Reformer : The Efficient Transformer

Longformer : The Long Document Transformer

Kitaev, Nikita, Lukasz Kaiser, and Anselm Levskaya. "Reformer: The Efficient Transformer." (ICLR 2020)

Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." arXiv preprint arXiv:2004.05150 (2020).

Seungone Kim

Department of Computer Science

Yonsei University

louisdebroglie@yonsei.ac.kr

2021.08.11

연세대학교 YONSEI UNIVERSITY | SOFT COMPUTING LABORATORY

Outline

- 문제 정의
- 관련 연구
- 제안하는 방법
- 실험 결과
- 결론

문제 정의 (Reformer)

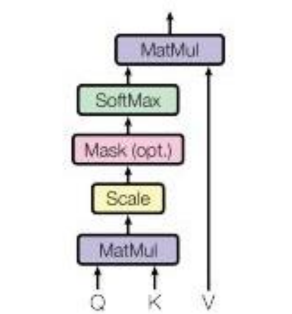
- Many large Transformer models can only realistically be trained in large industrial research laboratories

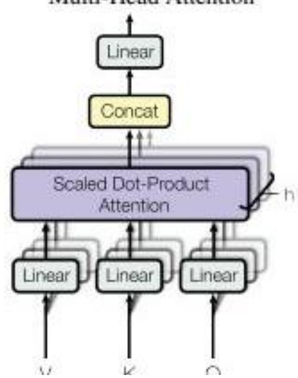
- Models trained with parallelism cannot even be fine-tuned on a single GPU
  - Memory requirements demand multi-accelerator hardware setup even for single training step

- Do large Transformer models fundamentally require huge resources or are they simply inefficient?
  - (Model) 0.5 Billion Parameter (largest reported Transformer layer) = 2GB of memory
  - (Embedding) 64K tokens with embedding size 1024 and batch size 8 = 0.5B floats = 2GB
  - (Data) Whole Corpus for BERT was 17GB
  - 2GB + 2GB + a(data) = 4+a GB(data)
  - If memory use was only per-layer, could easily fit a large Transformer even on sequences of length 64K on a single accelerator

문제 정의 (Reformer)

- Why can't we even fine-tune these models on single machines?
  - Memory in a model with N layers is N-times larger than single layer model due to the fact activations need to be stored for back-propagation
  - Depth of intermediate feed-forward layers is much larger than the depth of attention activations
  - Attention on sequence of length L is $O(L^2)$ in both computational and memory complexity





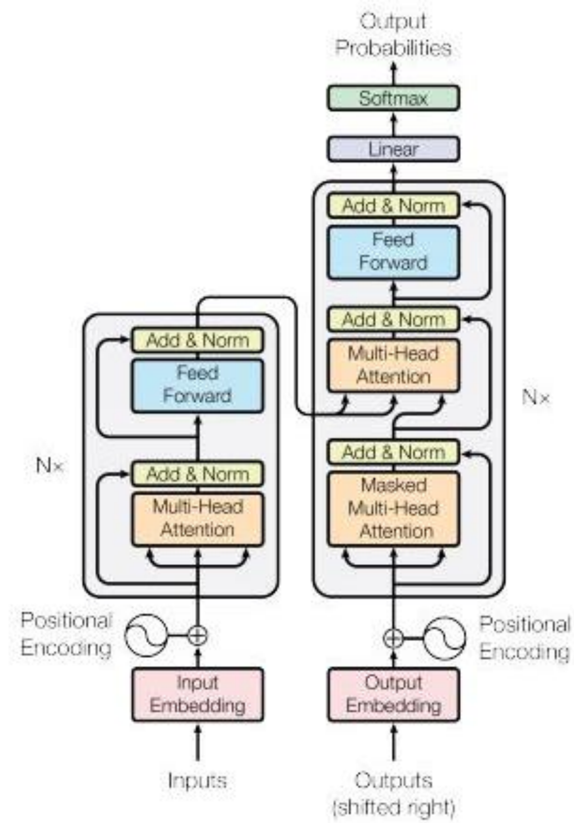


Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

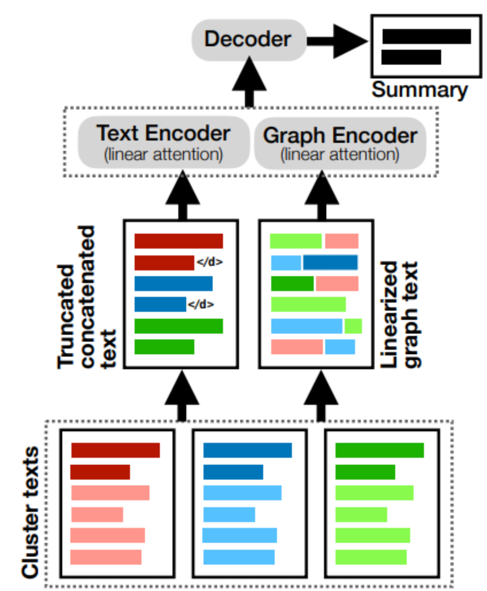
Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

문제 정의 (Longformer)

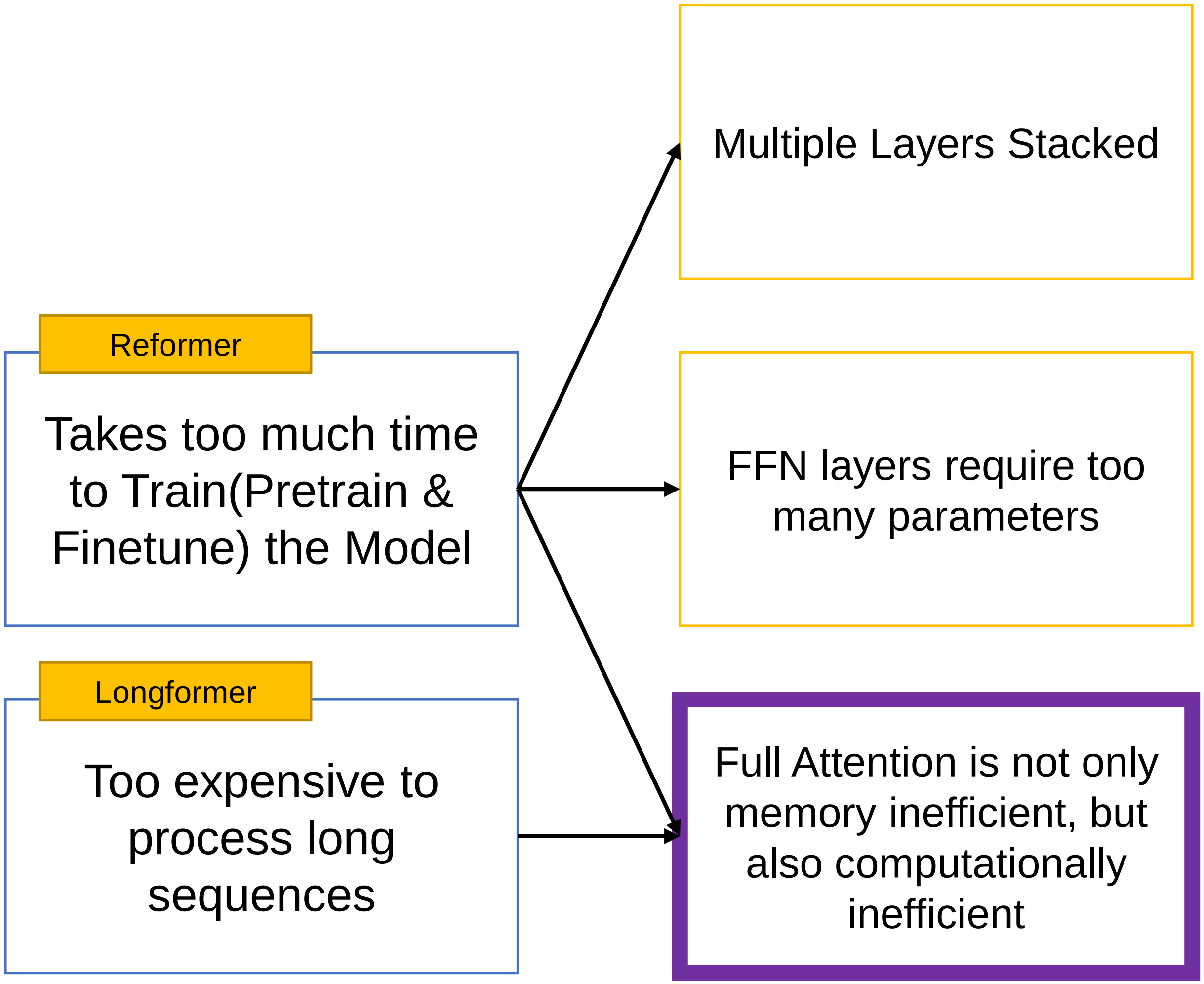
- While powerful, memory and computation requirements of self-attention grow quadratically with sequence length
  - Infeasible (or very expensive) to process long sequences

- Limitation to length is a critical disadvantage to tasks with long documents
  - Classification, Question Answering, Coreference Resolution, Summarization
  - Existing approaches partition or shorten the long context into smaller sequences that fall within the typical 512 token limit of BERT-style pretrained models
  - *e.g. Pasunuru, Ramakanth, et al. "Efficiently Summarizing Text and Graph Encodings of Multi-Document Clusters." (NAACL 2021)*

```python
from transformers import BertTokenizerFast

tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')

seq1 = 'This is a long uninteresting text'
seq2 = 'What could be a second sequence to the uninteresting text'

print(len(tokenizer.tokenize(seq1)))
print(len(tokenizer.tokenize(seq2)))

print(tokenizer(seq1, seq2))

print(tokenizer(seq1, seq2, truncation= True, max_length = 15))
```

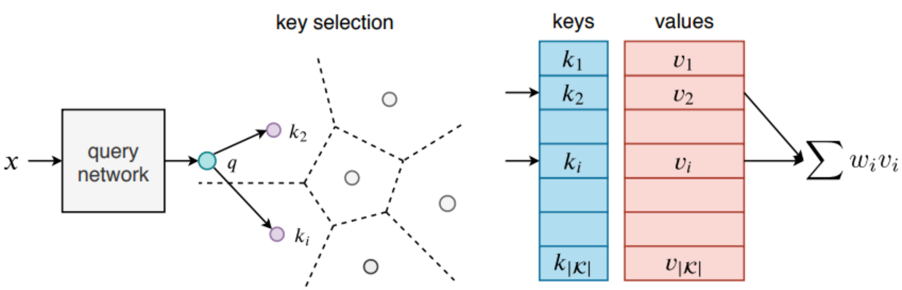


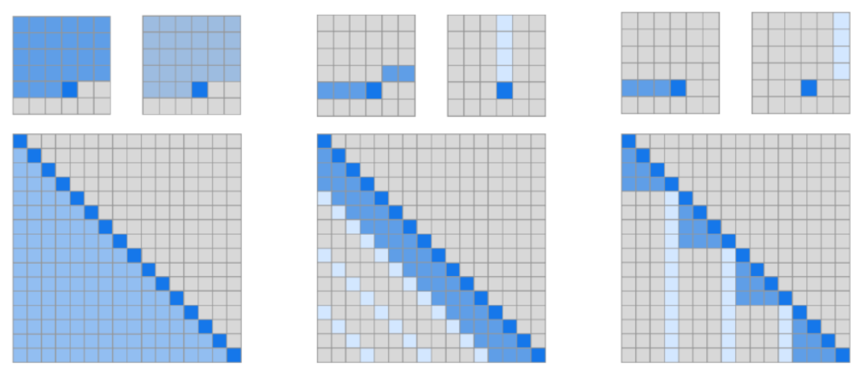
문제 정의 (총 정리)


Reformer
Takes too much time to Train(Pretrain & Finetune) the Model
Longformer
Too expensive to process long sequences
Multiple Layers Stacked
FFN layers require too many parameters
Full Attention is not only memory inefficient, but also computationally inefficient

관련 연구 (Reformer)

- Methods to reduce the memory and computational requirements of Transformers
  - *Child et al. "Generating long sequences with sparse transformers."* proposes $O(n\sqrt{n})$ complexity method using factorized sparse representation of attention with dilated sliding window
  - *Lample, Guillaume, et al. "Large memory layers with product keys." (NEURIPS 2019)* proposes increasing key space to reduce memory requirements in feed-forward layers
- LSH has not been directly applied to Transformers before Reformer
  - *Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." arXiv preprint arXiv:1410.3916 (2014).* uses external memory with neural networks
  - LSH could be used to perform memory lookup; querying memory locations that are useful
  - Previous methods include LSH and random kd-trees only for lookups in external memory



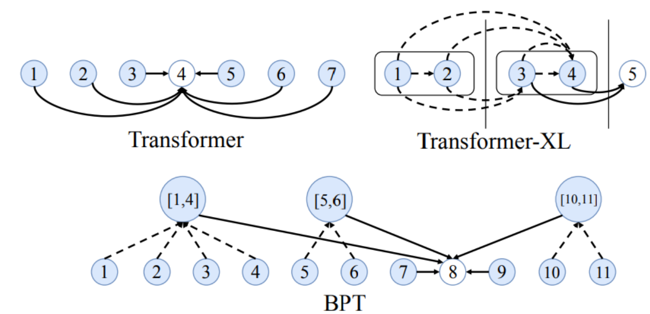


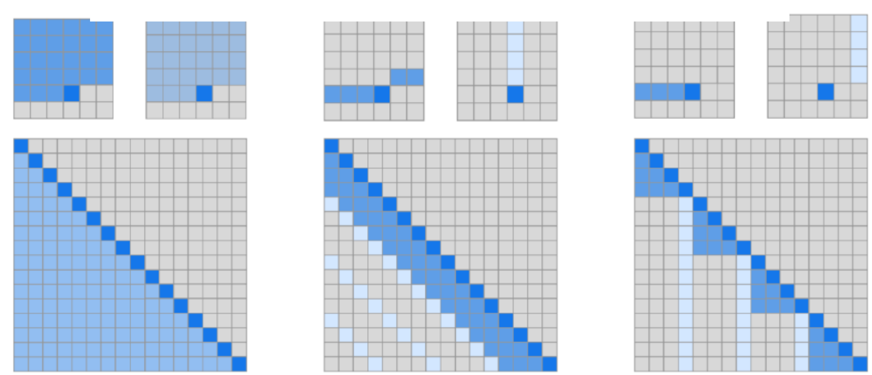
(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

관련 연구 (Longformer)

- Left-to-Right(LR) Approach
  - Process document in chunks moving from left-to-right (recursively!)
  - While successful in AR Language Modeling, unsuitable for tasks from bidirectional context
- Sparse Attention Pattern
  - Avoids computing the full quadratic attention matrix multiplication
  - *Child et al. "Generating long sequences with sparse transformers."* proposes $O(n\sqrt{n})$ complexity method using factorized sparse representation of attention with dilated sliding window
  - Other sparse attention approaches did not explore the pretrain – finetune setting

Model	attention matrix	char-LM	other tasks	pretrain
Transformer-XL (2019)	ltr	yes	no	no
Adaptive Span (2019)	ltr	yes	no	no
Compressive (2020)	ltr	yes	no	no
Reformer (2020)	sparse	yes	no	no
Sparse (2019)	sparse	yes	no	no
Routing (2020)	sparse	yes	no	no
BP-Transformer (2019)	sparse	yes	MT	no
Blockwise (2019)	sparse	no	QA	yes
Our Longformer	sparse	yes	multiple	yes

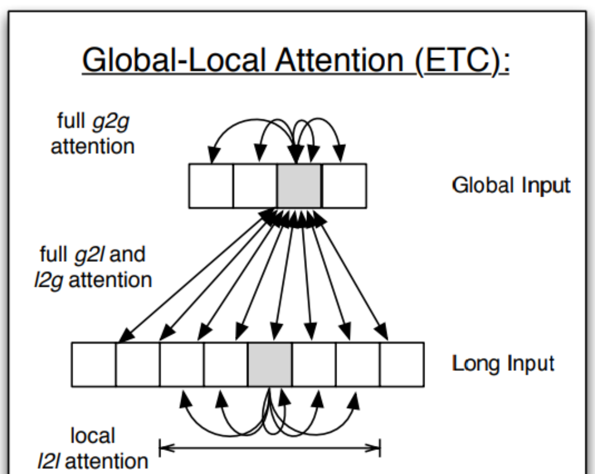


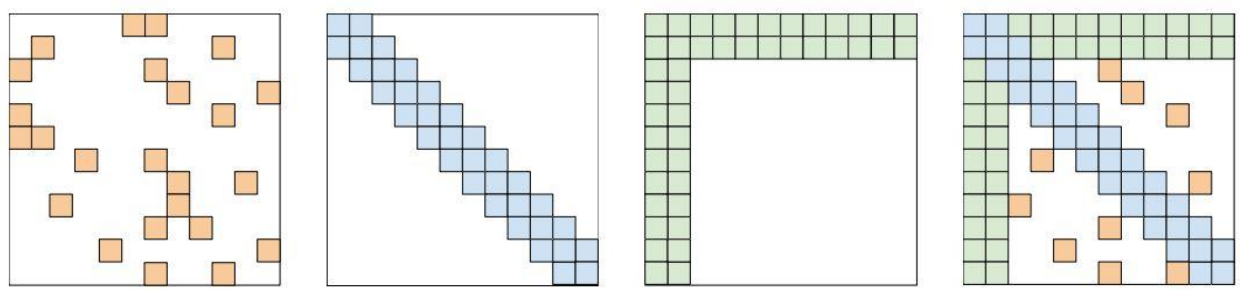


(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

관련 연구 (Longformer)

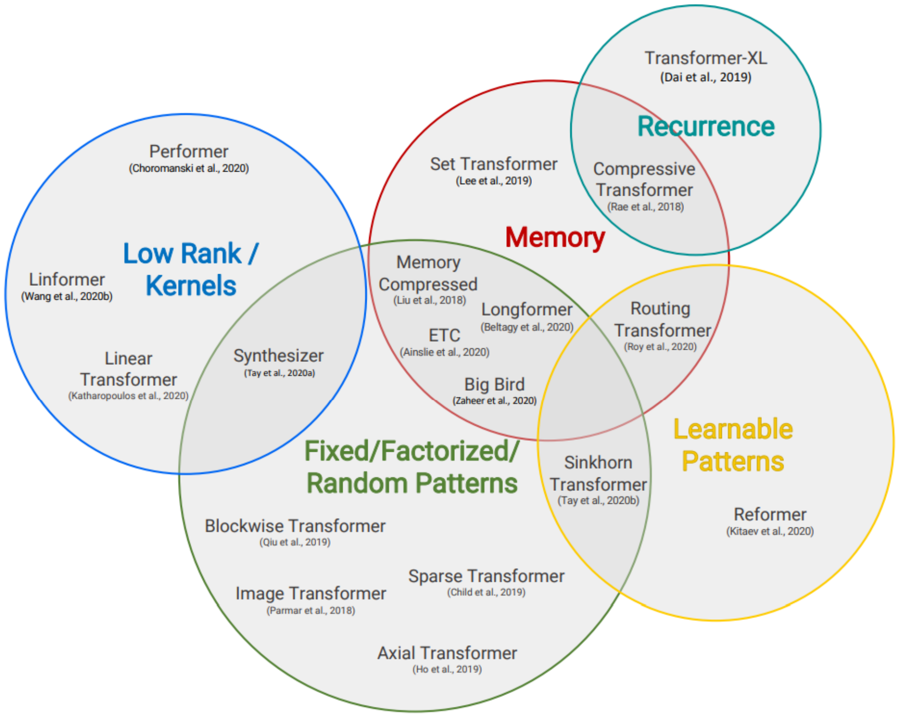
- Task specific Models for Long Documents
  - *Qizhe Xie et al., “Unsupervised data augmentation for consistency training”* truncates document for classification task
  - *Mandar Joshi et al., “BERT for coreference resolution: Baselines and analysis” (EMNLP-IJCNLP 2019)* chunks document into chunks of length 512, processes each chunk separately, then combines the activations with a task specific model
  - QA Tasks uses two stage model where the first stage retrieves relevant documents that are passed onto the second stage for answer extraction
  - All of these approaches suffer from information loss
- Local + Global Attention in Transformers
  - Contemporaneous works (published on arXiv after Longformer)
  - *Ainslie et al. "ETC: Encoding Long and Structured Inputs in Transformers." (EMNLP 2020)*
  - *Zaheer, Manzil, et al. "Big Bird: Transformers for Longer Sequences." (NeurIPS. 2020)*



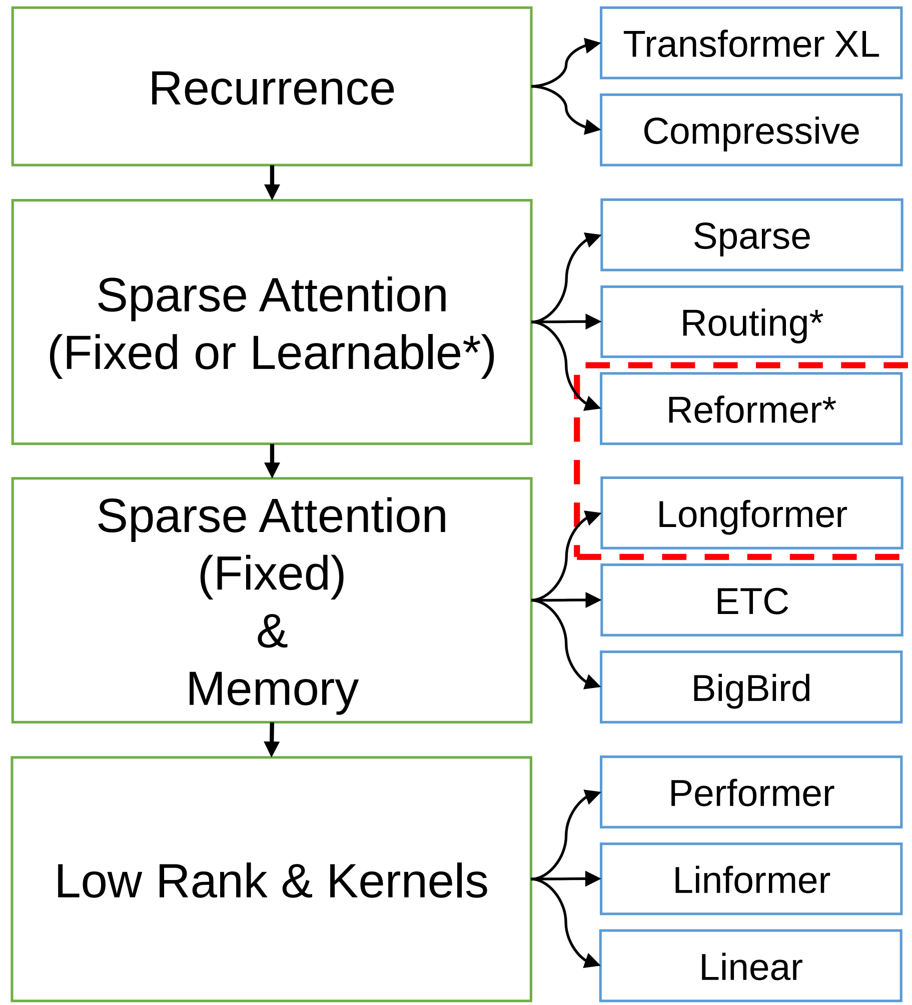


(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

관련 연구(총 정리)



Model / Paper	Complexity	Decode	Class
Memory Compressed† (Liu et al., 2018)	$\mathcal{O}(n_c^2)$	✓	FP+M
Image Transformer† (Parmar et al., 2018)	$\mathcal{O}(n.m)$	✓	FP
Set Transformer† (Lee et al., 2019)	$\mathcal{O}(nk)$	✗	M
Transformer-XL† (Dai et al., 2019)	$\mathcal{O}(n^2)$	✓	RC
Sparse Transformer (Child et al., 2019)	$\mathcal{O}(n\sqrt{n})$	✓	FP
Reformer† (Kitaev et al., 2020)	$\mathcal{O}(n \log n)$	✓	LP
Routing Transformer (Roy et al., 2020)	$\mathcal{O}(n \log n)$	✓	LP
Axial Transformer (Ho et al., 2019)	$\mathcal{O}(n\sqrt{n})$	✓	FP
Compressive Transformer† (Rae et al., 2020)	$\mathcal{O}(n^2)$	✓	RC
Sinkhorn Transformer† (Tay et al., 2020b)	$\mathcal{O}(b^2)$	✓	LP
Longformer (Beltagy et al., 2020)	$\mathcal{O}(n(k+m))$	✓	FP+M
ETC (Ainslie et al., 2020)	$\mathcal{O}(n_g^2 + nn_g)$	✗	FP+M
Synthesizer (Tay et al., 2020a)	$\mathcal{O}(n^2)$	✓	LR+LP
Performer (Choromanski et al., 2020)	$\mathcal{O}(n)$	✓	KR
Linformer (Wang et al., 2020b)	$\mathcal{O}(n)$	✗	LR
Linear Transformers† (Katharopoulos et al., 2020)	$\mathcal{O}(n)$	✓	KR
Big Bird (Zaheer et al., 2020)	$\mathcal{O}(n)$	✗	FP+M



제안하는 방법 (Reformer)

- Reversible Layers
  - Gomez, Aidan N., et al. "The reversible residual network: Backpropagation without storing activations." *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017.
  - Enable storing only a copy of activations in the whole model, discarding N (layer) factor
  - ALBERT referenced the paper above

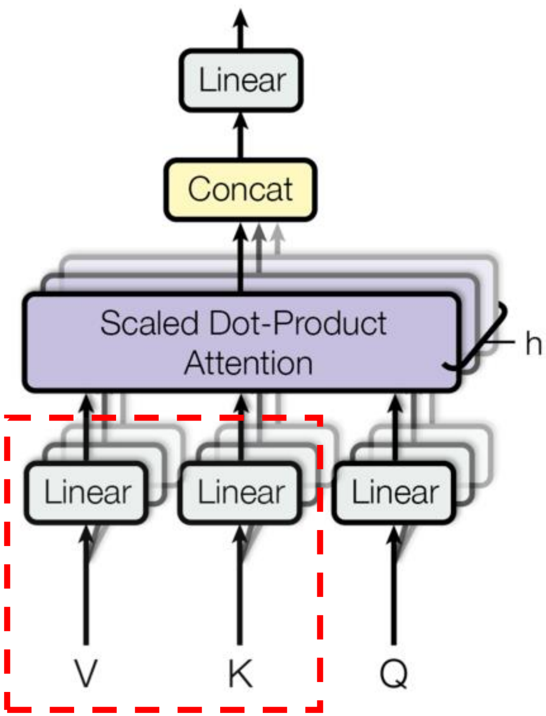
- Splitting Activations
  - Processing feed-forward layers in chunks to save memory
  - Discards dff (intermediate feed-forward depth) factor

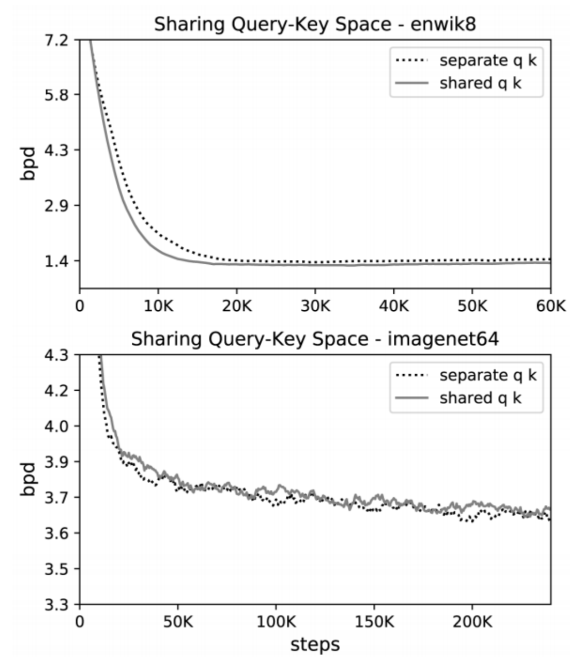
- Approximate attention computation based on locality-sensitive hashing
  - Replaces $O(L^2)$ factor with $O(L \log L)$, therefore allowing to operate on longer sequences

- Locality-sensitive Hashing in Attention is the major change
  - Influences the training dynamics

제안하는 방법 (Reformer)

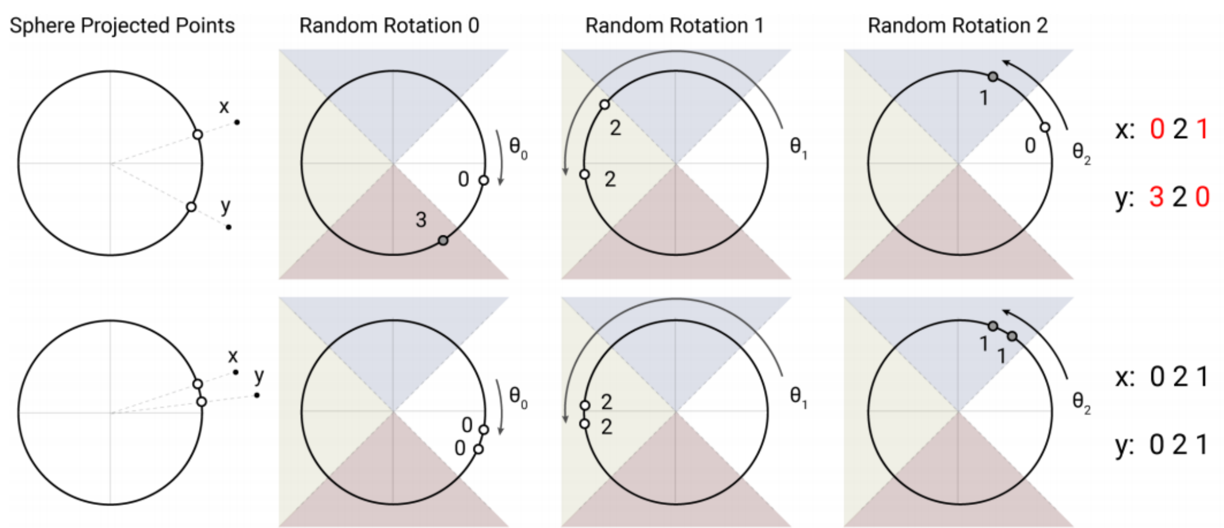
- Locality Sensitive Hashing Attention
  - Set queries(Q) and keys(K) to be identical by using same linear layer ($shape \ = \ [\,bs, sl, d_k]$)
  - Sharing QK does not affect the performance of Transformer
  - Instead of QK^T itself, we are interested in $\mathrm{soft}max(QK^T)$
  - Focus on keys in K that are closest to q_i (consider small subset of 32 or 64 closest keys)
  - Finding nearest neighbors quickly in high-dimensional spaces by Locality-Sensitive Hashing(LSH)





제안하는 방법 (Reformer)

- Locality Sensitive Hashing Attention
  - *Andoni, Alexandr, et al. "Practical and optimal LSH for angular distance." Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1. 2015.*
  - Assign vector x to a hash $h(x)$
  - Nearby vectors get same hash with high probability and distant ones do not
  - Achieve by employing random projection
  - Fix a random matrix R of $size\ = [\ d_k, \frac{b}{2}\ ]$
  - Define $h(x) = argmax(\ [xR; -xR\ ]\ )$
  - Using LSH algorithm, the goal is to put similar vectors into the same bucket

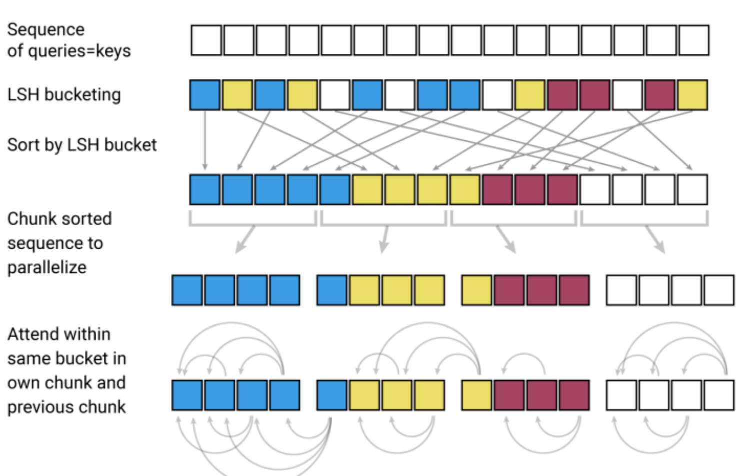


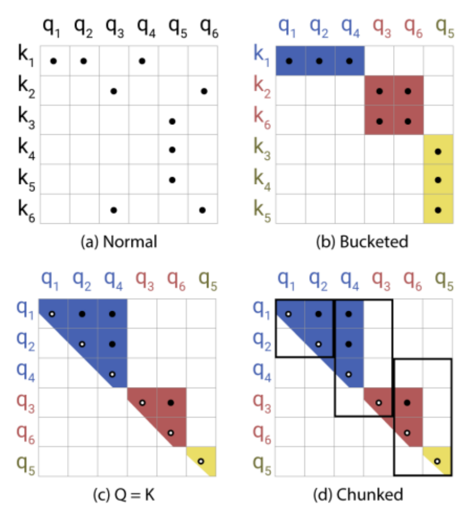
제안하는 방법 (Reformer)

- Locality Sensitive Hashing Attention
  - P_i is a set that the query at position i attends to
  - z is a partition function (i.e. normalizing term in the softmax)
  - m is a function where $m(j, \mathcal{P}_i) = \inf\ (if\ \ j \notin \mathcal{P}_i)\ \ and\ 0\ (otherwise)$

  - $Normal\ Attention\ :\ o_i = \sum_{j \in \mathrm{P}_i} \exp(q_i \cdot k_j - z(i, \mathcal{P}_i))\, v_j \quad where\ \mathcal{P}_i = \{j : i \geq j\}$
  - $Normal\ Attention^*:\ o_i = \sum_{j \in \widetilde{\mathrm{P}_i}} \exp(q_i \cdot k_j - m(j, \mathcal{P}_i) - z(i, \mathcal{P}_i))\, v_j \quad where\ \widetilde{\mathcal{P}_i} = \{0, 1, \dots, l\} \supseteq \mathcal{P}_i$
  - $LSH\ Attention : same\ with\ above,\ \ but\ \mathcal{P}_i = \{j : h(q_i) = h(k_j)\}$

  - $(a \rightarrow b)$ Applying LSH, Queries and Keys have been sorted according to their hash bucket
  - $(b \rightarrow c)$ To overcome uneven size, ensure $h(k_j) = h(q_j)$ by setting $k_j = \frac{q_j}{\|q_j\|}$
  - $(c \rightarrow d)$ Follow a batching approach where chunks of m consecutive queries attend to each other





제안하는 방법 (Reformer)

- Locality Sensitive Hashing Attention
  - The average bucket size is $\frac{l}{n_{buckets}}$ $where\ l\ is\ the\ sequence\ length$
  - There is small probability that similar items nevertheless fall in different buckets
  - Reduce this probability by doing multiple rounds of hashing with distinct hashing function
  - $\mathcal{P}_i = \bigcup_{r=1}^{n_{rounds}} \mathcal{P}_i^{(r)} \quad where \quad \mathcal{P}_i^{(r)} = \{j : h^{(r)}(q_i) = h^{(r)}(q_j)\}$
  - Replaces $O(L^2)$ factor with $O(L \log L)$, therefore allowing to operate on longer sequences

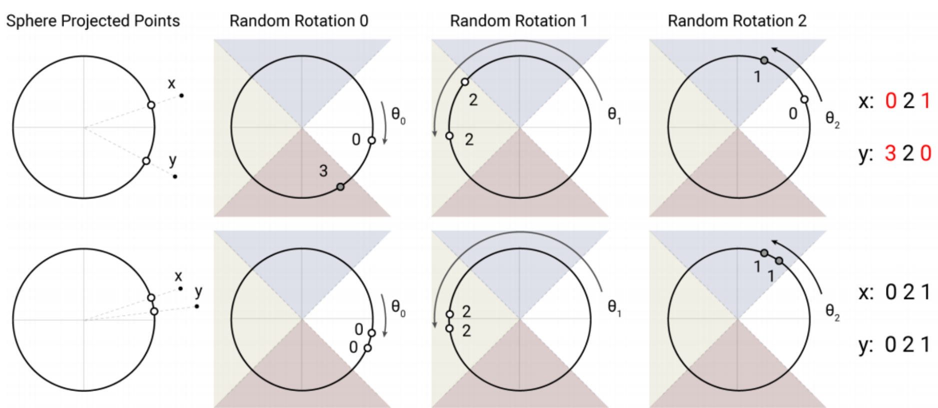


Table 1: Memory and time complexity of attention variants. We write l for length, b for batch size, n_h for the number of heads, n_c for the number of LSH chunks, n_r for the number of hash repetitions.

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	$\max(bn_hld_k, bn_hln_r(4l/n_c)^2)$	$\max(bn_hld_k, bn_hn_rl(4l/n_c)^2)$

Table 2: Accuracies on the duplication task of a 1-layer Transformer model with full attention and with locality-sensitive hashing attention using different number of parallel hashes.

Train \ Eval	Full Attention	LSH-8	LSH-4	LSH-2	LSH-1
Full Attention	100%	94.8%	92.5%	76.9%	52.5%
LSH-4	0.8%	100%	99.9%	99.4%	91.9%
LSH-2	0.8%	100%	99.9%	98.1%	86.8%
LSH-1	0.8%	99.9%	99.6%	94.8%	77.9%

제안하는 방법 (Reformer)

- Reversible Transformer
  - Complexity of attention was reduced from square in length to linear
  - But memory use of whole model is still large…
  - Dealing with the n_l *and* d_{ff} problem, we can solve the problem

  - Apply the RevNet idea to Transformers by combining the attention and feed-forward layers inside revnet block
  - $Y_1 = X_1 + Attention(X_2),\ Y_2 = X_2 + FeedForward(Y_1)$
  - Enable storing only a copy of activations in the whole model, discarding N (layer) factor

  - Split the Feed-forward layer's computation into c chunks
  - $Y_2 = [Y_2^{(1)}; \ldots; Y_2^{(c)}] = [X_2^{(1)} + FeedForward(Y_1^{(1)}); \ldots; X_2^{(c)} + FeedForward(Y_1^{(c)})]$
  - Discards dff (intermediate feed-forward depth) factor

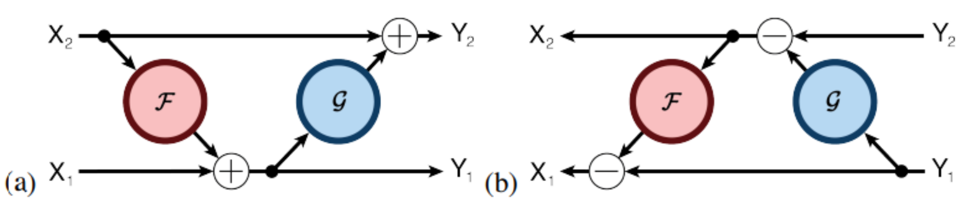


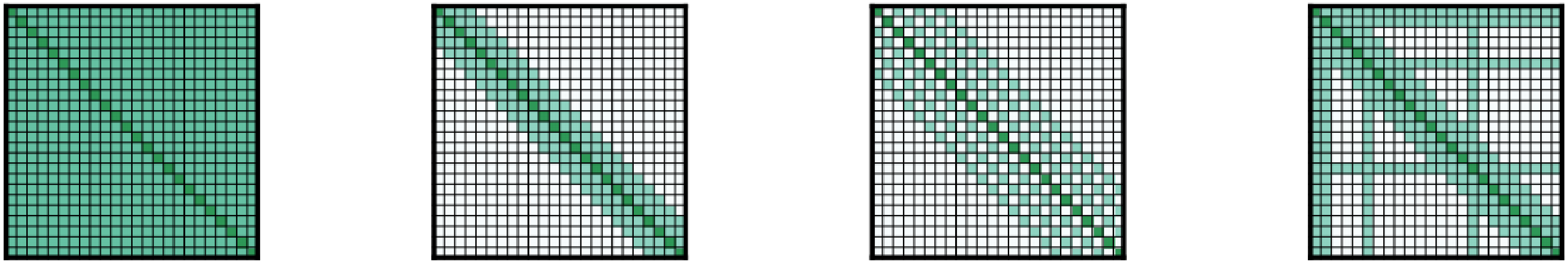
Table 3: Memory and time complexity of Transformer variants. We write d_{model} and d_{ff} for model depth and assume $d_{ff} \geq d_{model}$; b stands for batch size, l for length, n_l for the number of layers. We assume $n_c = l/32$ so $4l/n_c = 128$ and we write $c = 128^2$.

Model Type	Memory Complexity	Time Complexity
Transformer	$\max(bld_{ff}, bn_hl^2)n_l$	$(bld_{ff} + bn_hl^2)n_l$
Reversible Transformer	$\max(bld_{ff}, bn_hl^2)$	$(bn_hld_{ff} + bn_hl^2)n_l$
Chunked Reversible Transformer	$\max(bld_{model}, bn_hl^2)$	$(bn_hld_{ff} + bn_hl^2)n_l$
LSH Transformer	$\max(bld_{ff}, bn_hln_rc)n_l$	$(bld_{ff} + bn_hn_rlc)n_l$
Reformer	$\max(bld_{model}, bn_hln_rc)$	$(bld_{ff} + bn_hn_rlc)n_l$

제안하는 방법 (Longformer)

- Combination of 2 self-attention mechanisms
  - Windowed local-context self-attention
  - End task motivated global attention that encodes inductive bias about the task

- Sliding Window Attention
  - Employ a fixed-size window attention surrounding each token
  - Computation complexity is $O(n \times w)$ $where\ w\ is\ a\ fixed\ window\ size$
  - Use small window sizes for lower layers and increase window sizes in higher layers
  - To increase the receptive field, the sliding window can be dilated
  - Applying dilated sliding windows, Computation complexity is $O(n \times d)$ $where\ d\ is\ dilation\ size$
  - Setting with different dilation configurations per head improves performance
  - Some heads without dilation to focus on local context, others to focus on longer content

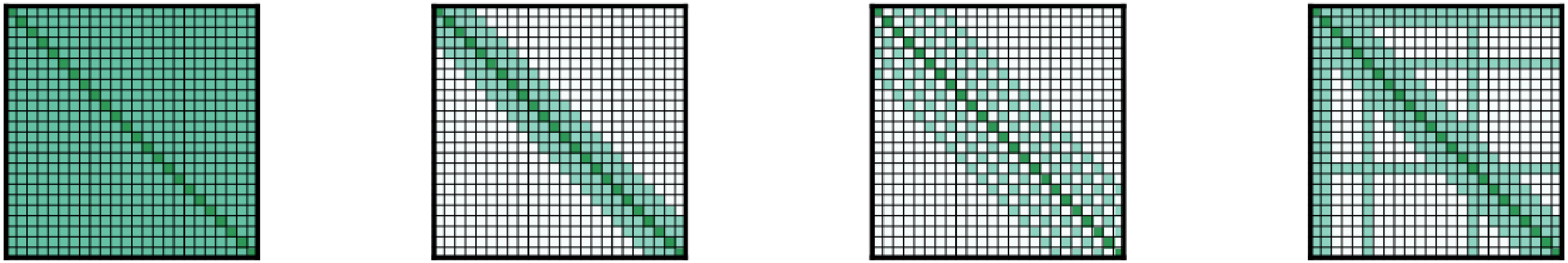


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

제안하는 방법 (Longformer)

- Combination of 2 self-attention mechanisms
  - Windowed local-context self-attention
  - End task motivated global attention that encodes inductive bias about the task

- Global Attention
  - Add global Attention on few pre-selected input locations
  - Since number of such token is small, the combined attention is still $O(n)$
  - Adding global attention is task specific, it is a easy way to add inductive bias to model's attention



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

실험 결과 (Reformer)

- BLEU score on newstest 2014 for WMT English-German(EnDe)

Table 4: BLEU scores on newstest2014 for WMT English-German (EnDe). We additionally report detokenized BLEU scores as computed by sacreBLEU (Post, 2018).

Model	BLEU	*sacreBLEU* *Uncased*[3]	*sacreBLEU* *Cased*[4]
Vaswani et al. (2017), base model	27.3		
Vaswani et al. (2017), big	28.4		
Ott et al. (2018), big	29.3		
Reversible Transformer (base, 100K steps)	27.6	*27.4*	*26.9*
Reversible Transformer (base, 500K steps, no weight sharing)	28.0	*27.9*	*27.4*
Reversible Transformer (big, 300K steps, no weight sharing)	29.1	*28.9*	*28.4*

실험 결과 (Longformer)

- Effect of different window size & Dilation size

Model	Dev BPC
Decreasing w (from 512 to 32)	1.24
Fixed w (= 230)	1.23
Increasing w (from 32 to 512)	**1.21**
No Dilation	1.21
Dilation on 2 heads	**1.20**

Table 4: Top: changing window size across layers. Bottom: with/without dilation (@ 150K steps on phase1)

- Big problem is position embeddings
  - Initialize by copying 512 position embeddings from RoBERTa multiple times
  - Continue Pretraining from RoBERTa released checkpoints

Model	base	large
RoBERTa (seqlen: 512)	1.846	1.496
Longformer (seqlen: 4,096)	10.299	8.738
+ copy position embeddings	1.957	1.597
+ 2K gradient updates	1.753	1.414
+ 65K gradient updates	1.705	1.358
Longformer (train extra pos. embed. only)	1.850	1.504

Table 5: MLM BPC for RoBERTa and various pretrained Longformer configurations.

실험 결과 (Longformer)

```python
def create_long_model(save_model_to, attention_window, max_pos):
    model = RobertaForMaskedLM.from_pretrained('roberta-base')
    tokenizer = RobertaTokenizerFast.from_pretrained('roberta-base', model_max_length=max_pos)
    config = model.config

    # extend position embeddings
    tokenizer.model_max_length = max_pos
    tokenizer.init_kwargs['model_max_length'] = max_pos
    current_max_pos, embed_size = model.roberta.embeddings.position_embeddings.weight.shape
    max_pos += 2  # NOTE: RoBERTa has positions 0,1 reserved, so embedding size is max position + 2
    config.max_position_embeddings = max_pos
    assert max_pos > current_max_pos
    # allocate a larger position embedding matrix
    new_pos_embed = model.roberta.embeddings.position_embeddings.weight.new_empty(max_pos, embed_size)
    # copy position embeddings over and over to initialize the new position embeddings
    k = 2
    step = current_max_pos - 2
    while k < max_pos - 1:
        new_pos_embed[k:(k + step)] = model.roberta.embeddings.position_embeddings.weight[2:]
        k += step
    model.roberta.embeddings.position_embeddings.weight.data = new_pos_embed
    model.roberta.embeddings.position_ids.data = torch.tensor([i for i in range(max_pos)]).reshape(1, max_pos)

    # replace the `modeling_bert.BertSelfAttention` object with `LongformerSelfAttention`
    config.attention_window = [attention_window] * config.num_hidden_layers
    for i, layer in enumerate(model.roberta.encoder.layer):
        longformer_self_attn = LongformerSelfAttention(config, layer_id=i)
        longformer_self_attn.query = layer.attention.self.query
        longformer_self_attn.key = layer.attention.self.key
        longformer_self_attn.value = layer.attention.self.value

        longformer_self_attn.query_global = copy.deepcopy(layer.attention.self.query)
        longformer_self_attn.key_global = copy.deepcopy(layer.attention.self.key)
        longformer_self_attn.value_global = copy.deepcopy(layer.attention.self.value)

        layer.attention.self = longformer_self_attn

    logger.info(f'saving model to {save_model_to}')
    model.save_pretrained(save_model_to)
    tokenizer.save_pretrained(save_model_to)
    return model, tokenizer
```

실험 결과(Longformer)

- Applied to Question Answering, Coreference Resolution, Classification
  - (QA) WikiHop, TriviaQA, HotpotQA
  - (Coreference Resolution) OntoNotes
  - (Classification) IMDB, Hyperpartisan

	QA			Coref.	Classification	
Model	WikiHop	TriviaQA	HotpotQA	OntoNotes	IMDB	Hyperpartisan
RoBERTa-base	72.4	74.3	63.5	78.4	95.3	87.4
Longformer-base	**75.0**	**75.2**	**64.4**	**78.6**	**95.7**	**94.8**

Wordpieces	WH	TQA	HQA	ON	IMDB	HY
avg.	1,535	6,589	1,316	506	300	705
95th pctl.	3,627	17,126	1,889	1,147	705	1,975

Table 6: Average and 95th percentile of context length of datasets in wordpieces. WH: WikiHop, TQA: TriviaQA, HQA: HotpotQA, ON: OntoNotes, HY: Hyperpartisan news

Model	WikiHop	TriviaQA	HotpotQA
Current* SOTA	78.3	73.3	**74.2**
Longformer-large	**81.9**	**77.3**	73.2

Table 8: Leaderboard results of Longformer-large at time of submission (May 2020). All numbers are F1 scores.

Model	ans.	supp.	joint
TAP 2 (ensemble) (Glaß et al., 2019)	79.8	86.7	70.7
SAE (Tu et al., 2019)	79.6	86.7	71.4
Quark (dev) (Groeneveld et al., 2020)	81.2	87.0	72.3
C2F Reader (Shao et al., 2020)	81.2	87.6	72.8
Longformer-large	81.3	88.3	73.2
ETC-large† (Ainslie et al., 2020)	81.2	89.1	73.6
GSAN-large†	81.6	88.7	73.9
HGN-large (Fang et al., 2020)	82.2	88.5	74.2

Table 9: HotpotQA results in distractor setting test set.

실험 결과(Longformer)

- Applied to Summarization
  - Encoder-Decoder Transformer models have achieved strong result on Summarization
  - Longformer-Encoder-Decoder(LED), encoder uses local+global attention pattern
  - Initialize LED parameters from BART

	R-1	R-2	R-L
Discourse-aware (2018)	35.80	11.05	31.80
Extr-Abst-TLM (2020)	41.62	14.69	38.03
Dancer (2020)	42.70	16.54	38.44
Pegasus (2020)	44.21	16.95	38.83
LED-large (seqlen: 4,096) (ours)	44.40	17.94	39.76
BigBird (seqlen: 4,096) (2020)	**46.63**	19.02	41.77
LED-large (seqlen: 16,384) (ours)	**46.63**	**19.62**	**41.83**

Table 11: Summarization results of Longformer-Encoder-Decoder (LED) on the arXiv dataset. Metrics from left to right are ROUGE-1, ROUGE-2 and ROUGE-L.

결론

- Instead of approaching to come up with a new variant of pretraining method, Reformer and Longformer tried to change the Full-Attention part of Transformers
    - (Reformer) Too much time consuming + Too much memory consuming
    - (Longformer) Too much time consuming

- Both approaches altered the $O(L^2)$ factor and then increased maximum L
    - (Reformer) Use LSH Algorithm to replace $O(L^2)$ factor with $O(L \log L)$
    - (Longformer) Use Sliding Window + Global Attention to replace $O(L^2)$ factor with $O(L)$

- Longformer also applied the Pretraining-Finetuning schema to this new variant of Transformers
    - (Longformer) Apply to downstream tasks such as QA, Coreference Resolution, Classification, and Summarization

- Future Works
    - (Reformer) Could be applied to generative tasks; long coherent text, time-series forecasting, music, image and video generation
    - (Longformer) Other pretraining objectives, especially for LED (to get benefit with the ability to handle long sequences) could be explored