Accelerating Neural Transformer via an Average Attention Network
Biao Zhang¹, ², Deyi Xiong³, Jinsong Su¹, ²*
¹Xiamen University, ²Soochow University
²Beijing 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:

<table>
<thead>
<tr>
<th>Model</th>
<th>Required Previous State During Decoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN</td>
<td>O(1)</td>
</tr>
<tr>
<td>CNN</td>
<td>O(k)</td>
</tr>
<tr>
<td>Self-Attention</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

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:
  \[ g_i = \text{FFN}(\frac{1}{N} \sum_{j=1}^{N} y_j) \]

• Gating Layer
  \[ i_j = \sigma(\text{FFN}(y_j; g_j)) \]
  \[ h_i = i_j g_i + f_i \circ g_i \]

• Output
  \[ y_i = \text{LayerNorm}(y_i + h_i) \]

Intuitively, AAN replaces the original dynamically computed weights by the self-attention network in the decoder of Transformer with simple and fixed average weights (i.e., 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:

\[
\begin{bmatrix}
1/4 & 1/4 & 1/4 & 1/4
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1/2 & 1/2 & 0 & 0 \\
1/4 & 1/4 & 1/4 & 1/4 \\
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\end{bmatrix}
= \begin{bmatrix}
y_1 + y_2 + y_3 + y_4 \\
\frac{1}{2} y_1 + \frac{1}{2} y_2 \\
\frac{1}{4} y_1 + \frac{1}{4} y_2 + \frac{1}{4} y_3 + \frac{1}{4} y_4 \\
\end{bmatrix}
\]

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:

\[
\begin{bmatrix}
1/4 & 1/4 & 1/4 & 1/4
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1/2 & 1/2 & 0 & 0 \\
1/4 & 1/4 & 1/4 & 1/4 \\
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\end{bmatrix}
= \begin{bmatrix}
y_1 + y_2 + y_3 + y_4 \\
\frac{1}{2} y_1 + \frac{1}{2} y_2 \\
\frac{1}{4} y_1 + \frac{1}{4} y_2 + \frac{1}{4} y_3 + \frac{1}{4} y_4 \\
\end{bmatrix}
\]

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:

\[
\tilde{g}_j = \tilde{g}_j + y_j \\
\tilde{g}_j = \text{FFN}(\tilde{g}_j)
\]

Where \(\tilde{g}_0 = 0\). 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 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.

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