax lsh-1 ours

10^-2
Autoregressive image generation

MNIST

Images / second

CIFAR-10

Images / second

A. Katharopoulos  Transformers are RNNs
Automatic speech recognition

**Error rate relative to softmax**

<table>
<thead>
<tr>
<th>Method</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bi-lstm</td>
<td>2.0</td>
</tr>
<tr>
<td>lsh-4</td>
<td>1.5</td>
</tr>
<tr>
<td>ours</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Lower is better

**Speedup relative to softmax**

<table>
<thead>
<tr>
<th>Method</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bi-lstm</td>
<td>0</td>
</tr>
<tr>
<td>lsh-4</td>
<td>1</td>
</tr>
<tr>
<td>ours</td>
<td>3</td>
</tr>
</tbody>
</table>

Higher is better

A. Katharopoulos Transformers are RNNs
**Automatic speech recognition**

- **Error rate relative to softmax**
  - bi-lstm: 2.0
  - lsh-4: 1.5
  - ours: 1.0

- **Speedup relative to softmax**
  - bi-lstm: 0
  - lsh-4: 1
  - ours: 2

Lower is better for error rate, higher is better for speedup.
Automatic speech recognition

Error rate relative to softmax

<table>
<thead>
<tr>
<th>Method</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>bi-lstm</td>
<td>2.0</td>
</tr>
<tr>
<td>lsh-4</td>
<td>1.5</td>
</tr>
<tr>
<td>ours</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Lower is better

Speedup relative to softmax

<table>
<thead>
<tr>
<th>Method</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>bi-lstm</td>
<td>0</td>
</tr>
<tr>
<td>lsh-4</td>
<td>1</td>
</tr>
<tr>
<td>ours</td>
<td>3</td>
</tr>
</tbody>
</table>

Higher is better
Automatic speech recognition

Error rate relative to softmax

- bi-lstm: 2.0
- lsh-4: 1.6
- ours: 1.0

Lower is better

Speedup relative to softmax

- bi-lstm: 1.0
- lsh-4: 0.5
- ours: 3.0

Higher is better

A. Katharopoulos Transformers are RNNs
Kernel feature maps and matrix associativity yield an attention with linear complexity.

Computing the key value matrix as a cumulative sum extends our efficient attention computation to the autoregressive case.

Using the RNN formulation to perform autoregressive inference requires constant memory and is many times faster.
Kernel feature maps and matrix associativity yield an attention with linear complexity.

Computing the key value matrix as a cumulative sum extends our efficient attention computation to the autoregressive case.

Using the RNN formulation to perform autoregressive inference requires constant memory and is many times faster.
Kernel feature maps and matrix associativity yield an attention with linear complexity.

Computing the key value matrix as a cumulative sum extends our efficient attention computation to the autoregressive case.

Using the RNN formulation to perform autoregressive inference requires constant memory and is many times faster.
Summary

- **Kernel feature maps** and **matrix associativity** yield an attention with linear complexity.
- Computing the key value matrix as a **cumulative sum** extends our efficient attention computation to the autoregressive case.
- Using the RNN formulation to perform autoregressive inference requires **constant memory** and is **many times faster**.
from fast_transformers.builders import TransformerEncoderBuilder
linear_bert = TransformerEncoderBuilder.from_kwargs(
    n_layers=12,
    n_heads=12,
    query_dimensions=64,
    value_dimensions=64,
    feed_forward_dimensions=3072,
    attention_type="linear",
).get()

# dummy 4000 long sequence
y = linear_bert(torch.randn(10, 4000, 768))


References II


References III


