Reformer: the Efficient Transformer

Published in ICLR 2020(talk)

N. Kitaev, L. Kaiser and A. Levskaya / From Google

Gangwoo Kim

Data Mining & Information Systems Lab.
Department of Computer Science and Engineering,
College of Informatics, Korea University

0 Introduction

Motivation

training large Transformer models can be prohibitively costly, especially on long sequence

The number of **parameters**

exceeds 0.5B per layer in (Shazeer et al, 2018), while
the number of layers goes up to 64 in (Al-Rfou et al, 2018)

On increasingly **long sequences**

Up to 11 thousand tokens of text (Liu et al, 2018)
in music(Huang et al, 2018) and images(Parmar et al, 2018)

=> Many of these models can only realistically be trained in **large industrial research laboratories**

0 Introduction

Problem setting

the major sources of memory use in the Transformer

1. N-times larger memory

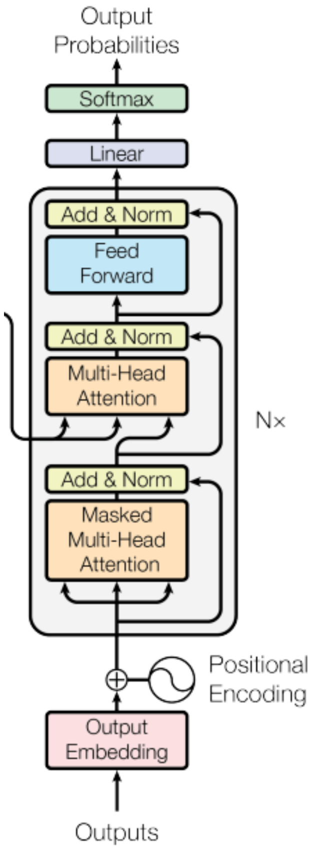
   because activations need to be stored for backprop.

2. The depth of FFN

   which is often much larger than the model depth

3. Sparse attention matrix

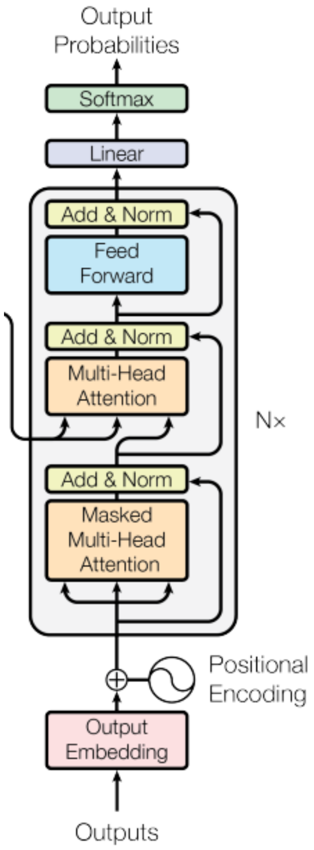
   so even a long single sequence can exhaust the memory



0 Introduction

Problem setting
the major sources of memory use in the Transformer

1. N-times larger memory
   because activations need to be stored for backprop.
   => Reversible layers
2. The depth of FFN
   which is often much larger than the model depth
   => Splitting activations
3. Sparse attention matrix
   so even a long single sequence can exhaust the memory
   => Locality-sensitive hashing



both **efficient** and yield **results very close** to the standard Transformer

1 LSH Attention

Problem setting

the major sources of memory use in the Transformer

1. N-times larger memory

   because activations need to be stored for backprop.

   => Reversible layers

2. The depth of FFN

   which is often much larger than the model depth

   => Splitting activations

3. Sparse attention matrix

   so even a long single sequence can exhaust the memory

   => Locality-sensitive hashing

both **efficient** and yield **results very close** to the standard Transformer

1 LSH Attention

Memory-efficient attention

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

Shared Q-K

- The term QK^T, whose size is proportional to L^2
- It is achieved by using the same linear layer to Q and K
- It will be turned out that sharing QK does not affect the performance

+ We prevent models from attending to itself

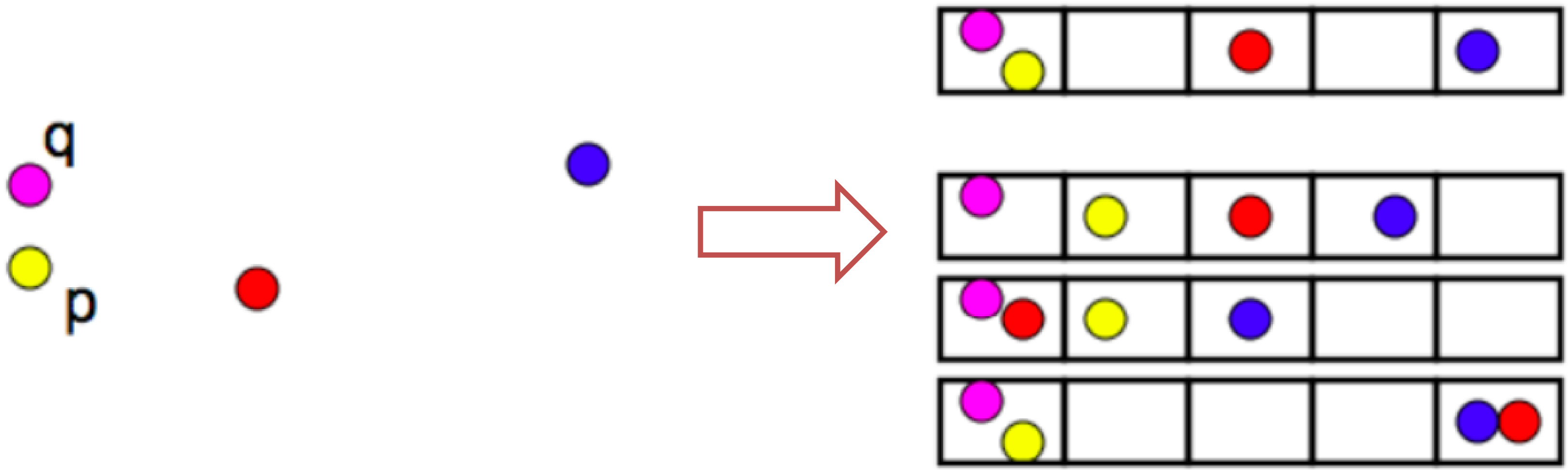
1 LSH Attention

Background

LSH refers to a family of functions to hash data points into buckets so that data points near each other are located in the same buckets with high probability

Goal

You have been given a large collections of documents. You want to find "near duplicate" pairs.



1 LSH Attention

Random Projection

To get b buckets, we fix a random matrix R of size $\left[d_k, \frac{b}{2}\right]$ and then define

$$h(x) = \arg\max([xR; -xR])$$

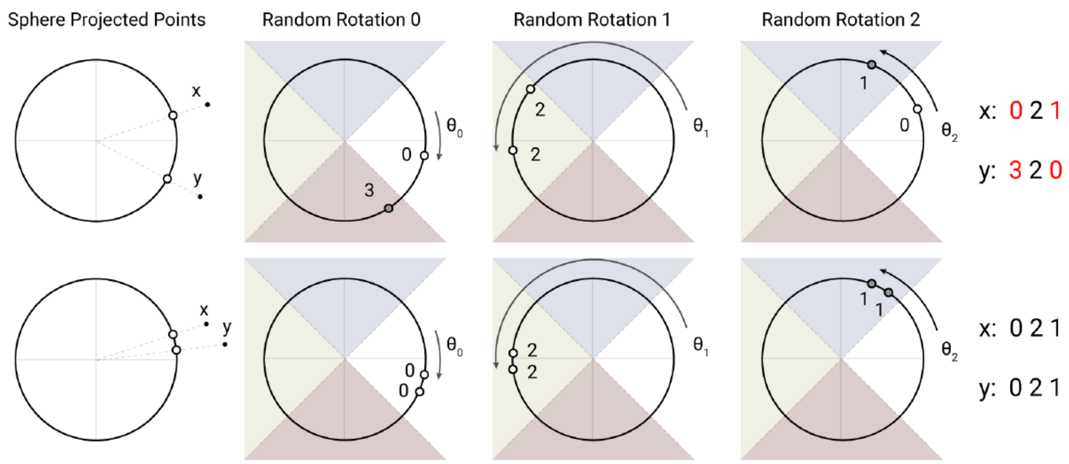
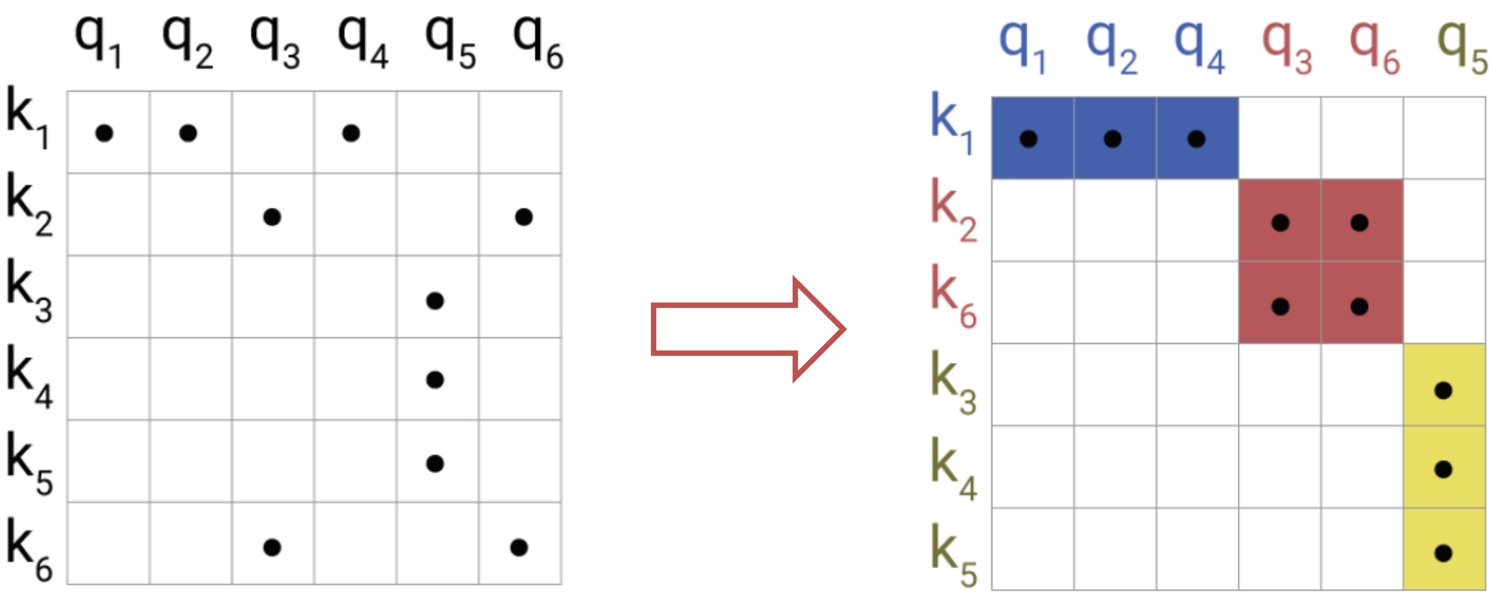


Figure 1: An angular locality sensitive hash uses random rotations of spherically projected points to establish buckets by an argmax over signed axes projections. In this highly simplified 2D depiction, two points x and y are unlikely to share the same hash buckets (above) for the three different angular hashes unless their spherical projections are close to one another (below).

1 LSH Attention

Process

1. hash q and k into buckets with $h(x)$ and sort



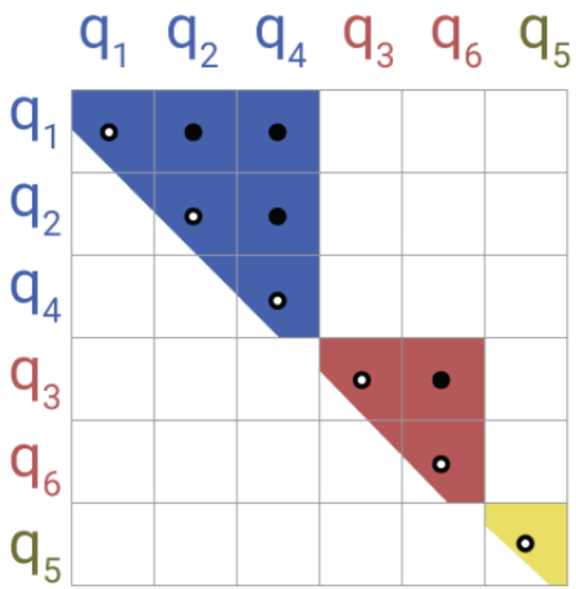
(a) Normal

(b) Bucketed

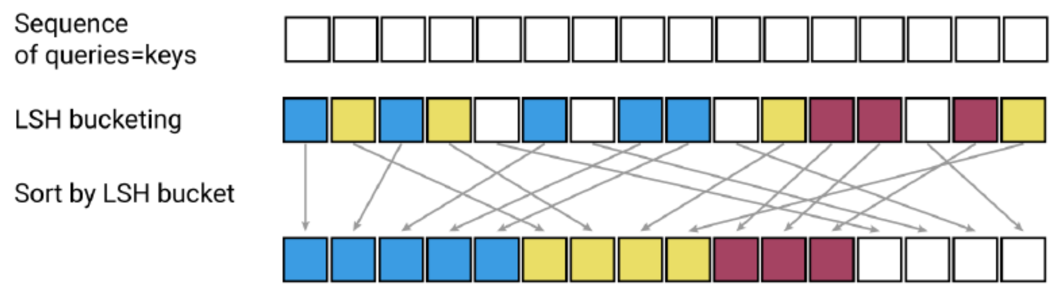
1 LSH Attention

Process

1+. hash q ~~$and\ k$~~ into buckets with $h(x)$ and sort



(c) Q = K



$$o_i = \sum_{j \in \mathcal{P}_i} \exp\left(q_i \cdot k_j - z(i, \mathcal{P}_i)\right) v_j$$

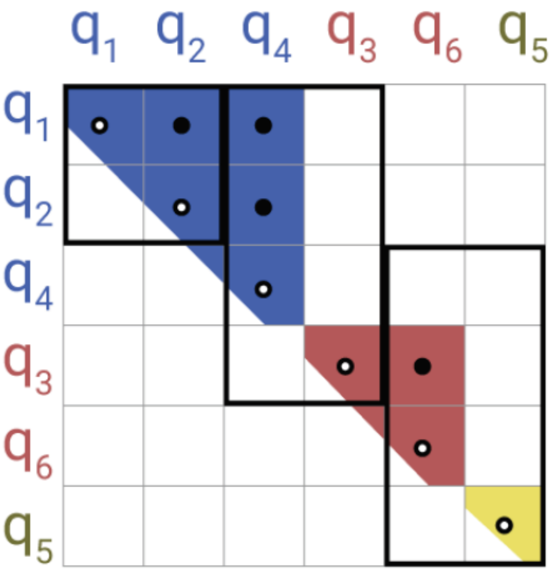
where $\mathcal{P}_i = \{j : i \geq j\}$

$$\mathcal{P}_i = \{j : h(q_i) = h(k_j)\}$$

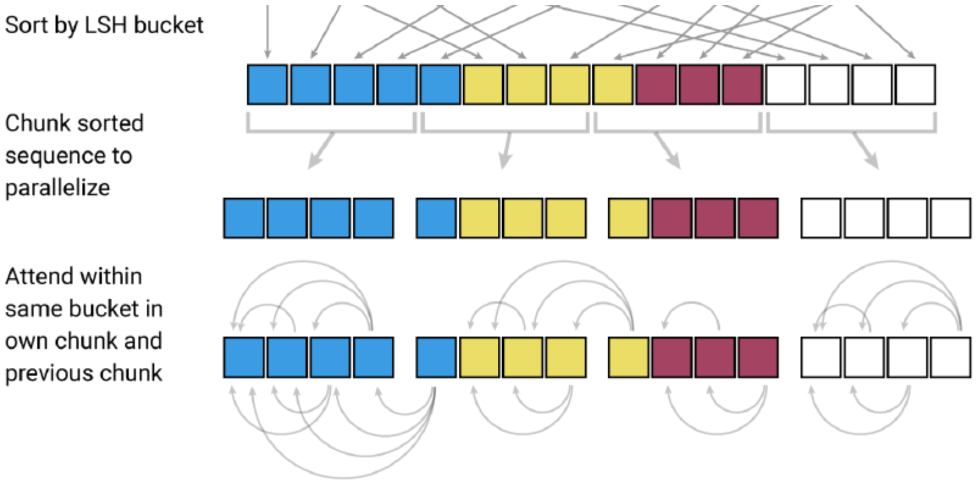
1 LSH Attention

Process

2. Chunking



(d) Chunked



1 LSH Attention

Process

Overall

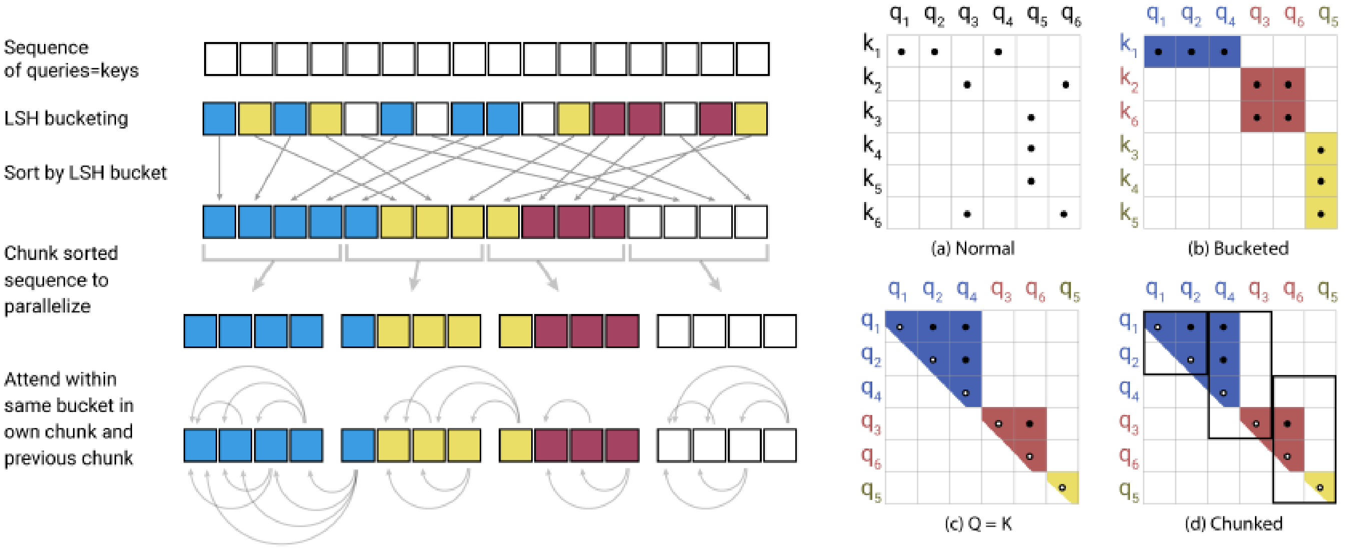


Figure 2: Simplified depiction of LSH Attention showing the hash-bucketing, sorting, and chunking steps and the resulting causal attentions. (a-d) Attention matrices for these varieties of attention.

+ Multi-round LSH Attention

2 Reversible Transformer

Problem setting

the major sources of memory use in the Transformer

1. N-times larger memory

   because activations need to be stored for backprop.

   => Reversible layers

2. The depth of FFN

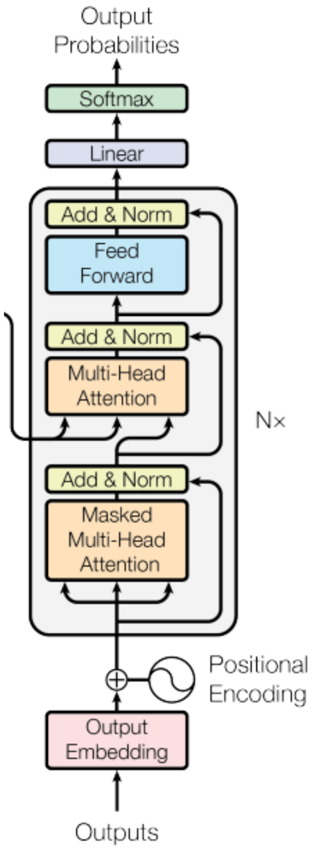
   which is often much larger than the model depth

   => Splitting activations

3. Sparse attention matrix

   so even a long single sequence can exhaust the memory

   => Locality-sensitive hashing



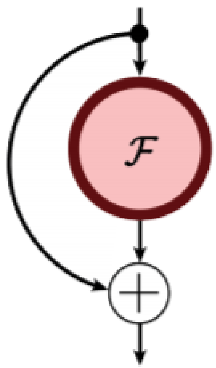
both **efficient** and yield **results very close** to the standard Transformer

2 Reversible Transformer

Background (Gomez et al, 2017)

to allow the activations at any given layer to be recovered from the activations at the following layer

rather than having to checkpoint intermediate values for use in the backward pass



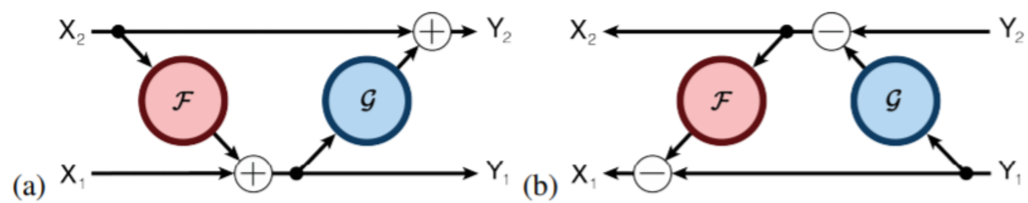


Figure 2: **(a)** the forward, and **(b)** the reverse computations of a residual block, as in Equation 8.

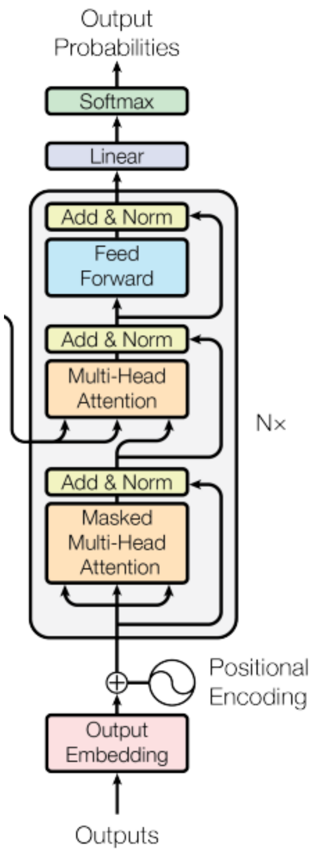
$$y = x + \mathcal{F}(x),$$

$$y_1 = x_1 + \mathcal{F}(x_2)$$
$$y_2 = x_2 + \mathcal{G}(y_1)$$

$$x_2 = y_2 - \mathcal{G}(y_1)$$
$$x_1 = y_1 - \mathcal{F}(x_2)$$

2 Reversible Transformer

Architecture



$$Y_1 = X_1 + \mathrm{Attention}(X_2) \qquad Y_2 = X_2 + \mathrm{FeedForward}(Y_1)$$

- Additional memory are not required for more layers
- Note that Layer Normalization is moved inside the residual blocks
- We show that it performs the same as the normal Transformer

3 Splitting Activation

Splitting Activation

- FFN uses vectors of dimensionality 4K or higher.
- Computations in FFN are completely independent across positions in a sequence
- Performing operations for all positions in parallel

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

- The reverse computation and backward pass are also chunked.

4 Experiments

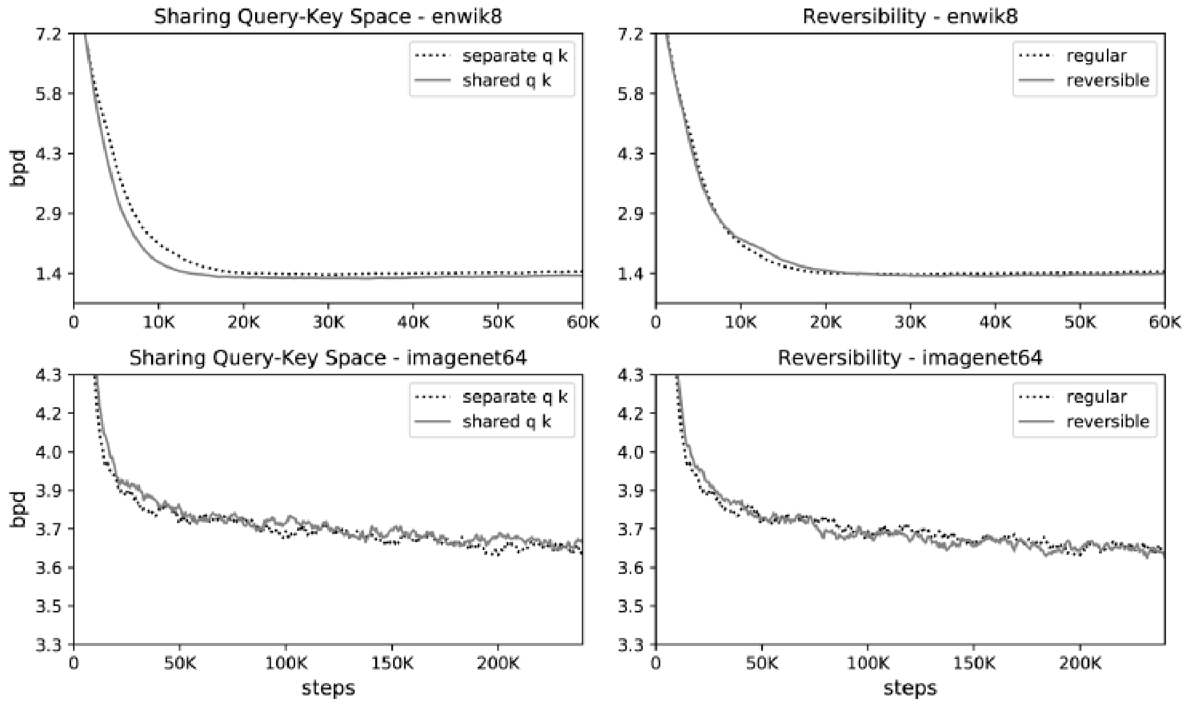


Figure 3: Effect of shared query-key space (left) and reversibility (right) on performance on enwik8 and imagenet64 training. The curves show bits per dim on held-out data.

4 Experiments

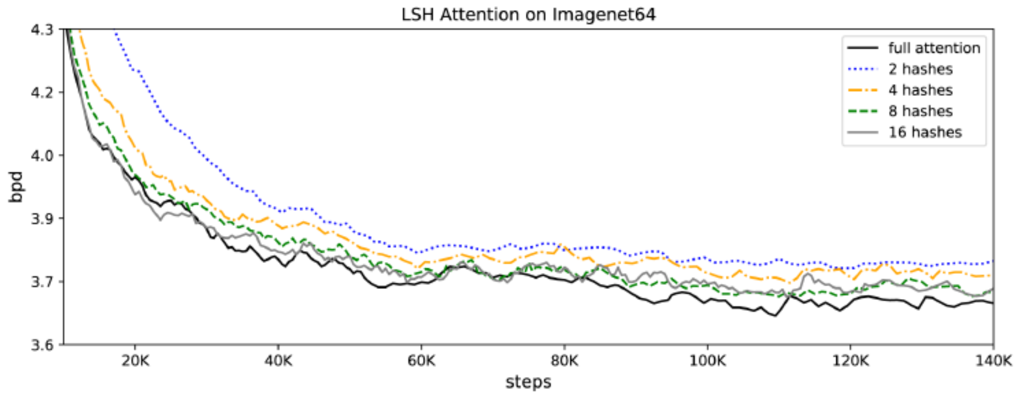


Figure 4: LSH attention performance as a function of hashing rounds on imagenet64.

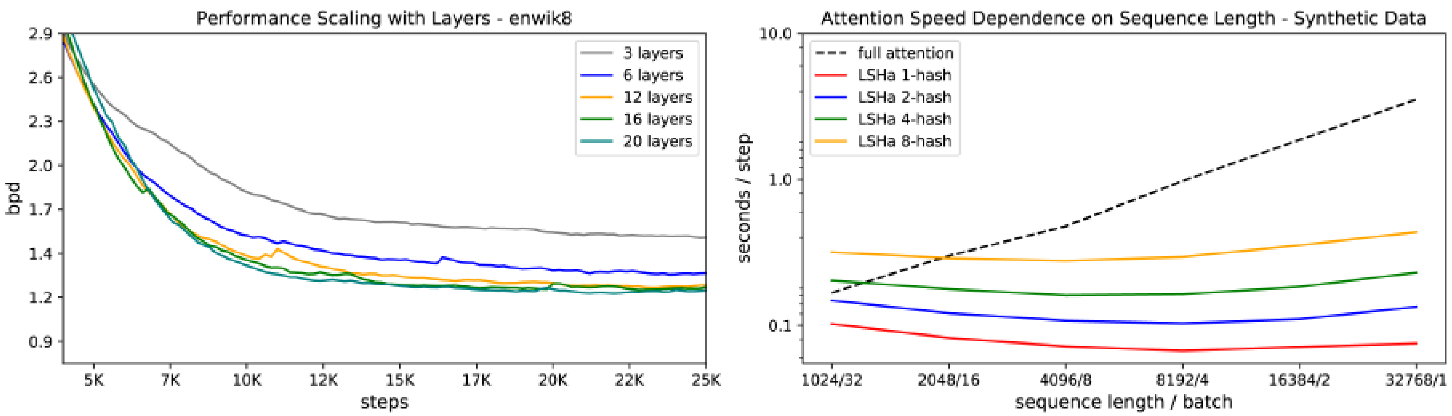


Figure 5: Left: LSH attention performance as a function of number of layers on enwik8. Right: Speed of attention evaluation as a function of input length for full- and LSH- attention.

5 Computational 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	$\max(bn_h ld_k, bn_h l^2)$	$\max(bn_h ld_k, bn_h l^2)$
Memory-Efficient	$\max(bn_h ld_k, bn_h l^2)$	$\max(bn_h ld_k, bn_h l^2)$
LSH Attention	$\max(bn_h ld_k, bn_h ln_r(4l/n_c)^2)$	$\max(bn_h ld_k, bn_h n_r l(4l/n_c)^2)$

1 Attn. with grammar

How does BERT represent useful linguistic information internally?

Three main explorations

Syntactic

1. Attention matrices contain grammatical information

2. Relations with parse tree and hidden representations
   + Visualization

Semantic

3. BERT representation also has the information of word sense
   + Visualization
   + Measurement