Sharing Attention Weights for Fast Transformer

Tong Xiao[1,2] , **Yinqiao Li**[1] , **Jingbo Zhu**[1,2] , **Zhengtao Yu**[3] and **Tongran Liu**[4]
[1]Northeastern University, Shenyang, China
[2]NiuTrans Co., Ltd., Shenyang, China
[3]Kunming University of Science and Technology, Kunming, China
[4]CAS Key Laboratory of Behavioral Science, Institute of Psychology, CAS, Beijing, China
{xiaotong, zhujingbo}@mail.neu.edu.cn, {li.yin.qiao.2012, ztyu}@hotmail.com, liutr@psych.ac.cn

Abstract

Recently, the Transformer machine translation system has shown strong results by stacking attention layers on both the source and target-language sides. But the inference of this model is slow due to the heavy use of dot-product attention in auto-regressive decoding. In this paper we speed up Transformer via a fast and lightweight attention model. More specifically, we share attention weights in adjacent layers and enable the efficient re-use of hidden states in a vertical manner. Moreover, the sharing policy can be jointly learned with the MT model. We test our approach on ten WMT and NIST OpenMT tasks. Experimental results show that it yields an average of 1.3X speed-up (with almost no decrease in BLEU) on top of a state-of-the-art implementation that has already adopted a cache for fast inference. Also, our approach obtains a 1.8X speed-up when it works with the AAN model. This is even 16 times faster than the baseline with no use of the attention cache.

1 Introduction

In recent years, neural models have led to great improvements in machine translation (MT). Approaches of this kind make it possible to learn good mappings between sequences via deep networks and attention mechanisms [Sutskever *et al.*, 2014; Bahdanau *et al.*, 2015; Luong *et al.*, 2015b]. Recent work has explored an architecture that just consists of stacked attentive and feed-forward networks (call it Transformer) [Vaswani *et al.*, 2017]. It makes use of multi-layer dot-product attention to capture the dependency among language units. Beyond this, training this kind of model is fast because we can parallelize computation over all positions of the sequence on modern graphics processing units (GPUs). These properties make Transformer popular in recent MT evaluations and industrial deployments.

However, standard implementations of Transformer are prone to slow inference though fast in training. At test time, the system produces one target word each time until an end symbol is reached. This process is auto-regressive and slow because we have to run dot-product attention for each position rather than batching the computation of the sequence. The situation is even worse if 6 or more attention layers are stacked and the attention model occupies the inference time. To address this issue, efficient networks have been investigated. For example, one can replace dot-product attention with additive attention and use average attention models instead [Zhang *et al.*, 2018], or explore non-autoregressive decoders that benefit from the trick of batched matrix operations over the entire sequence [Gu *et al.*, 2017]. But these methods either lose the explicit model of word dependencies, or require complicated networks that are hard to train.

In this work, we observe that the attention model shares a similar distribution among layers in weighting different positions of the sequence. This experience lead us to study the issue in another line of research, in which we reduce redundant computation and re-use some of the hidden states in the attention network. We propose a method to share attention weights in adjacent layers (call it shared attention network, or SAN for short). It leads to a model that shares attention computation in the stacked layers vertically. In addition to the new architecture, we develop a joint method to learn sharing policies and MT models simultaneously. As another "bonus", SAN reduces the memory footprint because some hidden states are kept in the same piece of memory.

SAN is simple and can be implemented in a few hours by anyone with an existing kit of Transformer. Also, it is orthogonal to previous methods and is straightforwardly applicable to the variants of Transformer. We test our approach in a state-of-the-art system where an attention cache is already in use for speed-up. Experimental results on ten WMT and NIST OpenMT tasks show an average of 1.3X speed-up with almost no decrease in BLEU. More interestingly, it obtains a bigger speed-up (1.8X) when working with the AAN model. The best result is 16 times faster than the baseline where no cache is adopted.

2 The Transformer System

The Transformer system follows the popular encoder-decoder paradigm. On the encoder side, there are a number of identical stacked layers. Each of them is composed of a self-attention sub-layer and a feed-forward sub-layer. In Transformer, the attention model is scaled dot-product attention. Let l be the length of the source sequence. The input of the attention sub-layer is a tuple of (Q, K, V), where $Q \in \mathbb{R}^{l \times d_k}$, $K \in \mathbb{R}^{l \times d_k}$ and $V \in \mathbb{R}^{l \times d_v}$ are the matrices of queries, keys,

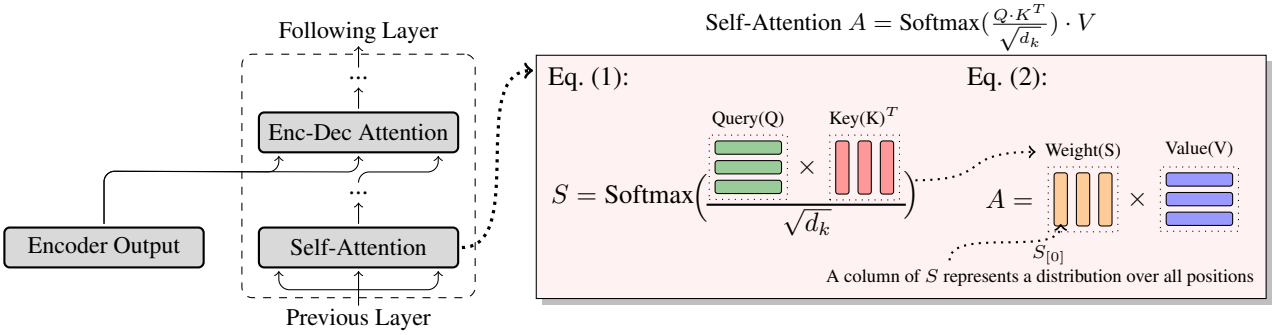


Figure 1: Decoder-side attention sub-layers in Transformer

and values packed over the sequence. In self-attention, we first compute the dot-product of queries and keys, followed by the rescaling and softmax operations.

$$S = \text{Softmax}(\frac{Q \cdot K^T}{\sqrt{d_k}}) \quad (1)$$

S is an $l \times l$ matrix, where entry (i, j) represents the strength of connecting position i with position j. Note that S is essentially a weight (or scalar) matrix where every column represents a distribution. The output of self-attention is simply defined as the weighted sum of values:

$$A = S \cdot V \quad (2)$$

Here Q, K and V are generated from the same source with a linear transformation. The self-attention result is then fed into a fully connected feed-forward network (FFN).

The decoder shares a similar structure with the encoder. Apart from the self-attention sub-layer, an encoder-decoder attention sub-layer is introduced to model the correspondence between source positions and target positions. Basically, the encoder-decoder attention has the same form as Eqs. (1) and (2), where the queries come from the output of the previous layer, and the keys and values come from the output of the encoder. See Figure 1 for an illustration of the attention model used in Transformer.

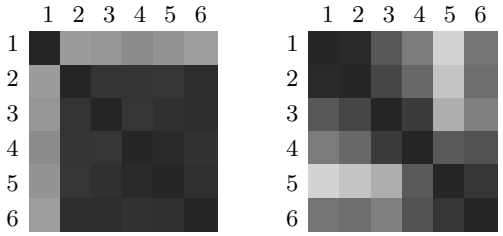
Note that the matrix multiplications in Eqs. (1) and (2) are time consuming. It is a bigger problem for inference because Eqs. (1) and (2) repeat for each position until we finish the generation.

3 Shared Attention Networks

In this work we speed up the decoder-side attention because the decoder is the heaviest component in Transformer.

3.1 Attention Weights

Self-attention is essentially a procedure that fuses the input values to form a new value at each position. Let $S_{[i]}$ be column i of weight matrix S. For position i , we first compute $S_{[i]}$ to weight all positions (as in Eq. (1)), and then compute the weighted sum of values by $S_{[i]}$ (as in Eq. (2)). In column vector $S_{[i]}$, element $S_{i,j}$ indicates the contribution that we fuse the value at position j to position i. Intuitively, the attention weight $S_{[i]}$ should not be volatile in different levels of language representation because the correlations between positions somehow reflect the dependency of language units. For example, for an English sentence, the subject and the verb correlate well no matter how many layers we make on top of the input sequence. On the other hand, the subject and the adverbial may not have a big impact to each other in all stacked layers.



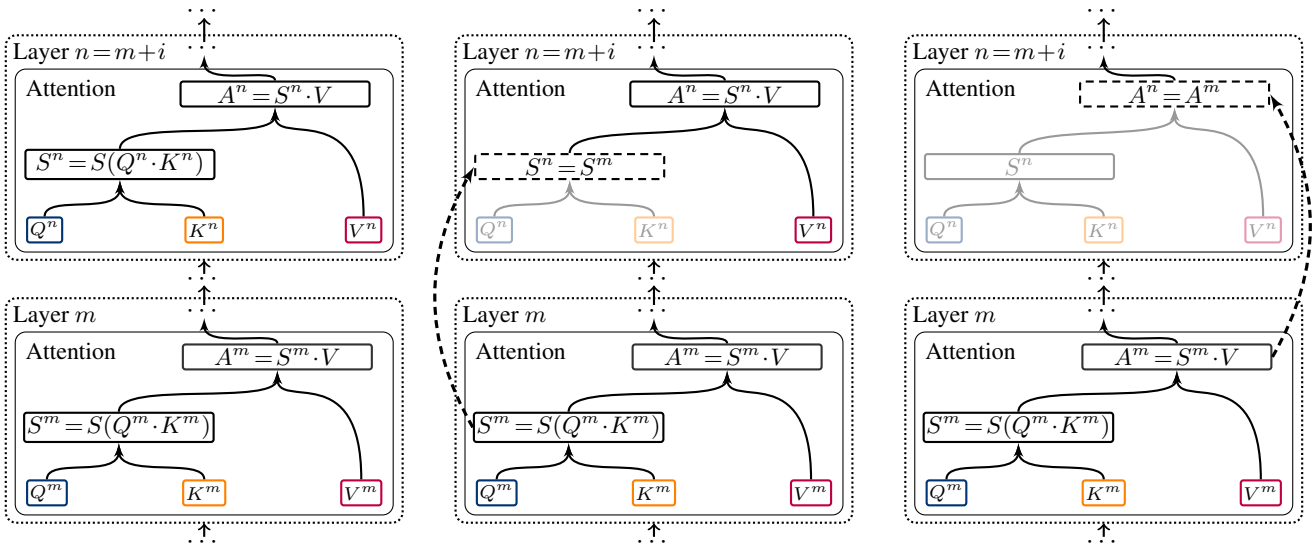
(a) Self-Attention (b) Enc-Dec Attention

Figure 2: The Jensen-Shannon divergence of the attention weights for every pair of layers on the WMT14 English-German task (a dark cell means the distributions are similar)

To verify this, we compute the Jensen-Shannon (JS) divergence to measure how the attention weight distribution of a layer is different from another [Lin, 1991]. We choose the JS divergence because it is symmetric and bounded. For multi-head attention, we regard different heads as separate channels. We compute the JS score for each individual head and then average them for final output. Figure 2 shows that the system generates similar weights over layers. For self-attention, layers 2-6 almost enjoy the same weight distribution. For encoder-decoder attention, we observe a larger variance but good similarities still exist among two or three adjacent layers (see entries around the diagonal of Figure 2(b)). All these show the possibility of removing redundant computation in Transformer.

3.2 The Model

An obvious next step is to develop a faster attention model that makes efficient re-use of the states in Eqs. (1) and (2),



(a) Standard Transformer Attention (b) SAN Self-Attention (c) SAN Encoder-Decoder Attention

Figure 3: Comparison of the standard attention model and the SAN model

instead of computing everything on the fly. In this work we present a shared attention network (SAN) to share weight matrix S for adjacent layers. The idea is that we just compute the weight matrix once and reuse it for upper-level layers. Here we describe SAN for both the self-attention and encoder-decoder attention models.

- **SAN Self-Attention**. We define the self-attention weight matrix in layer m as:

$$S^m = s(Q^m, K^m) \tag{3}$$

where $s(\cdot,\cdot)$ is the function described in Eq. (1), Q^m and K^m are the inputs, and S^m is the attention weight for the output. In SAN, we can share S^m with the layers above m, like this

$$\begin{aligned} S^{m+i} &= s(Q^m, K^m) \\ \text{for} &\quad i \in [1, \pi - 1] \end{aligned} \tag{4}$$

where π indicates how many layers share the same attention weights. For example, in a 6-layer decoder, we can share the self-attention weights for every two layers ($\pi = 2$), or share the weights for the first two layers ($\pi_1 = 2$) and let the remaining 4 layers use another weights ($\pi_2 = 4$). We discuss the sharing strategy in the later part of the section.

- **SAN Encoder-Decoder Attention**. For encoder-decoder attention, we do the same thing as in self-attention, but with a trick for further speed-up. In encoder-decoder attention, keys and values are from the output of the encoder, i.e., K and V have already been shared among layers on the decoder side. In response, we can share $A = S \cdot V$ for encoder-decoder attention layers. This can be described as

$$\begin{aligned} A^{m+i} &= A^m \\ &= S^m \cdot V \\ \text{for} &\quad i \in [1, \pi - 1] \end{aligned} \tag{5}$$

where A^m is the attention output of layer m, V is the context representation generated by the encoder. See Figure 3 for a comparison of the standard attention model and SAN.

In addition to system speed-up, SAN also reduces the memory footprint. In SAN, we just need one data copy of weight matrix for a layer block, rather than allocating memory space for every layer. Moreover, the linear transformation of the input (i.e., Q and K) can be discarded when the attention weights come from another layer. It reduces both the number of model parameters and the memory footprint in inference.

Another note on SAN. SAN is a process that simplifies the model and re-uses hidden states in the network. It is doing something similar to systems that share model parameters in different levels of the network [Wu *et al.*, 2016; Yang *et al.*, 2018; Luong *et al.*, 2015a]. Such methods have been proven to improve the robustness of neural models on many natural language processing tasks. Sharing parameters and/or hidden states can reduce the model complexity. Previous work has pointed out that MT systems cannot benefit a lot from very deep and complex networks [Vaswani *et al.*, 2017; Britz *et al.*, 2017]. SAN might alleviate this problem and makes it easier to train neural MT models. For example, in our experiments, we see that SAN can improve translation quality in some cases in addition to considerable speed-ups.

3.3 Learning to Share

The remaining issue is how to decide which layers can be shared. A simple way is to use the same setting of π for the

1: **Function** LEARNTOSHARE ($layers$, $model$)
2: **while** policy $\{\pi_i\}$ does change **do**
3: learn a new $model$ given policy $\{\pi_i\}$
4: learn a new policy $\{\pi_i\}$ on $layers$ given $model$
5: **return** $\{\pi_i\}$ & $model$

Figure 4: Joint learning of MT models and sharing policies

entire layer stack, and tune it on a development set. For example, we can try to share weights on layer blocks consisting of two layers, or three layers, or all layers ($\pi = 2$, or 3 , ...), and use the tuned π on test data.

But a uniform sharing strategy might not be optimal because we need to control the degree of sharing in difference levels of the network. For example, for the case in Figure 2(a), a good choice is to share weights for layers 2-6 and leave layer 1 as it is. Here we present a method that learns the sharing strategy in a dynamic way. To do this, we choose $ln(2)-$ JS divergence as the measure of the similarity between weights of layer i and layer j (denoted as $\mu(i,j)$). Given a layer block ranging from layer m to layer n (denoted as $b(m,n)$), the similarity over the block is defined as

$$sim(m,n) = \frac{\sum_{i=m}^{n}\sum_{j=m}^{n}(1-\delta(i,j))\mu(i,j)}{(n-m+1)\cdot(n-m)} \quad (6)$$

where $n-m+1$ is the size of the block, and $\delta(i,j)$ is the Kronecker delta function. $sim(m,n)$ measures how the weight of a layer is similar to that of another layer in block $b(m,n)$. We can do sharing when $sim(m,n) \geq \theta$ where θ is the parameter that controls how often a layer is shared.

We begin with layer 1 and search for the biggest block that satisfies the criterion. This process repeats until all the layers are considered, resulting in N layer blocks. For simplicity, we use $\{\pi_1, ...,\pi_N\}$ (or $\{\pi_i\}$) to represent the blocks in a bottom-up manner (call it sharing policy), where π_i is the size of block i. Obviously, for an M-layer stack we have $\sum_{i=1}^{N}\pi_i = M$.

Once we have a sharing policy, we need to re-train the MT model. It in turn leads to new attention weights and possibly a better policy. A desirable way is to continue learning until the model converges. To this end, we present a joint learning method that trains MT models and sharing policies simultaneously (Figure 4). In the method, MT training and policy learning loops for iterations, and the result of the final round is returned when there is no new update of the model.

4 Experiments

We experimented with our approach on WMT and NIST translation tasks.

4.1 Experimental Setup

The bilingual and evaluation data came from three sources

- WMT14 (En-De). We used all bilingual data provided within the WMT14 English-German task. We chose newstest 2013 as the tuning data, and newstest 2014 as the test data.

Source	Lang.	Train		Tune		Test	
		sent.	word	sent.	word	sent.	word
WMT14	En-De	4.5M	225M	3000	130K	3003	133K
WMT17	En-De	5.9M	276M	8171	356K	3004	128K
	De-En					3004	128K
	En-Fi	2.6M	108M	8870	330K	3002	110K
	Fi-En					3002	110K
	En-Lv	4.5M	115M	2003	90K	2001	88K
	Lv-En					2001	88K
	En-Ru	25M	1.2B	8819	391K	3001	132K
	Ru-En					3001	132K
NIST12	Zh-En	1.9M	85M	1164	227K	1357	198K

Table 1: Data statistics (# of sentences and # of words)

- WMT17 (En-De, De-En, En-Fi, Fi-En, En-Lv, Lv-En, En-Ru and Ru-En). We followed the standard data setting of the bidirectional translation tasks of German-English, Finnish-English, Latvian-English, and Russian-English. For tuning, we concatenated the data of newstest 2014-2016. For test, we chose newstest 2017.
- NIST12 (Zh-En). We also used parts of the bitext of NIST OpenMT12 to train a Chinese-English system[1]. The tuning and test sets were MT06 and MT08.

For Chinese, all sentences were word segmented by the segmentation system in the NiuTrans toolkit [Xiao *et al.*, 2012]. For other languages, we ran the official script of WMT for tokenization. All sentences of more than 50 words were removed for the NIST Zh-En task, and sentences of more than 80 words were removed for the WMT tasks. For all these tasks, sentences were encoded using byte-pair encoding, where we used a shared source target vocabulary of 32K tokens. See Table 1 for statistics of the data.

We used standard implementation of Transformer. Early versions of its inference system simply compute the attention output for target positions individually. This way is straightforward but with a double counting problem. For a stronger baseline, we chose the system with an attention cache that kept the attention output of previous positions in cache and re-used it in following steps.

The Transformer system used in our experiments consisted of a 6-layer encoder and a 6-layer decoder. By default, we set $d_k = d_v = 512$ and used 2,048 hidden units in the FFN sub-layers. We used multi-head attention (8 heads) because it was shown to be effective for state-of-the-art performance [Vaswani *et al.*, 2017]. Dropout (rate $= 0.1$) and label smoothing ($\epsilon_{ls} = 0.1$) methods were adopted for regularization and stabilizing the training [Szegedy *et al.*, 2016]. We trained the model using Adam with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-9}$ [Kingma and Ba, 2014]. The learning rate was scheduled as described in [Vaswani *et al.*, 2017]: $lr = d^{-0.5}\cdot min(t^{-0.5}, t\cdot 4\text{k}^{-1.5})$, where t is the step number. All models were trained for 100k steps with a mini-batch of 4,096 tokens on machines with 8 Nvidia 1080Ti GPUs except En-Fi (60k steps), Fi-En (60k steps) and Zh-En (24k steps). Every model was ensembled from the 5 latest checkpoints in

[1]LDC2000T46, LDC2000T47, LDC2000T50, LDC2003E14, LDC2005T10, LDC2002E18, LDC2007T09 and LDC2004T08

Source	Language	Model	BLEU	Δ_{BLEU}	Speed	Δ_{Speed}
WMT14	En-De	Baseline	27.52	0.00	1.03K	0.00%
		SAN	27.69	+0.17	1.44K	+39.81%
WMT17	En-De	Baseline	28.90	0.00	1.03K	0.00%
		SAN	28.82	-0.08	1.43K	+38.82%
	De-En	Baseline	34.57	0.00	0.99K	0.00%
		SAN	34.75	+0.18	1.38K	+39.19%
	En-Fi	Baseline	21.80	0.00	1.02K	0.00%
		SAN	21.45	-0.35	1.36K	+33.82%
	Fi-En	Baseline	24.94	0.00	1.02K	0.00%
		SAN	25.25	+0.31	1.25K	+23.28%
	En-Lv	Baseline	15.80	0.00	0.94K	0.00%
		SAN	16.08	+0.28	1.31K	+39.01%
	Lv-En	Baseline	18.06	0.00	0.92K	0.00%
		SAN	17.97	-0.09	1.26K	+36.28%
	En-Ru	Baseline	29.93	0.00	1.02K	0.00%
		SAN	29.51	-0.42	1.29K	+26.17%
	Ru-En	Baseline	33.63	0.00	1.01K	0.00%
		SAN	33.36	-0.27	1.23K	+21.17%
NIST12	Zh-En	Baseline	38.59	0.00	0.84K	0.00%
		SAN	38.19	-0.40	1.02K	+21.34%
Average		Baseline	27.37	0.00	0.98K	0.00%
		SAN	27.31	-0.07	1.30K	+31.98%

Table 2: BLEU scores (%) and translation speeds (token/sec) on the WMT and NIST tasks

training. For inference, both beam search and batch decoding methods were used (beam size = 4 and batch size = 16).

For our approach, we applied SAN to self-attention and encoder-decoder sub-layers on the decoder side. We learned sharing policies as in Figure 4. θ was tuned on the tuning data, which resulted in an optimal range of $[0.3, 0.4]$ for self-attention and $[0.4, 0.5]$ for encoder-decoder attention.

4.2 Results

We report the translation quality (in BLEU[%]) and speed (in token/sec) on all ten of the tasks (Table 2). We see, first of all, that SAN significantly improves the translation speed for all these languages. The average speed-up is 1.3X. Also, there is a very modest BLEU decrease, but not significant. These results indicate that SAN is robust and can improve a strong baseline on a wide range of translation tasks. Another interesting finding here is that the speed improvement on En-Ru, Ru-En and Zh-En is not as large as that on other language pairs. This is because we share fewer layers (i.e., larger θ) on these tasks to preserve good BLEU scores. Note that Russian and Chinese are very difficult languages for translation, and we need a complicated network to model the structure divergence. Less sharing is preferred to keep the expressive power for these languages.

To modulate the degree of sharing, we study the system behavior under different settings of θ (Table 3) . For comparison, we report the result of uniform $\{\pi_i\}$. Due to the limited space, we present the result on the WMT14 En-De task in the following sensitivity analysis. For uniform sharing, $\{\pi_i\}$ is set to $\{6\}$ for self-attention and $\{3,3\}$ for encoder-decoder attention. This results in a promising speed-up (see entry of uniform $\{\pi_i\}$). When we switch to joint learning of MT models and sharing policies, we obtain further improvements in both

Model	θ		BLEU	Δ_{BLEU}	Speed	Δ_{Speed}
	Self	Enc-Dec				
Baseline	N/A		27.52	0.00	1.03K	0.00%
SAN	uniform $\{\pi_i\}$		27.58	+0.06	1.43K	+38.83%
	0.30	0.40	26.89	-0.63	1.55K	+50.26%
	0.30	0.50	27.69	+0.17	1.44K	+39.81%
	0.40	0.40	26.96	-0.56	1.52K	+47.32%
	0.40	0.50	27.46	-0.06	1.40K	+36.33%

Table 3: BLEU scores (%) and translation speeds (token/sec) for different sharing policies. Self = self-attention. Enc-Dec = encoder-decoder attention

Model	Shared V	BLEU	Δ_{BLEU}	Speed	Δ_{Speed}
Baseline	-	27.52	0.00	1.03K	0.00%
SAN	no	27.49	-0.03	1.14K	+10.96%
	yes			1.27K	+22.91%

Table 4: BLEU scores (%) and translation speeds (token/sec) with/without a shared context (V) for encoder-decoder layers

BLEU and speed. More interestingly, we see that the system prefers a smaller θ for self-attention than the encoder-decoder counterpart. This is reasonable because the encoder-decoder attention captures the correspondence of two languages and needs more states in modeling than a single language. On the other hand, the BLEU improvements indicate that the MT system can benefit from simplified models. It gives a direction that we explore simpler models for better training of neural MT systems.

In encoder-decoder attention, we share the context V generated by the encoder for further speed-ups (see Figure 3(c)). It is therefore worth a study on how much this method can accelerate the system. Table 4 shows that sharing the context contributes half of the speed improvement. This agrees with our design that weight sharing is more beneficial to the decoder because attention is heavier on the decoder side. Another interesting question is whether SAN can improve the system on the encoder side. To seek an answer, we apply SAN to the encoder-side self-attention sub-layers and see small speed improvements (Table 5). This result confirms the previous report that the decoder occupies the inference time and the encoder is light [Zhang *et al.*, 2018].

Also, we plot the translation speed as functions of beam size and BLEU score. Figure 5 shows a consistent improvement under different beam settings. Moreover, SAN benefits more from larger beam sizes. For example, the speed-up of beam = 20 is larger than that of beam = 4 (1.48x vs. 1.40x). The Speed-vs-BLEU curves indicate a good ability of SAN in trading off between translation quality and speed.

In addition, we empirically compare SAN with other variants of the attention model, including AAN [Zhang *et al.*, 2018] and the model with no cache. Table 6 shows the attention cache plays an important role in fast inference. It leads to an 8-fold speed-up on top of the implementation where no cache is used. Also, SAN obtains a bigger speed improvement than AAN. This might be because AAN is used for self-attention only, while SAN is applicable to both self-attention and encoder-decoder attention. Finally, we combine AAN and

Model	BLEU	Δ_{BLEU}	Speed	Δ_{Speed}
Baseline	27.52	0.00	1.03K	0.00%
SAN	27.51	-0.01	1.05K	+1.94%

Table 5: BLEU scores (%) and translation speeds (token/sec) of the systems that use SAN on the encoder side

Model	BLEU	Δ_{BLEU}	Speed	Δ_{Speed}
Baseline	27.52	0.00	1.03K	0.00%
Baseline-Cache	27.52	0.00	0.12K	-88.65%
Baseline+AAN	27.51	-0.01	1.38K	+34.37%
Baseline+SAN	27.69	+0.17	1.44K	+39.81%
Baseline+AAN+SAN	27.19	-0.33	1.87K	+81.61%

Table 6: Comparison of different attention models

SAN in a new system where AAN is applied to self-attention and SAN is applied to encoder-decoder attention. It yields the best result which is 1.8 times faster than the baseline, and almost 16 times faster than the system without cache.

In training, we observe that systems tend to learn similar attention weights. Figure 6 plots the JS divergence between layer 4 and layers 5-6 at different training steps. The JS divergence curves go down significantly as the training proceeds. Adjacent layers show more similar weight distributions than distant layers. The fast convergence in JS divergence can speed up the iterative training. For example, for each training epoch (Figure 4), one can train the model for a shorter time, as the JS divergence among layers converges quickly. Thus, the system can find the optimal sharing policy more efficiently. In addition, we find that the training likelihood of SAN is higher than that of the baseline, but not significant.

5 Related Work

It has been observed that attention models are critical for state-of-the-art results on many MT tasks [Bahdanau *et al.*, 2015; Wu *et al.*, 2016; Vaswani *et al.*, 2017]. Several research groups have investigated attentive methods for different architectures of neural MT. The earliest is [Luong *et al.*, 2015b]. They introduced an additive attention model into MT systems based on recurrent neural networks (RNNs). More recently, multi-layer attention was successfully applied to convolutional neural MT systems [Gehring *et al.*, 2017] and Transformer systems [Vaswani *et al.*, 2017]. In particular, Transformer is popular due to its scalability on large-scale training and the good design of the architecture for implementation.

It is well-known that Transformer suffers from a high inference cost which makes it slower than the RNN-based counterpart. This partially due to the auto-regressive property of decoding, and partially due to the heavy use of dot-product attention where the expensive matrix multiplication is frequently used. Researchers have begun to explore solutions. For example, Gu et al. [2017] designed a non-autoregressive inference method for a Transformer-like system, which generated the entire target sequence at one time. This model is fast but is not easy to train.

In another line of research, Zhang et al. [2018] proposed the average attention network (AAN) and applied it to self-attention sub-layers on the decoder side with no cache. In this work, we study the issue on a strong baseline that has already used an attention cache for a reasonable inference speed. Also, our approach is straightforwardly applicable to systems with multi-layer attention. We improve both the self-attention and encoder-decoder attention components, which has not been studied in previous work.

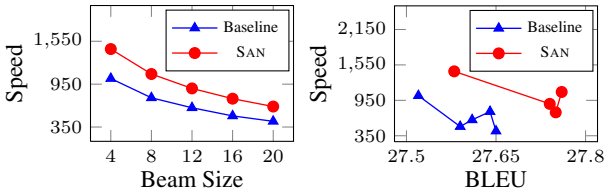


Figure 5: Translation speed (token/sec) vs beam size and BLEU (%)

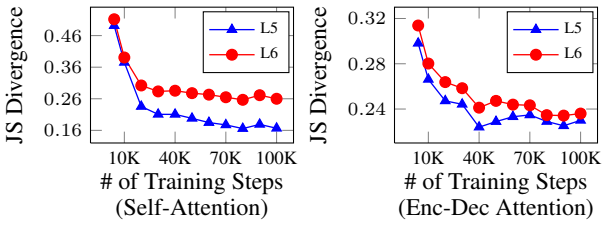


Figure 6: JS divergence vs number of training steps

In neural MT, fast inference methods have been investigated for years. These include vocabulary selection [L'Hostis *et al.*, 2016; Sankaran *et al.*, 2017], knowledge distillation [Hinton *et al.*, 2015; Kim and Rush, 2016], low-precision computation [Micikevicius *et al.*, 2017; Quinn and Ballesteros, 2018], recurrent stacked networks [Dabre and Fujita, 2019] and etc. Our method is orthogonal to them. Previous studies focus more on the reduction of model size and robust training, rather than fast inference. Here we study the issue in the context of speeding up attentive MT and confirm the effectiveness of this kind of models.

6 Conclusions

We have presented a shared attention network (SAN) for fast inference of Transformer. It shares attention weights among layers for both self-attention and encoder-decoder attention in a vertical manner. The policy of sharing can be jointly learned with the MT model, rather than being determined heuristically. Moreover, SAN reduces the memory footprint. Experiments on ten MT tasks show that SAN yields a speed-up of 1.3X over a strong baseline that has already used an attention cache. More interestingly, it is observed that the combination of SAN and AAN obtains a larger speed improvement. The system is 16X faster than the baseline with no cache.

Acknowledgments

This work was supported in part by the National Science Foundation of China (Nos. 61732005, 61876035 and 61432013) and the Fundamental Research Funds for the Central Universities (No. N181602013).

References

[Bahdanau *et al.*, 2015] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *In Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[Britz *et al.*, 2017] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906, 2017.

[Dabre and Fujita, 2019] Raj Dabre and Atsushi Fujita. Recurrent stacking of layers for compact neural machine translation models. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[Gehring *et al.*, 2017] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1243–1252, 2017.

[Gu *et al.*, 2017] Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. Non-autoregressive neural machine translation. *CoRR*, abs/1711.02281, 2017.

[Hinton *et al.*, 2015] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *Computer Science*, 14(7):38–39, 2015.

[Kim and Rush, 2016] Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1317–1327, 2016.

[Kingma and Ba, 2014] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computer Science*, 2014.

[L'Hostis *et al.*, 2016] Gurvan L'Hostis, David Grangier, and Michael Auli. Vocabulary selection strategies for neural machine translation. *CoRR*, abs/1610.00072, 2016.

[Lin, 1991] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Trans. Information Theory*, 37(1):145–151, 1991.

[Luong *et al.*, 2015a] Minh Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. 2015.

[Luong *et al.*, 2015b] Thang Luong, Hieu Pham, and D. Christopher Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics, 2015.

[Micikevicius *et al.*, 2017] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. *CoRR*, abs/1710.03740, 2017.

[Quinn and Ballesteros, 2018] Jerry Quinn and Miguel Ballesteros. Pieces of eight: 8-bit neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HTL 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 3 (Industry Papers)*, pages 114–120, 2018.

[Sankaran *et al.*, 2017] Baskaran Sankaran, Markus Freitag, and Yaser Al-Onaizan. Attention-based vocabulary selection for NMT decoding. *CoRR*, abs/1706.03824, 2017.

[Sutskever *et al.*, 2014] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[Szegedy *et al.*, 2016] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826, 2016.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

[Wu *et al.*, 2016] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[Xiao *et al.*, 2012] Tong Xiao, Jingbo Zhu, Hao Zhang, and Qiang Li. NiuTrans: An open source toolkit for phrase-based and syntax-based machine translation. In *Proceedings of the ACL 2012 System Demonstrations*, pages 19–24, Jeju Island, Korea, July 2012.

[Yang *et al.*, 2018] Zhen Yang, Wei Chen, Feng Wang, and Bo Xu. Unsupervised neural machine translation with weight sharing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 46–55. Association for Computational Linguistics, 2018.

[Zhang *et al.*, 2018] Biao Zhang, Deyi Xiong, and Jinsong Su. Accelerating neural transformer via an average attention network. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1789–1798. Association for Computational Linguistics, 2018.