REFORMER: The Efficient Transformer

2020. 01. 30

LG 사이언스파크 AI 머신러닝팀

전창욱

들어가며

- 작년 말 인터뷰에서 구글 AI의 제프 딘은 큰 맥락이 구글의 향후 작업의 주요 초점이 될것이라고 말했다. “지금 처럼 BERT 및 기타 모델은 수백 개의 단어에서 잘 작동하지만 10,000 단어는 작동하지 않는다.”

- 입력 단어가 많을수록 보다 넓은 문맥을 고려한다. 100만 단어까지 입력을 받는다.

- ICLR 2020에 승인된 논문이다.

- **Feed Forward**

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- inner-layer

- **Attention**

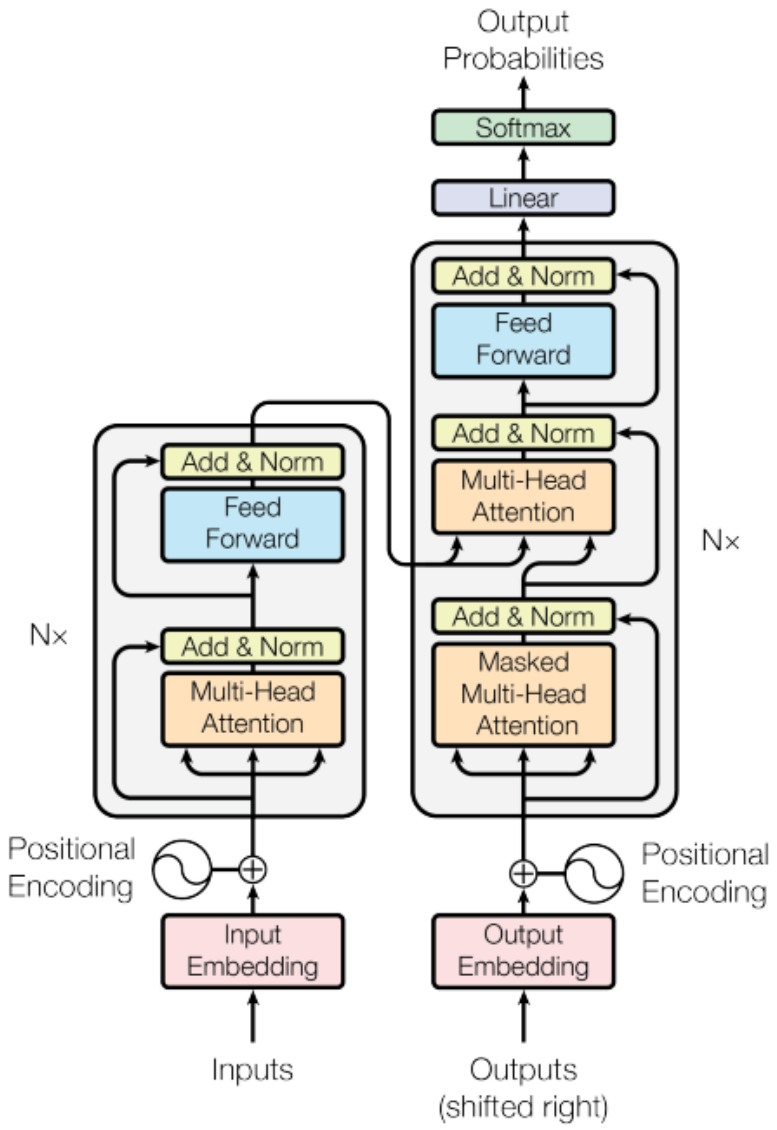


Figure 1: The Transformer - model architecture.

ABSTRACT

- Dot-product attention에 locality-sensitive hashing(LSH) 사용

  - Complexity $O(L^2)$ -> $O(L \log L)$

- Reversible Residual Layers (RevNet) 사용

  - N time에 activations only once

- 긴 sequences에 대해 Memofy-efficient 그리고 더 faster 학습 가능

REFORMER: THE EFFICIENT TRANSFORMER

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Transformer models routinely achieve state-of-the-art results on a number of tasks but training these models can be prohibitively costly, especially on long sequences. We introduce two techniques to improve the efficiency of Transformers. For one, we replace dot-product attention by one that uses locality-sensitive hashing, changing its complexity from $O(L^2)$ to $O(L \log L)$, where L is the length of the sequence. Furthermore, we use reversible residual layers instead of the standard residuals, which allows storing activations only once in the training process instead of N times, where N is the number of layers. The resulting model, the Reformer, performs on par with Transformer models while being much more memory-efficient and much faster on long sequences.

1 INTRODUCTION

The Transformer architecture (Vaswani et al., 2017) is widely used in natural language processing and yields state-of-the-art results on a number of tasks. To obtain these results, researchers have resorted to training ever larger Transformer models. The number of parameters exceeds 0.5B per layer in the largest configuration reported in (Shazeer et al., 2018) while the number of layers goes up to 64 in (Al-Rfou et al., 2018). Transformer models are also used on increasingly long sequences. Up to 11 thousand tokens of text in a single example were processed in (Liu et al., 2018) and when processing other modalities, like music (Huang et al., 2018) and images (Parmar et al., 2018), even longer sequences are commonplace. These large-scale long-sequence models yield great results but strain resources to the point where some argue that this trend is breaking NLP research[1]. Many large Transformer models can only realistically be trained in large industrial research laboratories and such models trained with model parallelism cannot even be fine-tuned on a single GPU as their memory requirements demand a multi-accelerator hardware setup even for a single training step.

Do large Transformer models fundamentally require such huge resources or are they simply inefficient? Consider the following calculation: the 0.5B parameters used in the largest reported Transformer layer account for 2GB of memory. Activations for 64K tokens with embedding size 1024 and batch size 8 account for $64\text{K} \times 1\text{K} \times 8 = 0.5\text{B}$ floats, requiring another 2GB of memory. If our memory use was only per-layer, then we should fairly easily fit a large Transformer even on sequences of length 64K on a single accelerator. Further, the whole corpus used to train BERT only requires 17GB to store. Why is it then that we cannot even fine-tune these models on single machines?

The above estimate includes only per-layer memory and input activations cost and does not take into account the following major sources of memory use in the Transformer.

- Memory in a model with N layers is N-times larger than in a single-layer model due to the fact that activations need to be stored for back-propagation.
- Since the depth d_{ff} of intermediate feed-forward layers is often much larger than the depth d_{model} of attention activations, it accounts for a large fraction of memory use.
- Attention on sequences of length L is $O(L^2)$ in both computational and memory complexity, so even for a single sequence of 64K tokens can exhaust accelerator memory.

We introduce the Reformer model which solves these problems using the following techniques:

[1] https://hackingsemantics.xyz/2019/leaderboards/

INTRODUCTION (문제)

- large Transformer models (각 layer 마다 0.5B parameters, 64 layer, 1,1000 tokens)

- 0.5B = 64K(token) * 1K(embedding size) * 8(batch size) (2GB)
  - layer에 필요한 비용

- Transformer 주요 memory 사용 비용
  - back-propagation을 위해 N layer model 모두 저장
  - Feed-Forward layers 의 d_{model} , d_{ff} 메모리 사용
  - Attention 은 sequences 길이 만큼 (L^2)의 메모리 비용과 계산 비용이 필요

REFORMER: THE EFFICIENT TRANSFORMER

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Transformer models routinely achieve state-of-the-art results on a number of tasks but training these models can be prohibitively costly, especially on long sequences. We introduce two techniques to improve the efficiency of Transformers. For one, we replace dot-product attention by one that uses locality-sensitive hashing, changing its complexity from $O(L^2)$ to $O(L \log L)$, where L is the length of the sequence. Furthermore, we use reversible residual layers instead of the standard residuals, which allows storing activations only once in the training process instead of N times, where N is the number of layers. The resulting model, the Reformer, performs on par with Transformer models while being much more memory-efficient and much faster on long sequences.

1 INTRODUCTION

The Transformer architecture (Vaswani et al., 2017) is widely used in natural language processing and yields state-of-the-art results on a number of tasks. To obtain these results, researchers have resorted to training ever larger Transformer models. The number of parameters exceeds 0.5B per layer in the largest configuration reported in (Shazeer et al., 2018) while the number of layers goes up to 64 in (Al-Rfou et al., 2018). Transformer models are also used on increasingly long sequences. Up to 11 thousand tokens of text in a single example were processed in (Liu et al., 2018) and when processing other modalities, like music (Huang et al., 2018) and images (Parmar et al., 2018), even longer sequences are commonplace. These large-scale long-sequence models yield great results but strain resources to the point where some argue that this trend is breaking NLP research[1]. Many large Transformer models can only realistically be trained in large industrial research laboratories and such models trained with model parallelism cannot even be fine-tuned on a single GPU as their memory requirements demand a multi-accelerator hardware setup even for a single training step.

Do large Transformer models fundamentally require such huge resources or are they simply inefficient? Consider the following calculation: the 0.5B parameters used in the largest reported Transformer layer account for 2GB of memory. Activations for 64K tokens with embedding size 1024 and batch size 8 account for $64K \times 1K \times 8 = 0.5B$ floats, requiring another 2GB of memory. If our memory use was only per-layer, then we should fairly easily fit a large Transformer even on sequences of length 64K on a single accelerator. Further, the whole corpus used to train BERT only requires 17GB to store. Why is it then that we cannot even fine-tune these models on single machines?

The above estimate includes only per-layer memory and input activations cost and does not take into account the following major sources of memory use in the Transformer.

- Memory in a model with N layers is N-times larger than in a single-layer model due to the fact that activations need to be stored for back-propagation.
- Since the depth d_{ff} of intermediate feed-forward layers is often much larger than the depth d_{model} of attention activations, it accounts for a large fraction of memory use.
- Attention on sequences of length L is $O(L^2)$ in both computational and memory complexity, so even for a single sequence of 64K tokens can exhaust accelerator memory.

We introduce the Reformer model which solves these problems using the following techniques:

[1] https://hackingsemantics.xyz/2019/leaderboards/

INTRODUCTION (해결 방법)

- **The Reversible Residual Network: Backpropagation Without Storing Activations (Toronto 2017 Gomez 발표)**

- **Feed-Forward layers를 chunks 단위로 처리**

- **LSH를 사용하여 Attention 부분을 O(L^2) -> O(L log L)**

- **두 가지 데이터를 사용하여 확인**

  - text task (enwik8) 64k sequences length

  - image generation task(imagenet-64 generation) 12K sequences length

- Reversible layers, first introduced in Gomez et al. (2017), enable storing only a single copy of activations in the whole model, so the N factor disappears.
- Splitting activations inside feed-forward layers and processing them in chunks removes the d_{ff} factor and saves memory inside feed-forward layers.
- Approximate attention computation based on locality-sensitive hashing replaces the O(L^2) factor in attention layers with O(L) and so allows operating on long sequences.

We study these techniques and show that they have negligible impact on the training process compared to the standard Transformer. Splitting activations in fact only affects the implementation; it is numerically identical to the layers used in the Transformer. Applying reversible residuals instead of the standard ones does change the model but has a negligible effect on training in all configurations we experimented with. Finally, locality-sensitive hashing in attention is a more major change that can influence the training dynamics, depending on the number of concurrent hashes used. We study this parameter and find a value which is both efficient to use and yields results very close to full attention.

We experiment on a synthetic task, a text task (enwik8) with sequences of length 64K and an image generation task (imagenet-64 generation) with sequences of length 12K. In both cases we show that Reformer matches the results obtained with full Transformer but runs much faster, especially on the text task, and with orders of magnitude better memory efficiency.

2 LOCALITY-SENSITIVE HASHING ATTENTION

Dot-product attention. The standard attention used in the Transformer is the scaled dot-product attention (Vaswani et al., 2017). The input consists of queries and keys of dimension d_k, and values of dimension d_v. The dot products of the query with all keys are computed, scaled by $\sqrt{d_k}$, and a softmax function is applied to obtain the weights on the values. In practice, the attention function on a set of queries is computed simultaneously, packed together into a matrix Q. Assuming the keys and values are also packed together into matrices K and V, the matrix of outputs is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

Multi-head attention. In the Transformer, instead of performing a single attention function with d_{model}-dimensional keys, values and queries, one linearly projects the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. Attention is applied to each of these projected versions of queries, keys and values in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values. This mechanism is known as multi-head attention.

Memory-efficient attention. To calculate the memory use of the attention mechanism, let us focus on the attention computation from Equation 1. Let us assume that Q, K and V all have the shape $[batch\_size, length, d_{model}]$. The main issue is the term QK^T, which has the shape $[batch\_size, length, length]$. In the experimental section we train a model on sequences of length $64K$ – in this case, even at batch-size of 1, this is a $64K \times 64K$ matrix, which in 32-bit floats would take 16GB of memory. This is impractical and has hindered the use of the Transformer for long sequences. But it is important to note that the QK^T matrix does not need to be fully materialized in memory. The attention can indeed be computed for each query q_i separately, only calculating $\text{softmax}(\frac{q_i K^T}{\sqrt{d_k}})V$ once in memory, and then re-computing it on the backward pass when needed for gradients. This way of computing attention may be less efficient but it only uses memory proportional to $length$. We use this memory-efficient implementation of attention to run the full-attention baselines presented in the experimental section.

Where do Q, K, V come from? The multi-head attention described above operates on keys, queries and values, but usually we are only given a single tensor of activations A of the shape $[batch\_size, length, d_{model}]$ – e.g., coming from embedding the tokens in a sentence into vectors. To build Q, K and V from A, the Transformer uses 3 different linear layers projecting A into Q, K and V with different parameters. For models with LSH attention, we want queries and keys (Q and

Locality-Sensitive Hashing Attention

- **Dot-product attention**

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

- **Multi-head attention**
  - d_q, d_k, d_v linear projection
  - head size 크기로 reshape
  - parallel 하게 attention 적용
  - output 값을 concatenated
  - 한번 더 projection 한다.
- **Memory-efficient attention**
  - issue QK^t 에서 발생
  - Q, K, V shape [batch_size, length, d_{model}]
  - QK^t 값 [batch_size, length, length]
  - 64K * 64K matix (32-bit floats) = 16GB (memory)

$$\text{softmax}(\frac{q_i K^T}{\sqrt{d_k}})V$$

- Reversible layers, first introduced in Gomez et al. (2017), enable storing only a single copy of activations in the whole model, so the N factor disappears.
- Splitting activations inside feed-forward layers and processing them in chunks removes the d_{ff} factor and saves memory inside feed-forward layers.
- Approximate attention computation based on locality-sensitive hashing replaces the O(L^2) factor in attention layers with O(L) and so allows operating on long sequences.

We study these techniques and show that they have negligible impact on the training process compared to the standard Transformer. Splitting activations in fact only affects the implementation; it is numerically identical to the layers used in the Transformer. Applying reversible residuals instead of the standard ones does change the model but has a negligible effect on training in all configurations we experimented with. Finally, locality-sensitive hashing in attention is a more major change that can influence the training dynamics, depending on the number of concurrent hashes used. We study this parameter and find a value which is both efficient to use and yields results very close to full attention.

We experiment on a synthetic task, a text task (enwik8) with sequences of length 64K and an image generation task (imagenet-64 generation) with sequences of length 12K. In both cases we show that Reformer matches the results obtained with full Transformer but runs much faster, especially on the text task, and with orders of magnitude better memory efficiency.

2 LOCALITY-SENSITIVE HASHING ATTENTION

Dot-product attention. The standard attention used in the Transformer is the scaled dot-product attention (Vaswani et al., 2017). The input consists of queries and keys of dimension d_k, and values of dimension d_v. The dot products of the query with all keys are computed, scaled by $\sqrt{d_k}$, and a softmax function is applied to obtain the weights on the values. In practice, the attention function on a set of queries is computed simultaneously, packed together into a matrix Q. Assuming the keys and values are also packed together into matrices K and V, the matrix of outputs is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

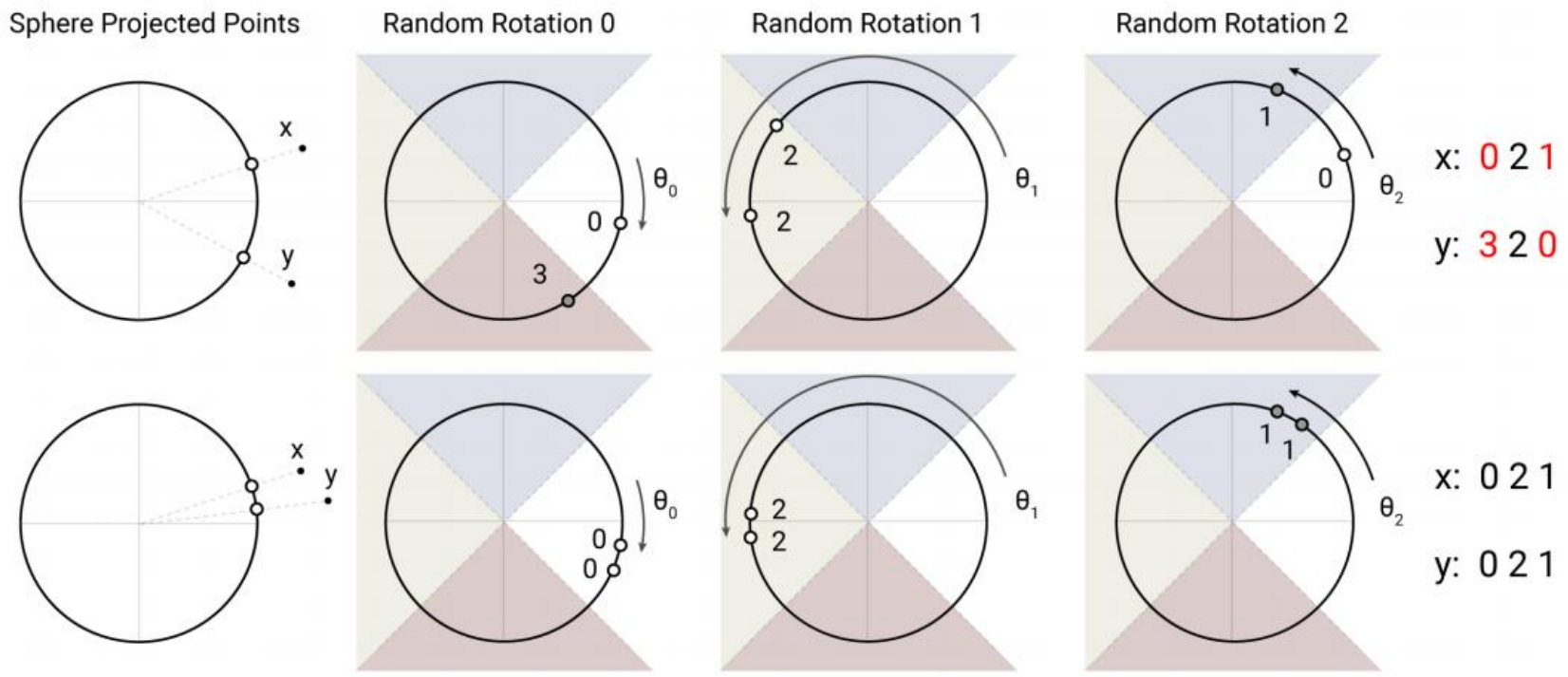
Multi-head attention. In the Transformer, instead of performing a single attention function with d_{model}-dimensional keys, values and queries, one linearly projects the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. Attention is applied to each of these projected versions of queries, keys and values in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values. This mechanism is known as multi-head attention.

Memory-efficient attention. To calculate the memory use of the attention mechanism, let us focus on the attention computation from Equation 1. Let us assume that Q, K and V all have the shape $[batch\_size, length, d_{model}]$. The main issue is the term QK^T, which has the shape $[batch\_size, length, length]$. In the experimental section we train a model on sequences of length $64K$ – in this case, even at batch-size of 1, this is a $64K \times 64K$ matrix, which in 32-bit floats would take 16GB of memory. This is impractical and has hindered the use of the Transformer for long sequences. But it is important to note that the QK^T matrix does not need to be fully materialized in memory. The attention can indeed be computed for each query q_i separately, only calculating $\text{softmax}(\frac{q_i K^T}{\sqrt{d_k}})V$ once in memory, and then re-computing it on the backward pass when needed for gradients. This way of computing attention may be less efficient but it only uses memory proportional to $length$. We use this memory-efficient implementation of attention to run the full-attention baselines presented in the experimental section.

Where do Q, K, V come from? The multi-head attention described above operates on keys, queries and values, but usually we are only given a single tensor of activations A of the shape $[batch\_size, length, d_{model}]$ – e.g., coming from embedding the tokens in a sentence into vectors. To build Q, K and V from A, the Transformer uses 3 different linear layers projecting A into Q, K and V with different parameters. For models with LSH attention, we want queries and keys (Q and

Locality-Sensitive Hashing Attention

- **Angular locality sensitive hash (random rotations)**

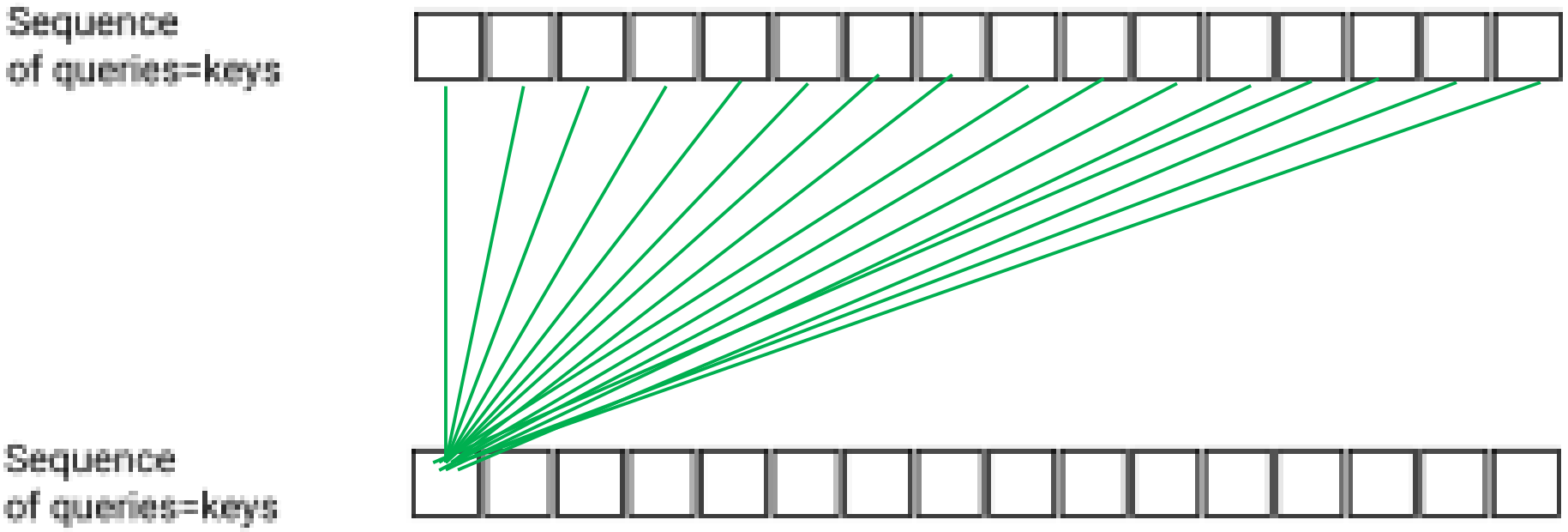


- **Locality sensitive hashing**

- 각 qi vector를 h(x) function에 넣어서 만약 결과 값이 가깝다면 같은 hash 그렇지 않다면 먼 hash 할당

Locality-Sensitive Hashing Attention

- Attention (Complexity $O(L^2)$)



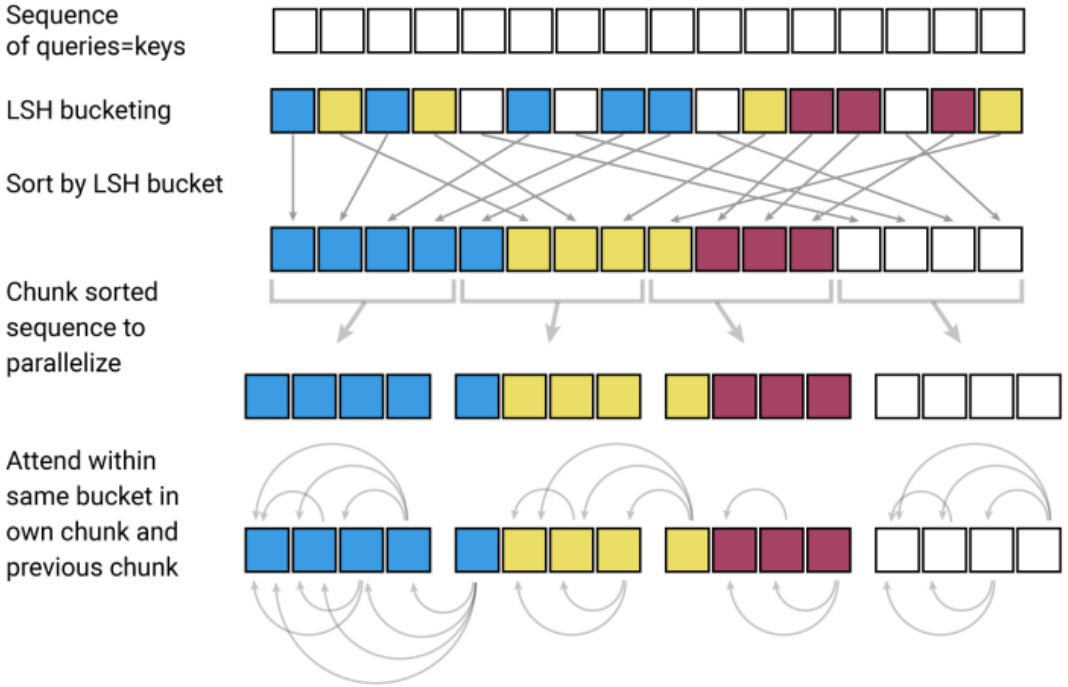
- L size = 16 이며 16^2 가 된다.

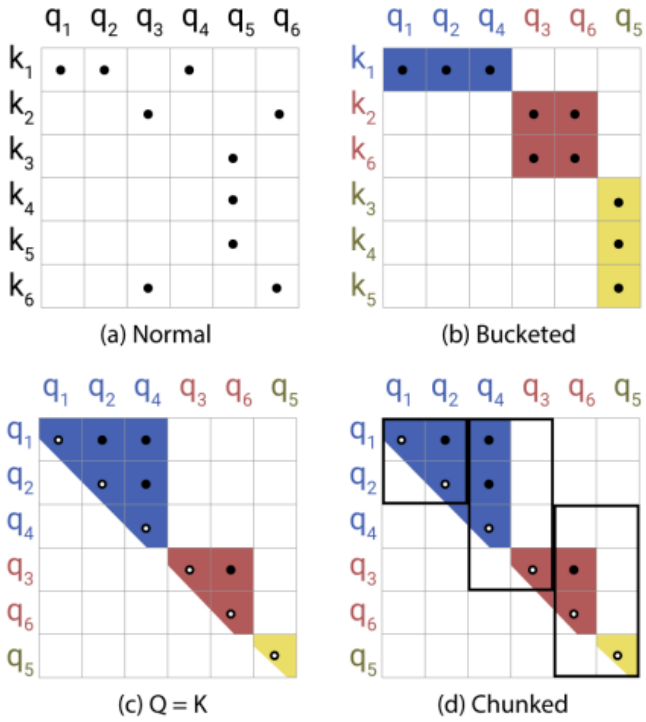
Locality-Sensitive Hashing Attention

- **LSH attention (Complexity O(L log L))**

$$o_i = \sum_{j \in \mathcal{P}_i} \exp\left(q_i \cdot k_j - z(i, \mathcal{P}_i)\right) v_j \qquad \text{where } \mathcal{P}_i = \{j : i \geq j\} \tag{2}$$

$$o_i = \sum_{j \in \widetilde{\mathcal{P}}_i} \exp\left(q_i \cdot k_j - m(j, \mathcal{P}_i) - z(i, \mathcal{P}_i)\right) v_j \qquad \text{where } m(j, \mathcal{P}_i) = \begin{cases} \infty & \text{if } j \notin \mathcal{P}_i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$





Locality-Sensitive Hashing Attention

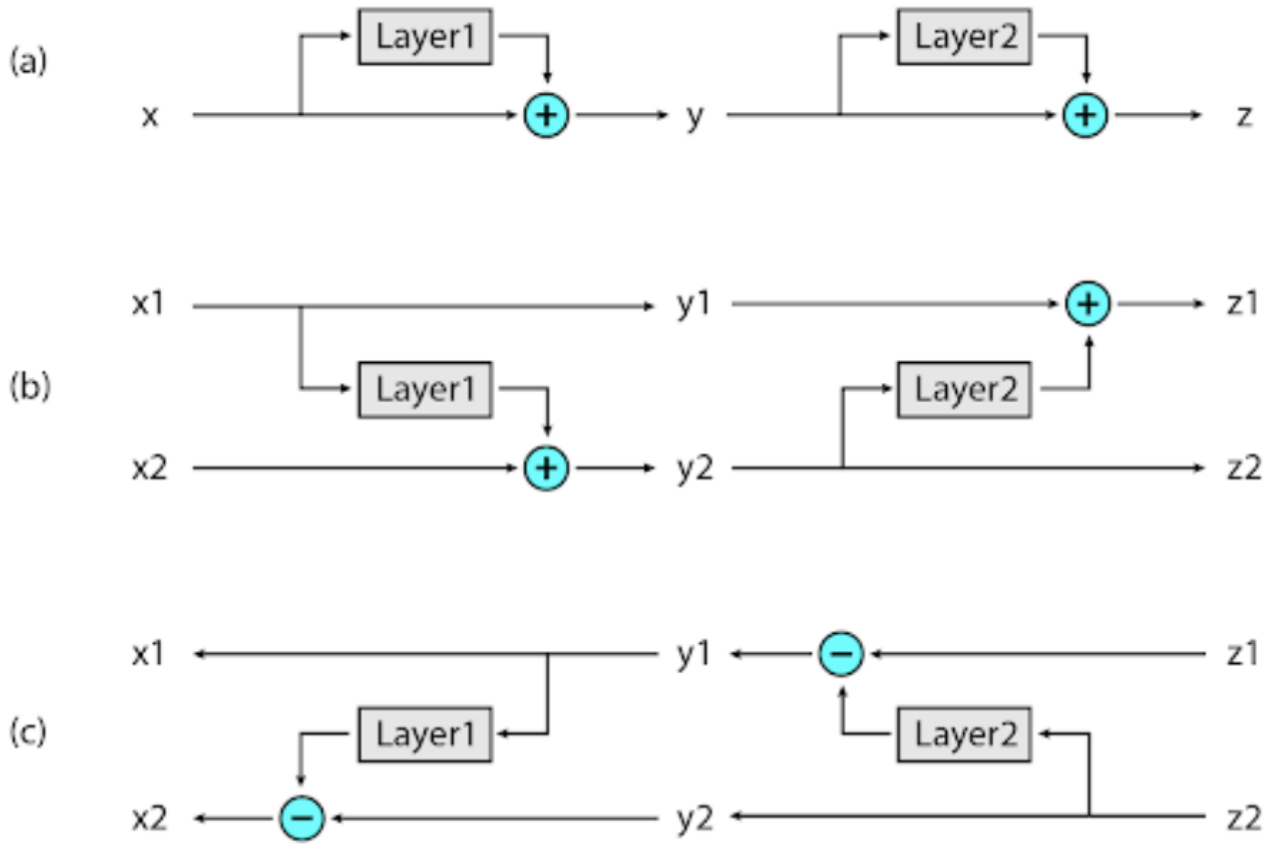
- **Memory and Time complexity**

Table 1: Memory and time complexity of attention variants. We write l for length, b for batch size, n_h for the number of heads, n_c for the number of LSH chunks, n_r for the number of hash repetitions.

Attention Type	Memory Complexity	Time Complexity
Scaled Dot-Product	$\max(bn_hld_k, bn_hl^2)$	$\max(bn_hld_k, bn_hl^2)$
Memory-Efficient	$\max(bn_hld_k, bn_hl^2)$	$\max(bn_hld_k, bn_hl^2)$
LSH Attention	$\max(bn_hld_k, bn_hln_r(4l/n_c)^2)$	$\max(bn_hld_k, bn_hn_rl(4l/n_c)^2)$

Reversible Transformer

- RevNets



  - $y = x + F(x)$
  - $y_2 = x_2 + F(x_1)$
  - $z_1 = y_1 + G(y_2)$
  - $y_1 = z_1 - G(z_2)$
  - $x_2 = y_2 - F(y_1)$

- Reversible Transformer

$Y_1 = X_1 + \text{Attention}(X_2)$ $\quad\quad$ $Y_2 = X_2 + \text{FeedForward}(Y_1)$

The Reversible Residual Network:
Backpropagation Without Storing Activations (2017 University of Toronto Aidan N. Gomez)

- **Chunking**

$$Y_2 = \left[Y_2^{(1)}; \ldots; Y_2^{(c)}\right] = \left[X_2^{(1)} + \text{FeedForward}(Y_1^{(1)}); \ldots; X_2^{(c)} + \text{FeedForward}(Y_1^{(c)})\right] \quad (10)$$

  - **Chunks 로 잘라서 FF를 진입하면 inner layer에서 memory 감소 기능을 제공 한다.**

- **Memory and time complexity of Transformer variants**

Table 3: Memory and time complexity of Transformer variants. We write d_{model} and d_{ff} for model depth and assume $d_{ff} \geq d_{model}$; b stands for batch size, l for length, n_l for the number of layers. We assume $n_c = l/32$ so $4l/n_c = 128$ and we write $c = 128^2$.

Model Type	Memory Complexity	Time Complexity
Transformer	$\max(bld_{ff}, bn_h l^2)n_l$	$(bld_{ff} + bn_h l^2)n_l$
Reversible Transformer	$\max(bld_{ff}, bn_h l^2)$	$(bn_h l d_{ff} + bn_h l^2)n_l$
Chunked Reversible Transformer	$\max(bld_{model}, bn_h l^2)$	$(bn_h l d_{ff} + bn_h l^2)n_l$
LSH Transformer	$\max(bld_{ff}, bn_h l n_r c)n_l$	$(bld_{ff} + bn_h n_r lc)n_l$
Reformer	$\max(bld_{model}, bn_h l n_r c)$	$(bld_{ff} + bn_h n_r lc)n_l$

- **Effect of sharing QK**
- **Effect of reversible layer**

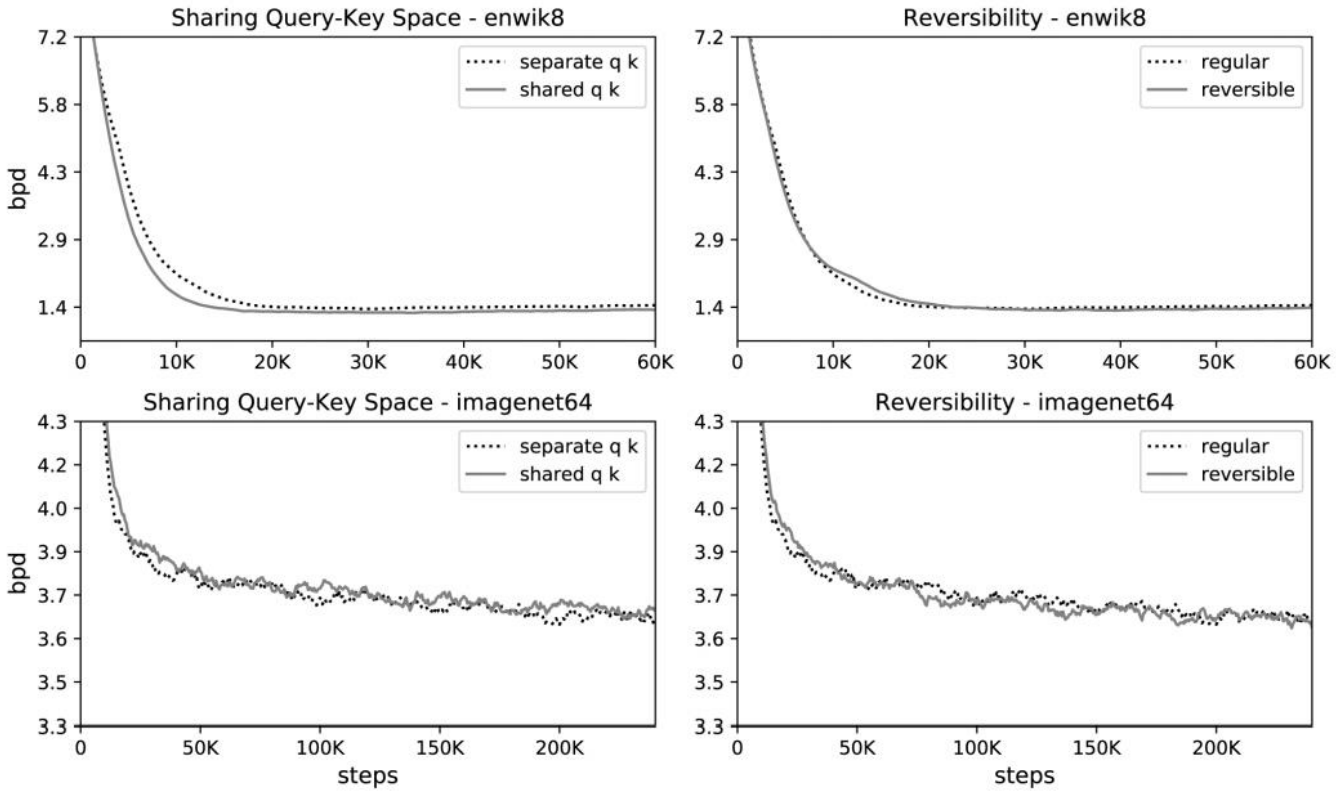


Figure 3: Effect of shared query-key space (left) and reversibility (right) on performance on enwik8 and imagenet64 training. The curves show bits per dim on held-out data.

Experiments

- **LSH attention in Transformer**

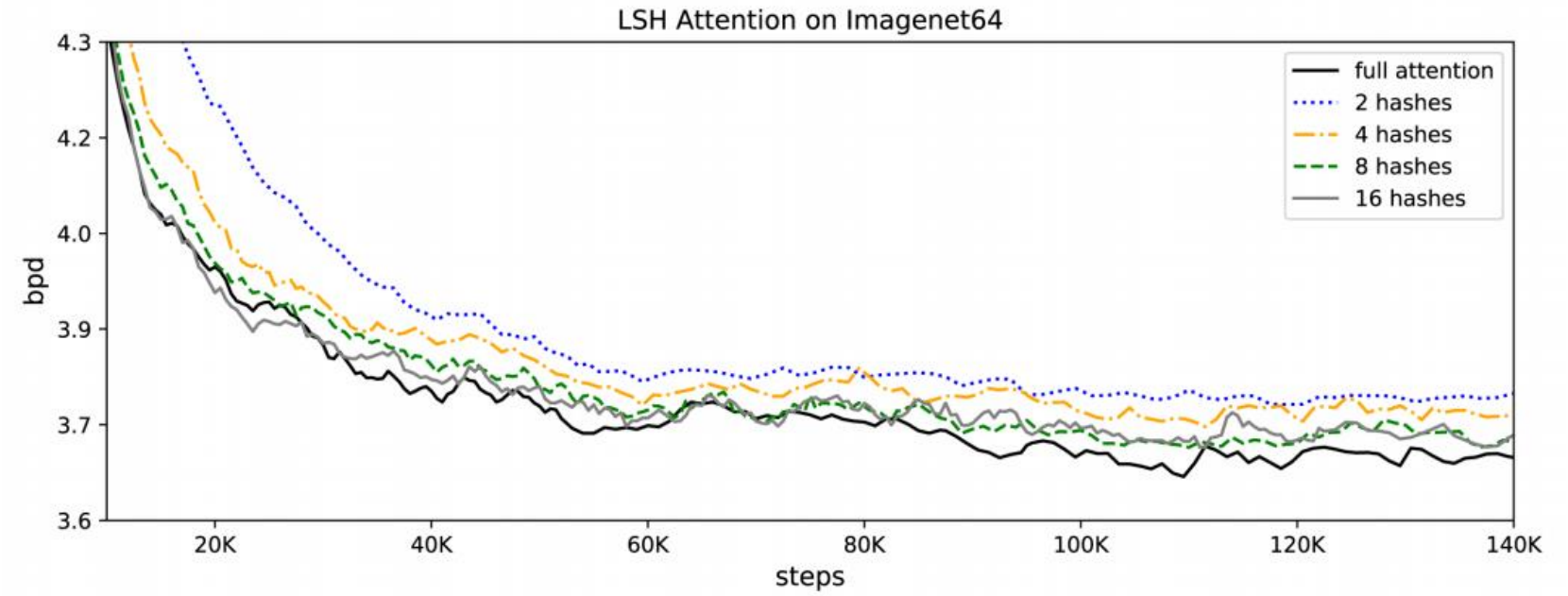


Figure 4: LSH attention performance as a function of hashing rounds on imagenet64.

Experiments

- **Large Reformer model**

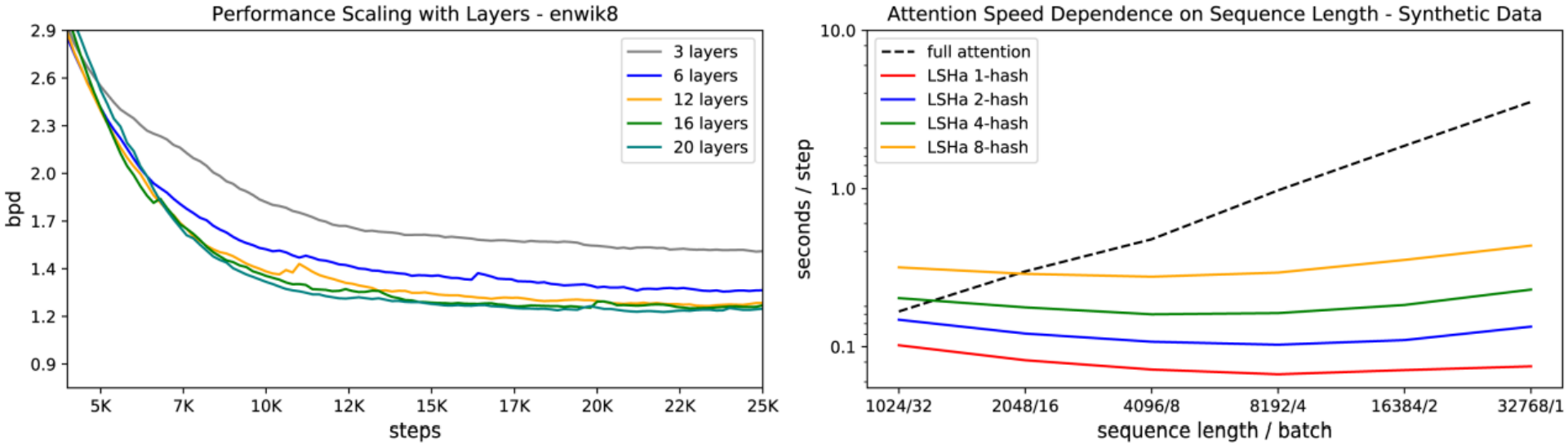


Figure 5: Left: LSH attention performance as a function of number of layers on enwik8. Right: Speed of attention evaluation as a function of input length for full- and LSH- attention.

Applications of Reformer

- Image

https://colab.research.google.com/github/google/trax/blob/master/trax/models/reformer/image_generation.ipynb

- text "Crime and Punishment", by Fyodor Dostoevsky

There was a time when the balcasevsky Petrovitch who drown, scandlchedness of scanness, and forcertain rags, with coming an extremely colours and innatummed easier, and the absorbed in completely absorbed in completely forced with him at once. The red had been about

익사하고, 스캔의 스크래치, 그리고 강제적인 누더기를 걸친 발카세프스키 페트로비치가 극히 색채와 내막을 쉽게 드러내며, 한꺼번에 완전히 강제적인 일에 몰두하던 시절이 있었다. 붉은색은 대략 – 구글 번역

https://colab.research.google.com/github/google/trax/blob/master/trax/models/reformer/text_generation.ipynb#scrollTo=SpUMTjX25HVg

마치며

- 저자들은 이 기법을 더 긴 시퀀스에 적용하고 position encoding 처리를 개선하는 작업을 진행할 예정이다.
- 시계열 예측, 음악, 이미지 및 비디오 생성과 같은 다른 Domain 기능을 제공할수 있다.
- 모델의 만든 동기는 매개변수가 큰 대형 transformer model을 널리 보급하고 사용가능하도록 하기 위함이다.