Accelerating Neural Transformer via an Average Attention Network

Biao Zhang[1,2], Deyi Xiong[3], Jinsong Su[1,2*]

[1]Xiamen University, [3]Soochow University

[2]Beijing Advanced Innovation Center for Language Resources

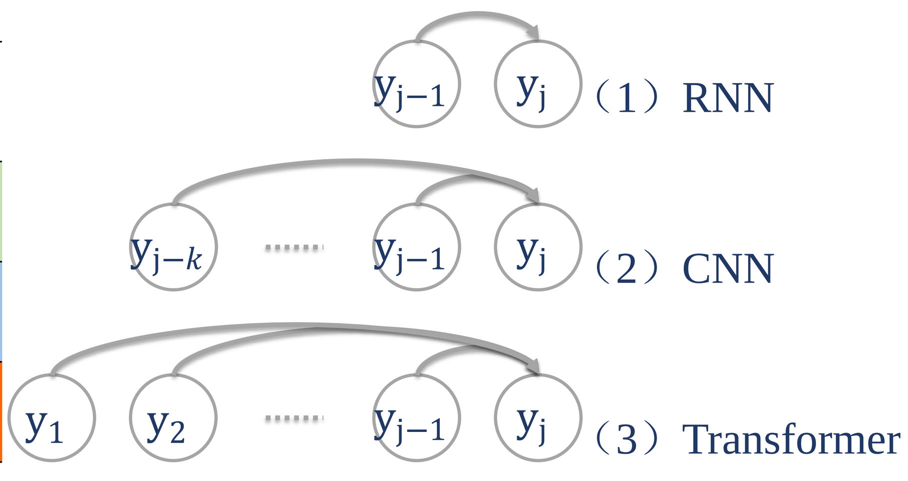
Motivation

The neural Transformer achieves state-of-the-art performance with solely attention network. Thanks to its full parallelization, Transformer can be trained very fast. However, because of the auto-regressive architecture and self-attention in the decoder:

Transformer is slow at decoding phase.

Below lists a comparison among CNN, RNN and Self-Attention when used for the decoder:

Model	Required Previous State During Decoding
RNN	O(1)
CNN	O(k)
Self-Attention	***O(n)***



Theoretically, Self-Attention needs O(n) previous hidden states to predict the next target word.

Could we reduce this complexity from O(n) to O(1)?

This is what our paper tries to solve.

The Approach: Average Attention Network

We propose average attention network (AAN). Instead of calculating dynamic weights over all previous hidden states,

AAN assumes that attention weights are equally distributed to each previous hidden state.

Its architecture and formal definition are shown below:

- Average Layer:

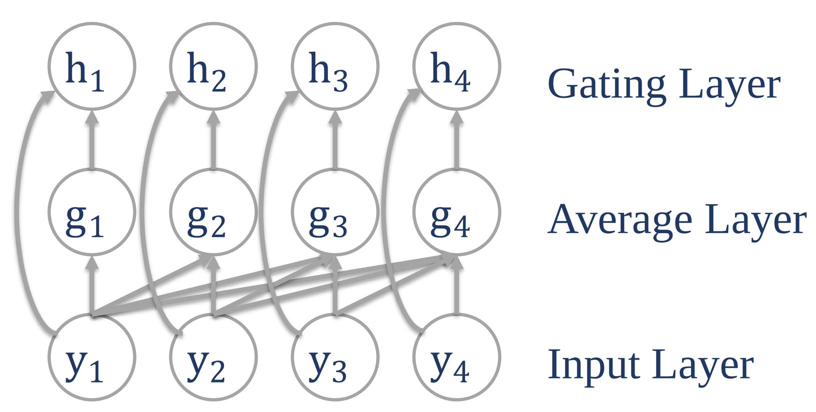
$$g_j = \mathrm{FFN}\left(\frac{1}{j}\sum_{k=1}^{j} y_k\right)$$

- Gating Layer

$$i_j, f_j = \sigma\left(W\left[y_j; g_j\right]\right)$$
$$\tilde{h}_j = i_j \odot y_j + f_j \odot g_j$$

- Output

$$h_j = \mathrm{LayerNorm}\left(y_j + \tilde{h}_j\right)$$



Intuitively, AAN replaces the original dynamically computed weights by the self-attention network in the decoder of Transformer with simple and fixed average weights ($\frac{1}{j}$).

In spite of its simplicity, the cumulative-average operation builds up dependencies with previous inputs so that the generated representations are not independent of each other.

Training and Decoding Acceleration

Training Acceleration

The cumulative operation in AAN disables the model from fully parallelizable training. Thanks to the simplicity of average,

cumulative-average operation can be implemented as purely matrix multiplication via mask trick.

For example, given four input embeddings (y_1, y_2, y_3, y_4), the average layer can be implemented as follows:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{pmatrix} \times \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} y_1 \\ \frac{y_1 + y_2}{2} \\ \frac{y_1 + y_2 + y_3}{3} \\ \frac{y_1 + y_2 + y_3 + y_4}{4} \end{pmatrix}$$

Mask Matrix

$$g_j = \mathrm{FFN}\left(\frac{1}{j}\sum_{k=1}^{j} y_k\right) \Rightarrow G = \mathrm{FFN}(MY)$$

Where G is the average output matrix, M is the mask matrix, Y is the input matrix. In this way, training with AAN will have the same computational complexity as that with Self-Attention.

Decoding Acceleration

Again, thanks to the simple average operation,

AAN can be accelerated during decoding via dynamic programming.

Concretely, we decompose the average layer into the following two steps:

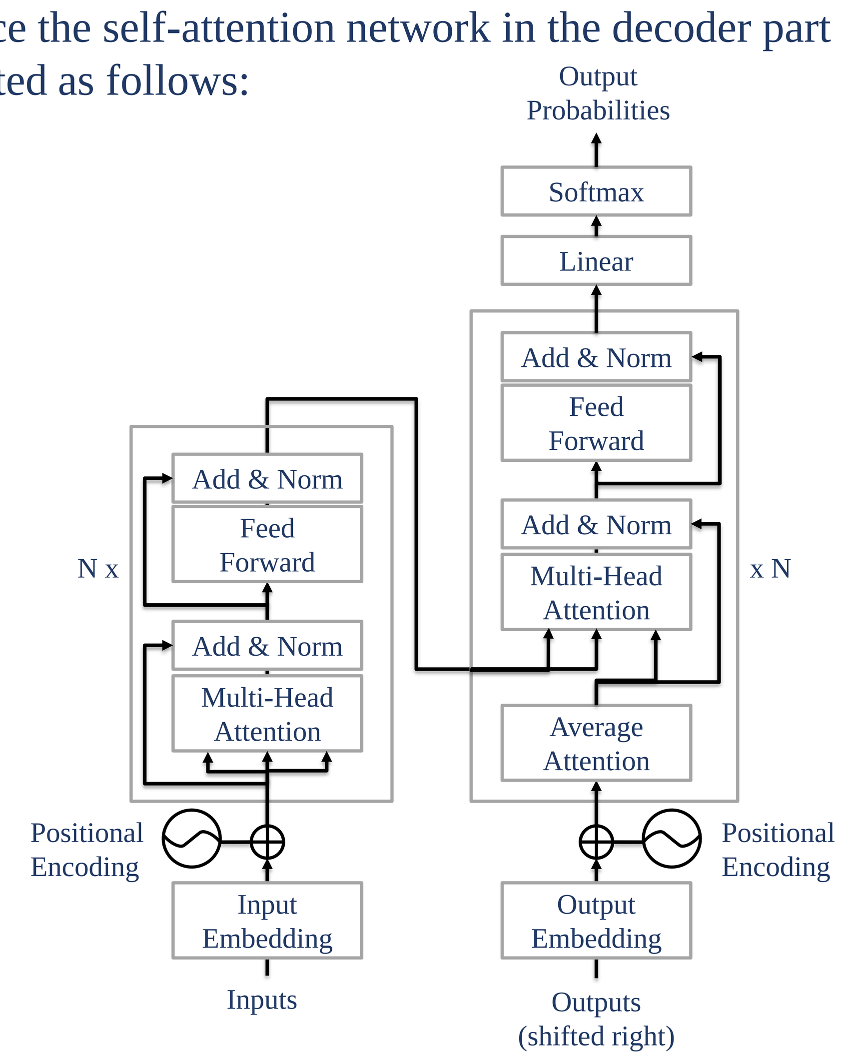
$$\tilde{g}_j = \tilde{g}_{j-1} + y_j$$
$$g_j = FFN\left(\frac{\tilde{g}_j}{j}\right)$$

Where $\tilde{g}_0 = \mathbf{0}$. In this way, decoder with AAN can compute the j-th input representation based on only one previous state $\tilde{g}_{j-1}$ during decoding, instead of relying on all previous states as the self-attention does.

Decoding with AAN requires O(1) previous state.

Neural Transformer with AAN

We use AAN to replace the self-attention network in the decoder part of Transformer. The overall architecture is illustrated as follows:



Experiments

Performance on WMT14 En-De Task

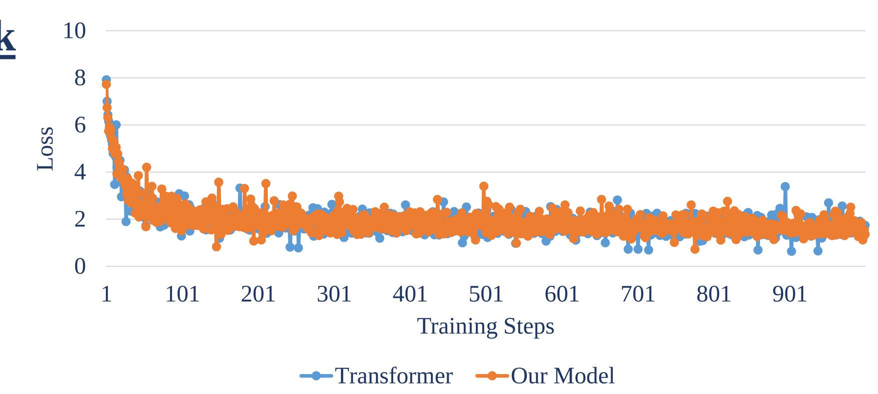
Translation performance of Transformer and Our model is almost the same.

Our model is not too sensitive to the FFN and Gate activation.

Model	BLEU
Transformer	26.37
Our Model	26.31
Our Model w/o FFN	26.05
Our Model w/o Gate	25.91

Model Convergence on WMT14 En-De Task

The convergence of Transformer and Our model is similar.



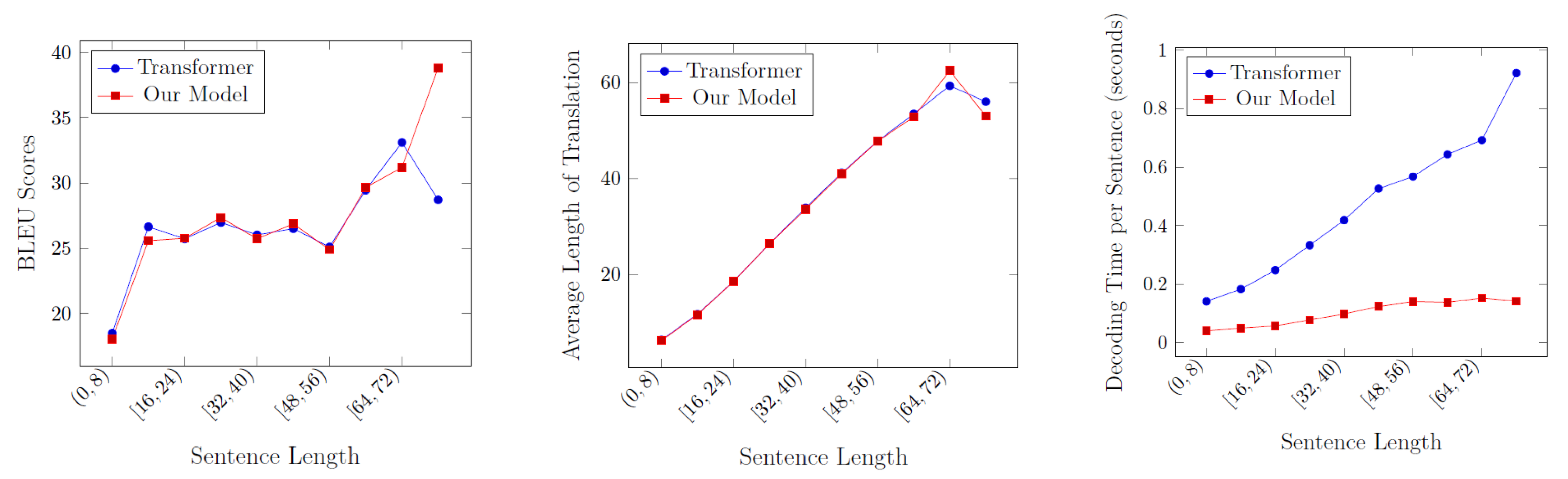
Speed on WMT14 En-De Task

Training speed is similar.

Decoding of AAN is ~4 times faster than that of Transformer.

	Transformer	Our Model	$\triangle_r$
Training	0.2474	0.2464	1.00
Decoding			
beam=4	0.1804	0.0488	3.70
beam=8	0.3576	0.0881	4.06
beam=12	0.5503	0.1291	4.26
beam=16	0.7323	0.1700	4.31
beam=20	0.9172	0.2122	4.32

Effects on Length (WMT14 En-De task)



The first two figures: ***Our model generates translations of the similar length and BLEU score as that of Transformer.***

The third figure: ***As sentence length increases, AAN achieves significantly better acceleration.***

Results on WMT17 Translation Tasks

	Case-sensitive BLEU				Average Decoding Time		
	Winner	Transformer	Our Model	$\triangle_d$	Transformer	Our Model	$\triangle_r$
En→De	28.3	27.33	27.22	-0.11	0.1411	0.02871	4.91
De→En	35.1	32.63	32.73	+0.10	0.1255	0.02422	5.18
En→Fi	20.7	21.00	20.87	-0.13	0.1289	0.02423	5.32
Fi→En	20.5	25.19	24.78	-0.41	0.1285	0.02336	5.50
En→Lv	21.1	16.83	16.63	-0.20	0.1850	0.03167	5.84
Lv→En	21.9	17.57	17.51	-0.06	0.1980	0.03123	6.34
En→Ru	29.8	27.82	27.73	-0.09	0.1821	0.03140	5.80
Ru→En	34.7	31.51	31.36	-0.15	0.1595	0.02778	5.74
En→Tr	18.1	12.11	11.59	-0.52	0.2078	0.02968	7.00
Tr→En	20.1	16.19	15.84	-0.35	0.1886	0.03027	6.23
En→Cs	23.5	21.53	21.12	-0.41	0.1150	0.02425	4.74
Cs→En	30.9	27.49	27.45	-0.04	0.1178	0.02659	4.43

On six different language pairs, our conclusion is the same.