Applied Deep Learning

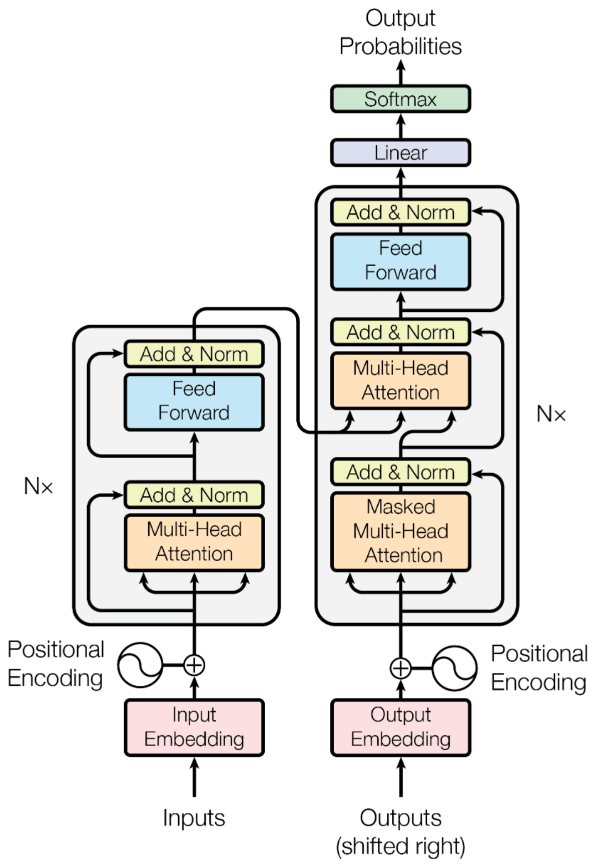
More on Transformers

April 14th, 2020 http://adl.miulab.tw

National Taiwan University

2

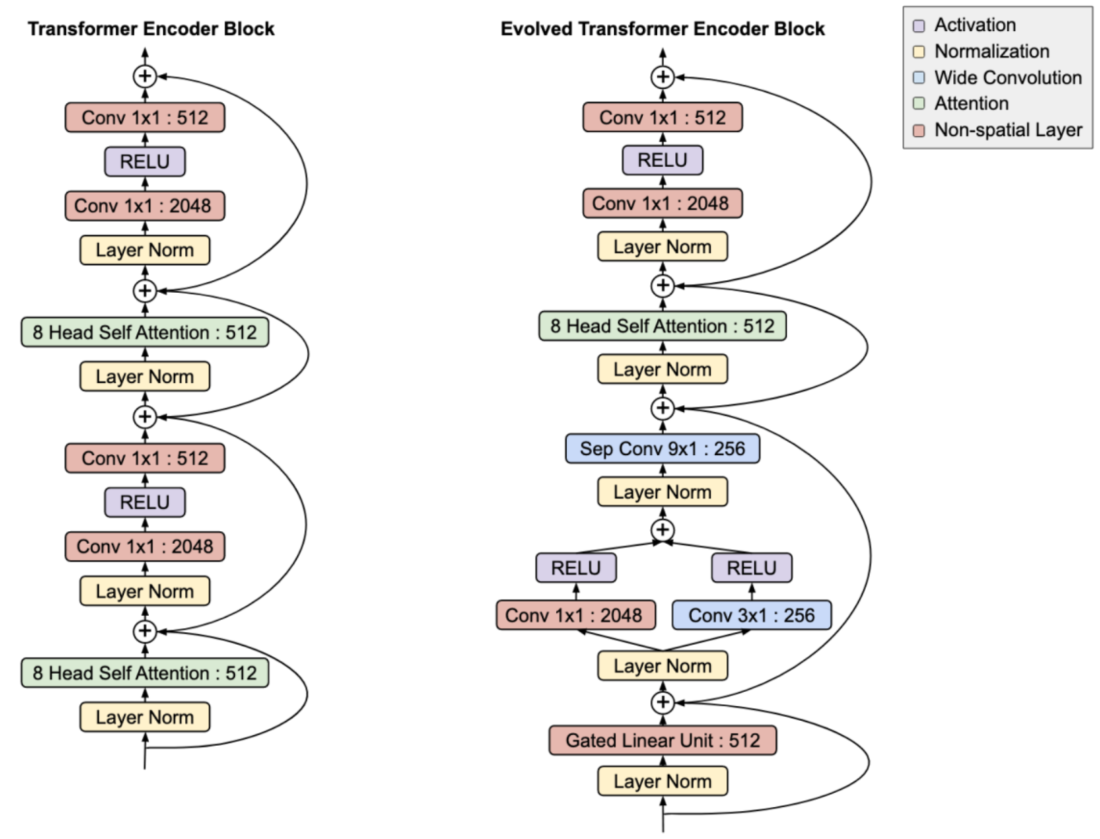
Why Transformer?



The Evolved Transformer

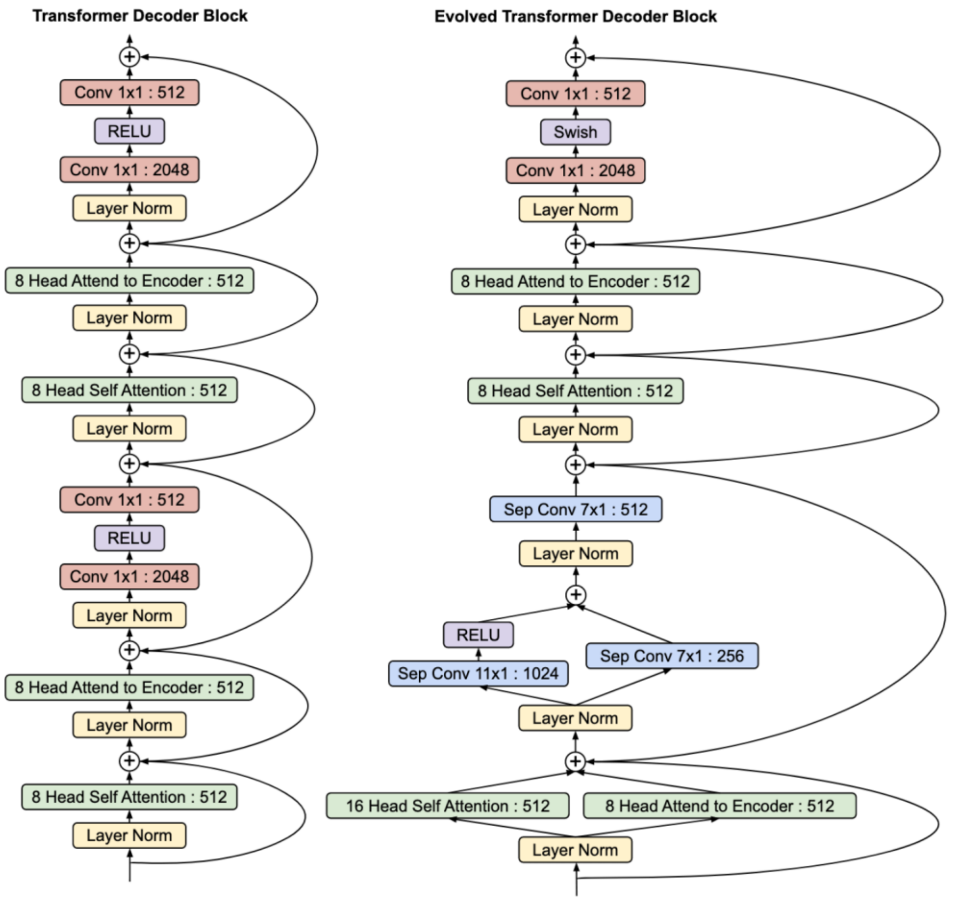
- Apply neural architecture search (NAS) on Transformer architecture
- It also proved to be efficient at smaller sizes, achieving the same quality as the original ”big” Transformer with 37.6% less parameters and outperforming the Transformer by 0.7 BLEU at a mobile-friendly model size of ∼7M parameters.

The Evolved Transformer



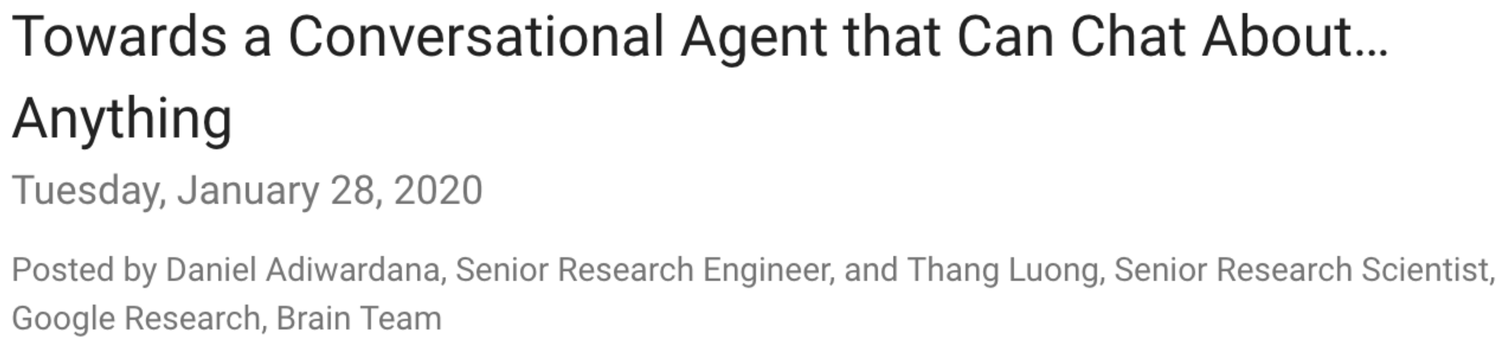
5

The Evolved Transformer


Transformer Decoder Block
Conv 1x1 : 512
RELU
Conv 1x1 : 2048
Layer Norm
8 Head Attend to Encoder : 512
Layer Norm
8 Head Self Attention : 512
Layer Norm
Conv 1x1 : 512
RELU
Conv 1x1 : 2048
Layer Norm
8 Head Attend to Encoder : 512
Layer Norm
8 Head Self Attention : 512
Layer Norm
Evolved Transformer Decoder Block
Conv 1x1 : 512
Swish
Conv 1x1 : 2048
Layer Norm
8 Head Attend to Encoder : 512
Layer Norm
8 Head Self Attention : 512
Layer Norm
Sep Conv 7x1 : 512
Layer Norm
RELU
Sep Conv 11x1 : 1024
Sep Conv 7x1 : 256
Layer Norm
16 Head Self Attention : 512
8 Head Attend to Encoder : 512
Layer Norm

The Evolved Transformer

- Google recently release a model “Meena” base on the ET.



Towards a Conversational Agent that Can Chat About… Anything

Tuesday, January 28, 2020

Posted by Daniel Adiwardana, Senior Research Engineer, and Thang Luong, Senior Research Scientist, Google Research, Brain Team

Though achieving great improvements…

- **No Recurrent Inductive Bias**: The Transformer trades the recurrent inductive bias of RNN’s for parallelizability. However, the recurrent inductive bias appears to be crucial for generalizing on different sequence modeling tasks of varying complexity.
- For instance, when it is necessary to model the hierarchical structure of the input, or when the distribution of input length is different during training and inference, i.e. when good length generalization is needed.

No Recurrent Inductive Bias

- *“The Importance of Being Recurrent for Modeling Hierarchical Structure”*
- Transformer vs LSTM
- Using task like logical inference to study the ability of capturing the underlying hierarchical structure of sequential data.

$$\text{(d (or f))} \sqsupset \text{(f (and a))}$$
$$\text{(d (and (c (or d))))} \;\#\; \text{(not f)}$$
$$\text{(not (d (or (f (or c)))))} \sqsubset \text{(not (c (and (not d))))}$$

No Recurrent Inductive Bias

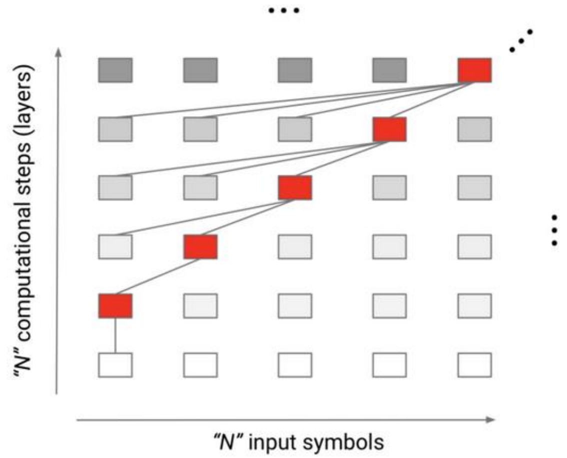
- The experiments showed that LSTMs are more robust and generalize better.
- This does not imply that LSTMs should always be preferred over non-recurrent architectures.
- In fact, both FAN- and CNN-based networks have proved to perform comparably or better than LSTM-based ones on a very complex task like machine translation

The Transformer Is Not Turing Complete

- While the Transformer executes a total number of operations that scales with the input size, the number of sequential operations is constant and independent of the input size, determined solely by the number of layers.
- Assuming finite precision, this means that the Transformer cannot be computationally universal.

The Transformer Is Not Turing Complete

- An intuitive example are functions whose execution requires the sequential processing of each input element. In this case, for any given choice of depth T, one can construct an input sequence of length N > T that cannot be processed correctly by a Transformer.



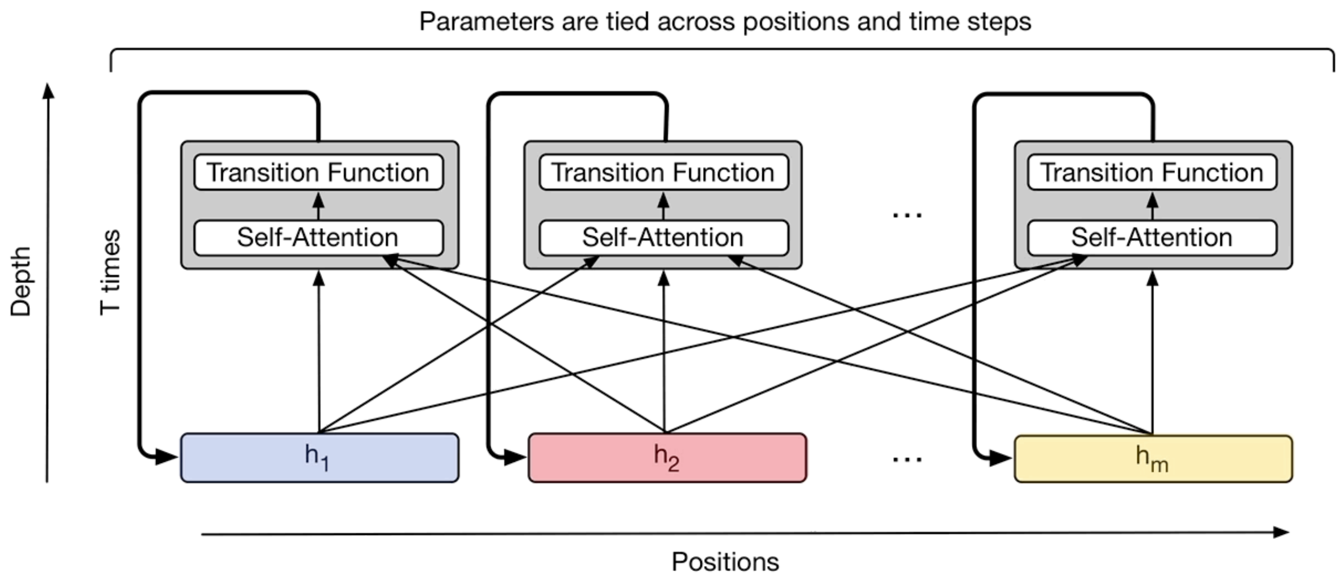
Lack of Conditional Computation

- The Transformer applies the same amount of computation to all inputs (as well as all parts of a single input). However, not all inputs need the same amount of computation and this can be conditioned on the complexity of the input.
- “*I arrived at the **bank** after crossing the **river***.”

Universal Transformers

- A Concurrent-Recurrent Sequence Model
- The Universal Transformer is an extension to the Transformer models which combines the parallelizability and global receptive field of the Transformer model with the recurrent inductive bias of RNNs.
- **Recurrence in depth**

How Does the UT Work?

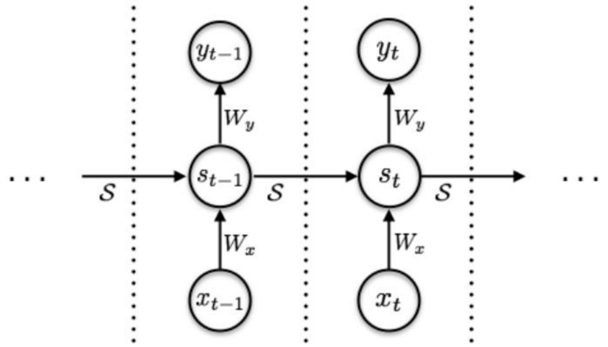


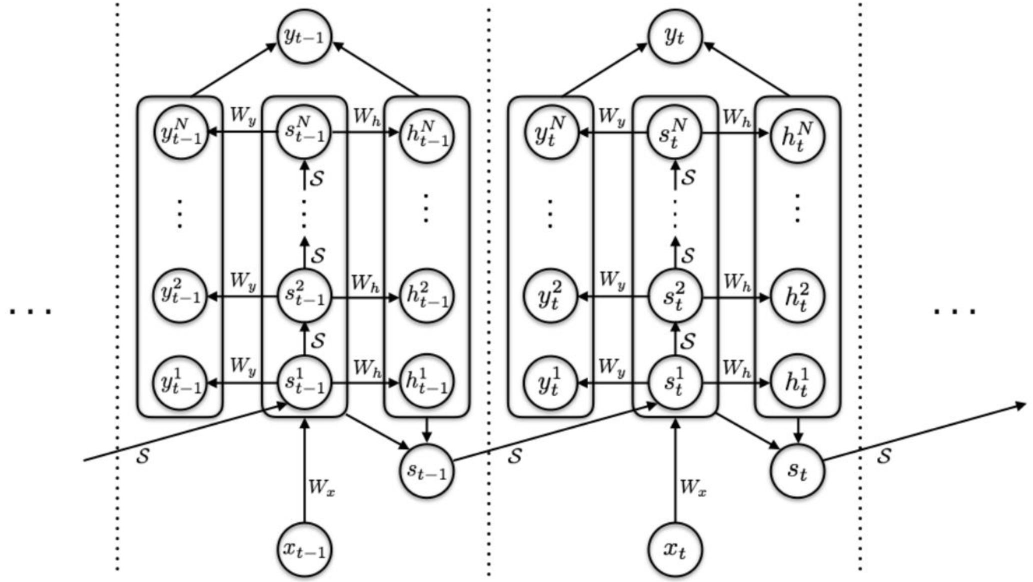
Dynamic Halting

- modulates the number of computational steps needed to process each input symbol dynamically based on a scalar pondering value that is predicted by the model at each step.
- uses an Adaptive Computation Time (ACT) mechanism, which was originally proposed for RNNs, to enable conditional computation.

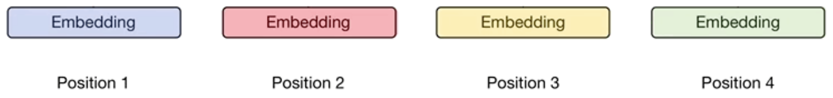
Adaptive Computation Time

- RNN vs RNN with ACT



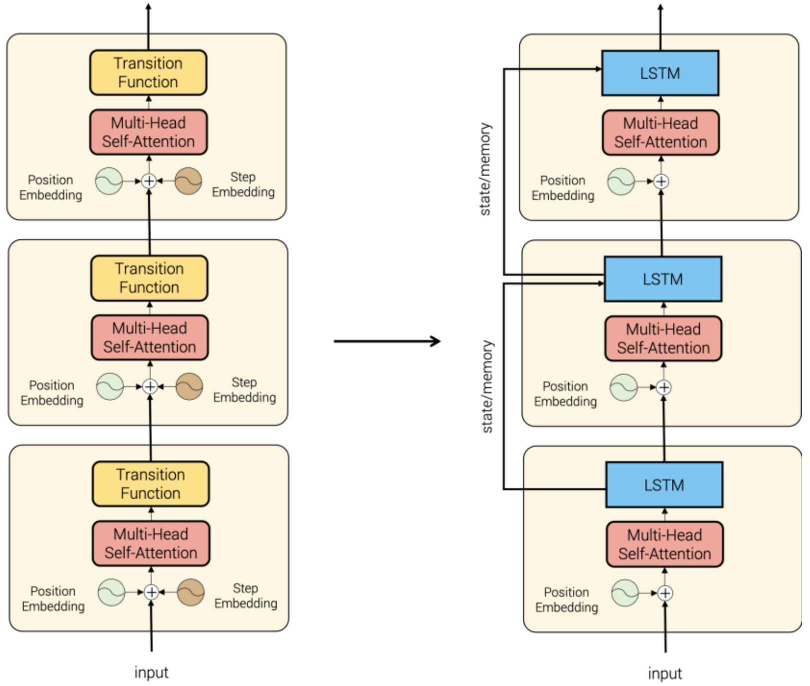


UT with Dynamic Halting



Notes

- weight sharing in depth leads to better performance of UTs (compared to the standard Transformer) on **very small datasets** and allows the UT to be a very data efficient model
- Transition functions can be replaced

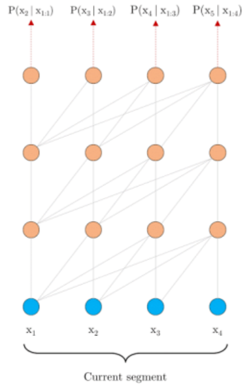


Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context

- The original Transformer: no recurrence
- 2nd generation (**Universal Transformer**): recurrence in depth
- 3rd generation (**Transformer-XL**): recurrence in length

Transformer for LM

- in language modeling, Transformers are currently implemented with a fixed-length context, i.e. a long text sequence is truncated into fixed-length segments of a few hundred characters, and each segment is processed separately.



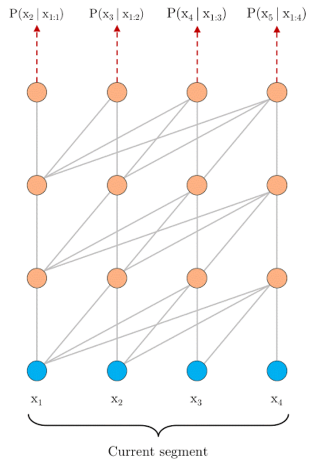
Transformer for LM

- This introduces two critical limitations:
- The algorithm is not able to model dependencies that are longer than a fixed length.
- The segments usually do not respect the sentence boundaries, resulting in context fragmentation which leads to inefficient optimization.

~~它不僅是一個能夠處理可~~變長度序列的模型，在多個任~~務中刷新了當前的最好性能。~~

Transformer-XL: Segment-level Recurrence

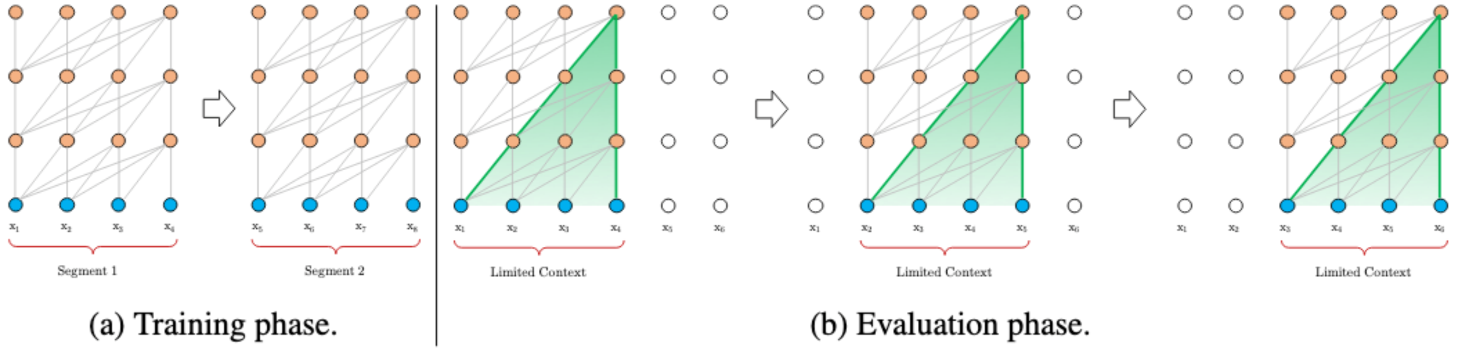
- During training, the representations computed for the previous segment are fixed and cached to be reused as an extended context when the model processes the next new segment.



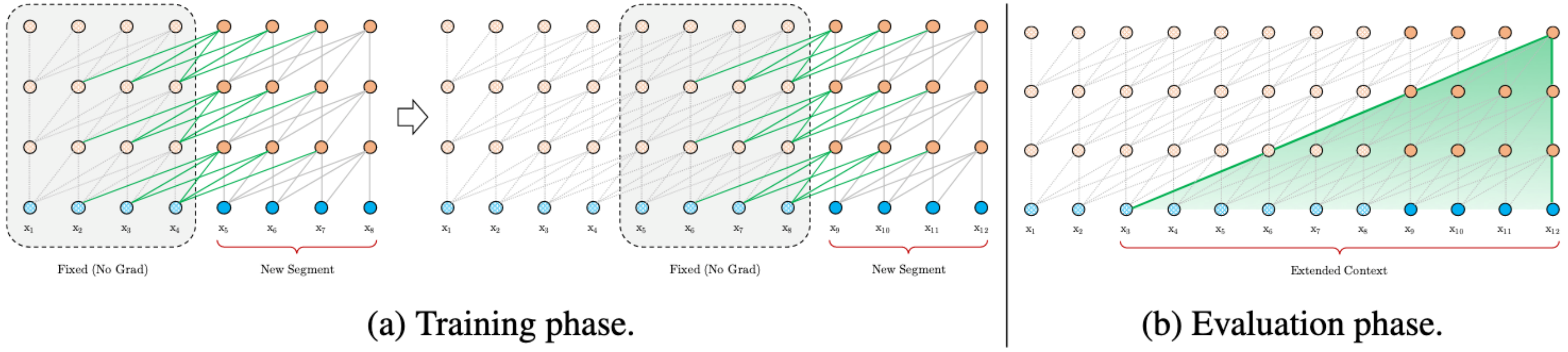
Relative Positional Encodings

- Naively applying segment-level recurrence does not work, however, because the positional encodings are not coherent when we reuse the previous segments.
- For example, consider an old segment with contextual positions [0, 1, 2, 3]. When a new segment is processed, we have positions [0, 1, 2, 3, 0, 1, 2, 3] for the two segments combined, where the semantics of each position id is incoherent through out the sequence.
- **parameterization to only encode the relative positional information based on content**

Transformer-XL



(a) Training phase. (b) Evaluation phase.



(a) Training phase. (b) Evaluation phase.

Transformer-XL

- Transformer-XL is up to 1,800+ times faster than a vanilla Transformer during evaluation on language modeling tasks, because no re-computation is needed.
- Transformer-XL has better performance in perplexity (more accurate at predicting a sample) on long sequences because of long-term dependency modeling, and also on short sequences by resolving the context fragmentation problem.

Reformer: The Efficient Transformer

- extending Transformer to even larger context windows runs into limitations. The power of Transformer comes from attention, the process by which it considers all possible pairs of words within the context window to understand the connections between them.
- So, in the case of a text of 100K words, this would require assessment of 100K x 100K word pairs, or 10 billion pairs for each step, which is impractical.

Reformer: The Efficient Transformer

- Since softmax is dominated by the largest elements, for each query q_i we only need to focus on the keys in K that are closest to q_i. For example, if K is of length 64K, for each q_i we could only consider a small subset of, say, the 32 or 64 closest keys.
- That is much more efficient, but how can we find the nearest neighbors among the keys?

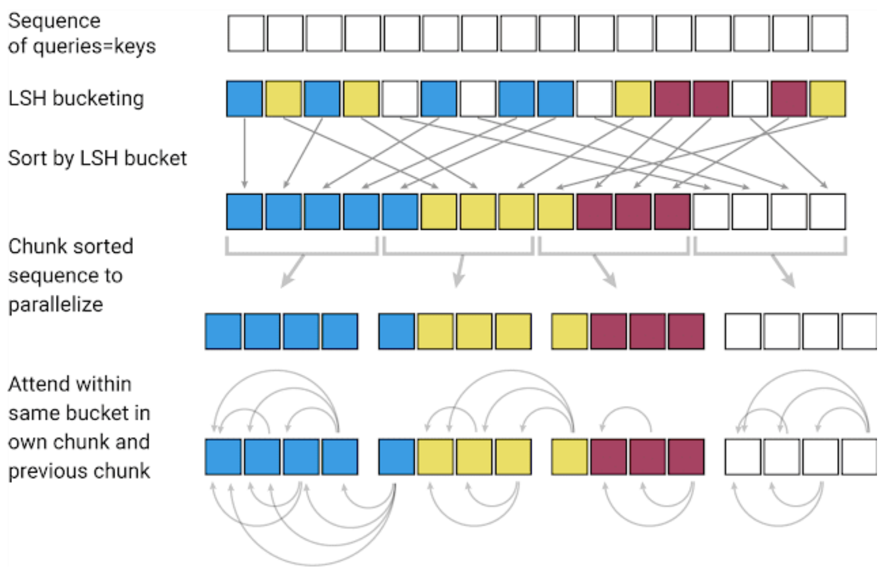
$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Locality Sensitive Hashing (LSH)

- The problem of finding nearest neighbors quickly in high-dimensional spaces can be solved by locality-sensitive hashing (LSH).
- A hashing scheme that assigns each vector x to a hash h(x) is called locality-sensitive if nearby vectors get the same hash with high probability and distant ones do not.

LSH Attention

- Attention is then applied within these much shorter chunks (and their adjoining neighbors to cover the overflow), greatly reducing the computational load.


Sequence of queries=keys
LSH bucketing
Sort by LSH bucket
Chunk sorted sequence to parallelize
Attend within same bucket in own chunk and previous chunk

The Memory Problem

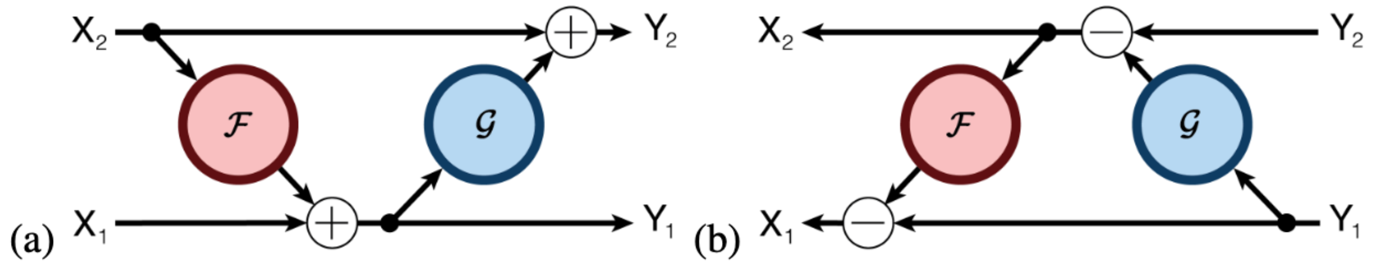
- While LSH solves the problem with attention, there is still a memory issue. A single layer of a network often requires up to a few GB of memory and usually fits on a single GPU, so even a model with long sequences could be executed if it only had one layer.
- But when training a multi-layer model with gradient descent, activations from each layer need to be saved for use in the backward pass. A typical Transformer model has a dozen or more layers, so memory quickly runs out if used to cache values from each of those layers.

Reversible Layer

- *“The Reversible Residual Network:* ***Backpropagation Without Storing Activations***”
- recompute the input of each layer on-demand during back-propagation, rather than storing it in memory.

Reversible Layer

- recompute the input of each layer on-demand during back-propagation, rather than storing it in memory.

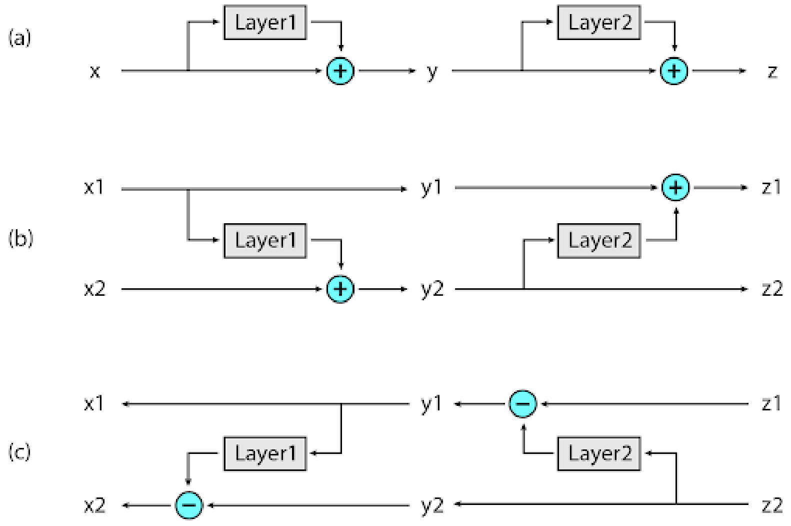


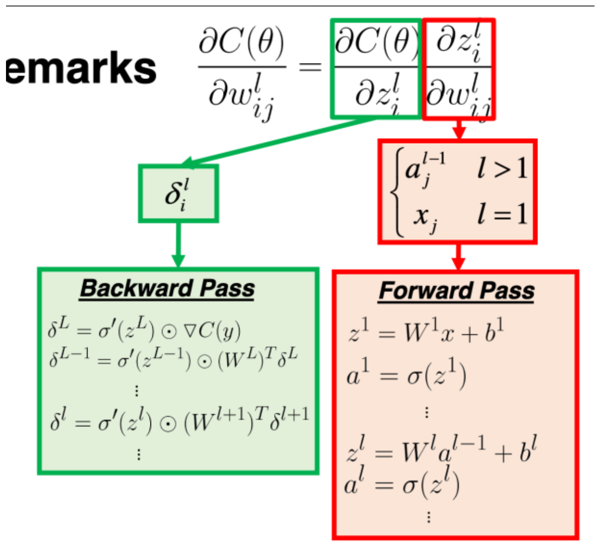
$$y_1 = x_1 + \mathcal{F}(x_2)$$
$$y_2 = x_2 + \mathcal{G}(y_1)$$

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

Reversible Layer

- **Reversible Transformer:** F becomes an attention layer while G becomes the feed-forward layer.




emarks
$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$
δ_i^l
$\begin{cases} a_j^{l-1} & l>1 \\ x_j & l=1 \end{cases}$
Backward Pass
$\delta^L = \sigma'(z^L) \odot \nabla C(y)$
$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$
$\vdots$
$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$
$\vdots$
Forward Pass
$z^1 = W^1 x + b^1$
$a^1 = \sigma(z^1)$
$\vdots$
$z^l = W^l a^{l-1} + b^l$
$a^l = \sigma(z^l)$
$\vdots$

Reformer: The Efficient Transformer

- combines the modeling capacity of a Transformer with an architecture that can be executed efficiently on long sequences and with small memory use even for models with a large number of layers.
- The Attention Problem: **Locality-Sensitive Hashing**
- The Memory Problem: **Reversible Layer**

References

- https://ai.googleblog.com/2020/01/towards-conversational-agent-that-can.html
- https://arxiv.org/abs/1901.11117
- https://arxiv.org/pdf/1803.03585.pdf
- https://mostafadehghani.com/2019/05/05/universal-transformers/
- https://arxiv.org/pdf/1807.03819.pdf
- https://arxiv.org/pdf/1603.08983.pdf
- https://medium.com/@moocaholic/adaptive-computation-time-act-in-neural-networks-part-1-2a28484b53df

References

- https://ai.googleblog.com/2020/01/towards-conversational-agent-that-can.html
- https://ai.googleblog.com/2019/01/transformer-xl-unleashing-potential-of.html
- https://arxiv.org/pdf/2001.04451.pdf
- https://arxiv.org/pdf/1707.04585.pdf