Multi-Head Highly Parallelized LSTM Decoder for Neural Machine Translation

Hongfei Xu[1] **Qiuhui Liu**[2] **Josef van Genabith**[1] **Deyi Xiong**[3,4] **Meng Zhang**[5]
[1]DFKI and Saarland University, Informatics Campus, Saarland, Germany
[2]China Mobile Online Services, Henan, China
[3]Tianjin University, Tianjin, China
[4]Global Tone Communication Technology Co., Ltd.
[5]Huawei Noah's Ark Lab
{hfxunlp, liuqhano}@foxmail.com, josef.van_genabith@dfki.de,
dyxiong@tju.edu.cn, zhangmeng92@huawei.com

Abstract

One of the reasons Transformer translation models are popular is that self-attention networks for context modelling can be easily parallelized at sequence level. However, the computational complexity of a self-attention network is $O(n^2)$, increasing quadratically with sequence length. By contrast, the complexity of LSTM-based approaches is only $O(n)$. In practice, however, LSTMs are much slower to train than self-attention networks as they cannot be parallelized at sequence level: to model context, the current LSTM state relies on the full LSTM computation of the preceding state. This has to be computed n times for a sequence of length n. The linear transformations involved in the LSTM gate and state computations are the major cost factors in this. To enable sequence-level parallelization of LSTMs, we approximate full LSTM context modelling by computing hidden states and gates with the current input and a simple bag-of-words representation of the preceding tokens context. This allows us to compute each input step efficiently in parallel, avoiding the formerly costly sequential linear transformations. We then connect the outputs of each parallel step with computationally cheap element-wise computations. We call this the Highly Parallelized LSTM. To further constrain the number of LSTM parameters, we compute several small HPLSTMs in parallel like multi-head attention in the Transformer. The experiments show that our MHPLSTM decoder achieves significant BLEU improvements, while being even slightly faster than the self-attention network in training, and much faster than the standard LSTM.

1 Introduction

The Transformer translation model (Vaswani et al., 2017) has achieved great success and is used extensively in the NLP community. It achieves outstanding performance compared to previous RNN/CNN based translation models (Bahdanau et al., 2015; Gehring et al., 2017) while being much faster to train.

The Transformer can be trained efficiently due to the highly parallelized self-attention network. It enables sequence-level parallelization in context modelling, as all token representations can be computed in parallel, and linear transformations are only required to compute the sequence once. On the other hand, previous RNN-based methods process a sequence in a token-by-token manner, which means that they have to compute linear layers once for each token, i.e. n times if the number of tokens in the sequence is n.

However, the complexity of a self-attention network which compares each token with all the other tokens is $O(n^2)$, while for LSTM (Hochreiter and Schmidhuber, 1997) it is only $O(n)$. In practice, however, LSTM is slower than the self-attention network in training. This is mainly due to the fact that the computation of its current step relies on the computation output of the previous step, which prevents efficient parallelization over the sequence. As for the performance of using recurrent models in machine translation, Chen et al. (2018) shows that an LSTM-based decoder can further improve the performance over the Transformer.

In this paper, we investigate how we can efficiently parallelize all linear transformations of an LSTM at the sequence level, i.e. compute its linear transformations only once with a given input sequence. Given that linear transformations are implemented by matrix multiplication, compared to the other element-wise operations, we suggest that they take the largest part of the model's overall computation, and parallelizing the linear transformations at sequence level may significantly accelerate the training of LSTM-based models.

Our contributions are as follows:

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 273–282
August 1–6, 2021. ©2021 Association for Computational Linguistics

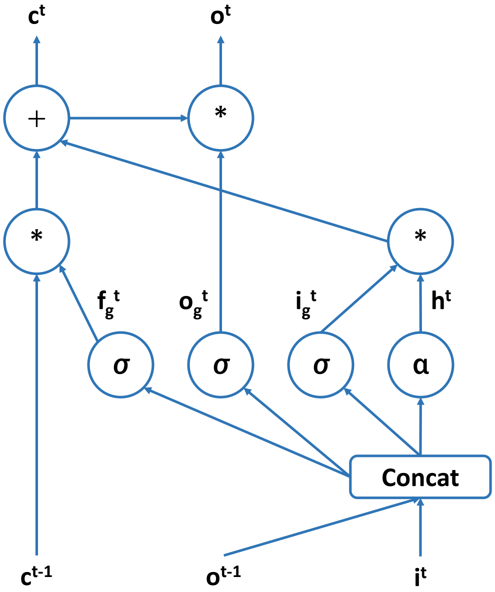

Figure 1: LSTM. Layer normalization is omitted for simplicity.

- We present the HPLSTM model, which computes LSTM gates and the hidden state with the current input embedding and a bag-of-words representation of preceding representations, rather than with the current input and the full LSTM output of the previous step, to enable efficient parallelization over the sequence and handling long sequences;
- We propose to divide a high-dimensional HPLSTM computation into several low-dimensional HPLSTM transformations, namely Multi-head HPLSTM, to constrain both the number of parameters and computation cost of the model;
- We empirically show that the MHPLSTM decoder can achieve improved performance over self-attention networks and recurrent approaches, while being even slightly faster in training, and significantly faster in decoding.

2 Preliminaries: LSTM

We design our HPLSTM based on the Layer Normalization (Ba et al., 2016) enhanced LSTM (LN-LSTM) presented by Chen et al. (2018) as illustrated in Figure 1, which achieves better performance than the Transformer when used in decoding.

For the computation of gates and the hidden state, the model concatenates the input i^t of the current step t to the output of the previous step o^{t-1}:

$$v^t = i^t | o^{t-1} \tag{1}$$

where "|" indicates concatenation, and v^t is the concatenated vector.

Next, it computes three gates (input gate i_g^t, forget gate f_g^t and output gate o_g^t) and the hidden representation h^t with v^t:

$$i_g^t = \sigma(\mathrm{LN}(W_i v^t + b_i)) \tag{2}$$

$$f_g^t = \sigma(\mathrm{LN}(W_f v^t + b_f)) \tag{3}$$

$$o_g^t = \sigma(\mathrm{LN}(W_o v^t + b_o)) \tag{4}$$

$$h^t = \alpha(\mathrm{LN}(W_h v^t + b_h)) \tag{5}$$

where W_i, W_f, W_o, W_h and b_i, b_f, b_o, b_h are weight and bias parameters, σ indicates the sigmoid activation function, α is the activation function for the hidden state computation, LN is the layer normalization.

Layer normalization (Ba et al., 2016) is computed as follows:

$$\mathrm{LN}_{Output} = \frac{\mathrm{LN}_{Input} - \mu}{\delta} * w_{\mathrm{LN}} + b_{\mathrm{LN}} \tag{6}$$

where LN_{Input} is the input, μ and δ stand for the mean and standard deviation of LN_{Input}, w_{LN} and b_{LN} are two vector parameters initialized by ones and zeros respectively.

After the computation of the hidden state, the cell c^t and the output of the LSTM unit o^t are computed as:

$$c^t = c^{t-1} * f_g^t + h^t * i_g^t \tag{7}$$

$$o^t = c^t * o_g^t \tag{8}$$

where $*$ indicates element-wise multiplication.

3 Our Approach

3.1 Highly Parallelized LSTM

Equation 1 shows that the computation of the hidden state and gates for step t requires the output of the step $t-1$. This prevents the LSTM from efficient parallelization at the sequence level: unless o^{t-1} is ready, we cannot compute o^t.

To enable the LSTM to compute o^t in parallel, we propose the HPLSTM, as shown in Figure 2.

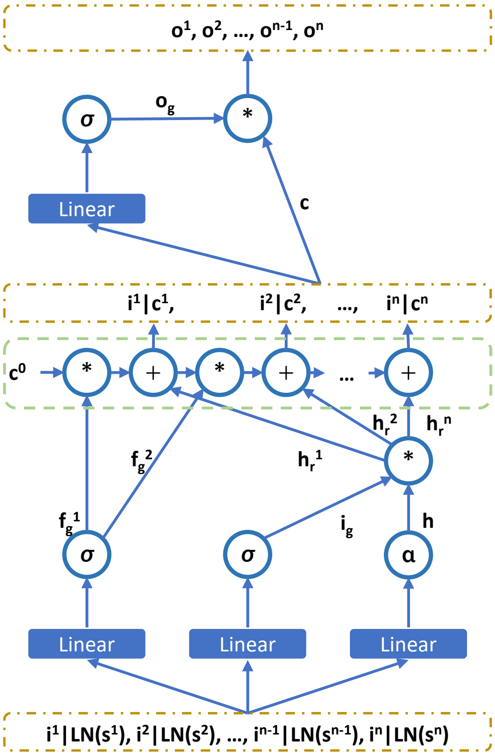
$o^1, o^2, \ldots, o^{n-1}, o^n$
σ
o_g
*
Linear
c
$i^1|c^1$, $i^2|c^2$, …, $i^n|c^n$
c^0
*
+
*
+
…
+
h_r^2
h_r^n
f_g^2
h_r^1
*
f_g^1
i_g
h
σ
σ
α
Linear
Linear
Linear
$i^1|LN(s^1), i^2|LN(s^2), \ldots, i^{n-1}|LN(s^{n-1}), i^n|LN(s^n)$

Figure 2: HPLSTM. All computations are parallelized at sequence level except for the green dashed block.

The HPLSTM uses a bag-of-words representation s^t of preceding tokens for the computation of gates and the hidden state:

$$s^t = \sum_{k=1}^{t-1} i^k \tag{9}$$

where s^1 is a zero vector. The bag-of-words representations s^t can be obtained efficiently via the cumulative sum operation.

Next, we concatenate the input i and the corresponding layer normalized bag-of-words representation $LN(s)$ for subsequent computing:

$$v = i|\mathrm{LN}(s) \tag{10}$$

the layer normalization is introduced to prevent potential explosions due to accumulation in Equation 9 to stabilize training.

Next, we compute the input gate, forget gate and the hidden state:

$$i_g = \sigma(\mathrm{LN}(W_i v + b_i)) \tag{11}$$

$$f_g = \sigma(\mathrm{LN}(W_f v + b_f)) \tag{12}$$

$$h = \alpha(\mathrm{LN}(W_h v + b_h)) \tag{13}$$

Since v is computed over the sequence before the computation of these gates and the hidden states, Equations 11, 12 and 13 are only required to be computed once for the whole sequence, enabling efficient sequence-level parallelization of high cost linear transformations, while in the original LSTM, they (Equations 2, 3 and 5) have to be computed one after the other as many times as the number of items in the sequence. However, the bag-of-words context representation s^t lacks a weighting mechanism compared to the previous step output o^{t-1} of the original LSTM, thus we also try to use a two-layer feed-forward network for the hidden state computation to alleviate potentially related drawbacks:

$$h = W_{h2}\alpha(\mathrm{LN}(W_{h1}v + b_{h1})) + b_{h2} \tag{14}$$

Then we update the hidden state h with the input gate i_g:

$$h_r = h * i_g \tag{15}$$

where h_r is the updated hidden state.

With h_r and f_g, we compute LSTM cells across the sequence:

$$c^t = c^{t-1} * f_g^t + h_r^t \tag{16}$$

Equation 16 preserves the step-by-step recurrence update of the LSTM cell and cannot be parallelized across the sequence, but it only contains element-wise multiplication-addition operations, which are light-weight and, compared to linear transformations, can be computed very fast on modern hardware.

Unlike the original LSTM which computes the output gate o_g based on the concatenated vector v^t (Equation 4), we compute the output gate with the newly produced cell state c and the input to the LSTM, as c is expected to have better quality than the bag-of-words representation.

$$o_g = \sigma(\mathrm{LN}(W_o i|c + b_o)) \tag{17}$$

Finally, we apply the output gate to the cell, and obtain the output of the HPLSTM layer.

$$o = c * o_g \tag{18}$$

Both Equation 17 (including the linear transformation for the computation of the output gate) and 18 can also be efficiently parallelized over the sequence.

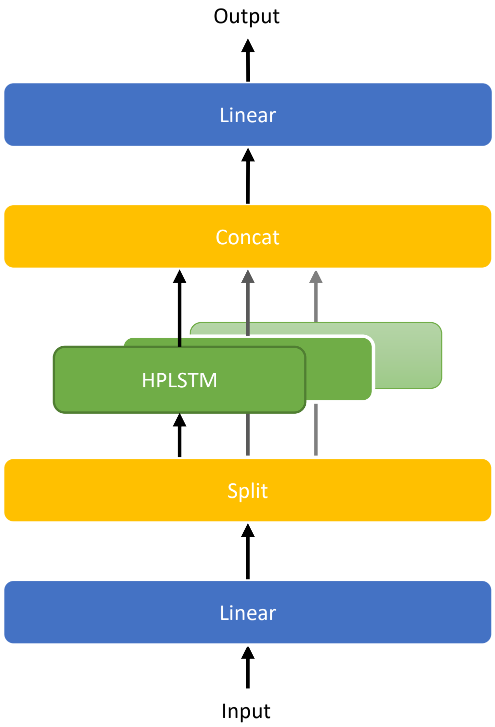

Figure 3: Multi-head HPLSTM

3.2 Multi-Head HPLSTM

Computing n smaller networks in parallel can remove the connections between hidden units across sub-networks, reducing both computation and the number of parameters.

Take for example a $512 \to 512$ transformation: using a densely fully-connected linear layer costs 8 times the number of parameters and computation compared to splitting the 512 dimension input into 8 folds and processing them with $8 \times 64 \to 64$ linear transformations correspondingly.

Since our HPLSTM involves more parameters and computation than a self-attention network with the same input size, to constrain the number of parameters, we compute n low-dimensional HPLSTMs in parallel. The resulting Multi-head HPLSTM (MHPLSTM) is illustrated in Figure 3.

Specifically, the MHPLSTM first transforms its input i into n different embedding spaces of HPLSTM transformations with a linear transformation and splits the transformed representation into n folds:

$$i_1|...|i_n{=}W_s i + b_s \tag{19}$$

Next, the kth input i_k is fed into the corresponding HPLSTM network HPLSTM_k, and the output o_k is obtained:

Models	En-De	En-Fr
Transformer Base	27.55	39.54
HPLSTM	**28.37**†	**40.31**†
Transformer Big	28.63	41.92
HPLSTM	**29.76**†	**42.84**†

Table 1: Results on WMT 14 En-De and En-Fr. † indicates $p < 0.01$ in the significance test.

$$o_k = \text{HPLSTM}_k(i_k) \tag{20}$$

In practice, the forward propagation of each HPLSTM is independent, thus for each HPLSTM Equation 20 is computed in parallel.

Finally, outputs of all individual HPLSTM networks are concatenated and transformed by another linear transformation as the output of the MHPLSTM layer o:

$$o = W_m(o_1|...|o_n) + b_m \tag{21}$$

4 Experiments

We replace the self-attention layers of the Transformer **decoder** with the MHPLSTM in our experiments.

4.1 Settings

To compare with Vaswani et al. (2017), we conducted our experiments on the WMT 14 English to German and English to French news translation tasks. The concatenation of newstest 2012 and newstest 2013 was used for validation and newstest 2014 as test set.

We applied joint Byte-Pair Encoding (BPE) (Sennrich et al., 2016) with $32k$ merging operations on all data sets. We only kept sentences with a maximum of 256 subword tokens for training. Training sets were randomly shuffled in each training epoch.

We followed Vaswani et al. (2017) for the experiment settings. The training steps for Transformer Base and Transformer Big were $100k$ and $300k$ respectively. We used a dropout of 0.1 for all experiments except for the Transformer Big setting on the En-De task which was 0.3. For the Transformer Base setting, the embedding dimension and the hidden dimension of the position-wise feed-forward neural network were 512 and 2048 respectively, the corresponding values for the Transformer Big

Model	BLEU	Para. (M)	Speed-Up	
			Train	Decode
Attention Based				
Transformer (Vaswani et al., 2017)	27.55	62.37	1.00	1.00
AAN (Zhang et al., 2018a)	27.63	74.97	1.04	1.52
Recurrent				
LN-LSTM (Chen et al., 2018)	27.96	68.69	0.45	1.47
ATR (Zhang et al., 2018b)	27.93	**59.23**	0.50	**1.69**
Ours				
MHPLSTM	**28.37**	62.80	**1.16**	**1.69**

Table 2: Comparison on WMT 14 En-De. For recurrent approaches, we replace the self-attention sub-layer of standard Transformer decoder layers with the corresponding module proposed in previous work.

setting were 1024 and 4096 respectively. The dimension of each head is 64, thus there were 8 and 16 heads for the base setting and the big setting respectively. We implemented our approaches based on the Neutron implementation (Xu and Liu, 2019) of the Transformer translation model. Parameters were initialized under the Lipschitz constraint (Xu et al., 2020c).

We used a beam size of 4 for decoding, and evaluated tokenized case-sensitive BLEU with the averaged model of the last 5 checkpoints for the Transformer Base setting and 20 checkpoints for the Transformer Big setting saved with an interval of 1500 training steps. We also conducted significance tests (Koehn, 2004).

4.2 Main Results

We first verify the performance by comparing our approach with the Transformer in both the base setting and the big setting. Results are shown in Table 1.

Table 1 shows that using an LSTM-based decoder can bring significant improvements over the self-attention decoder. Specifically, using MHPLSTM improves +0.82 and +0.77 BLEU on the En-De and En-Fr task respectively using the base setting, +1.13 and +0.92 correspondingly using the big setting. The fact that using an LSTM-based decoder can improve the translation quality is consistent with Chen et al. (2018), with MHPLSTM further improving over LN-LSTM (Table 2).

We also compare our approach with the Averaged Attention Network (AAN) decoder (Zhang et al., 2018a), LN-LSTM and the Addition-subtraction Twin-gated Recurrent (ATR) network (Zhang et al., 2018b) on the WMT 14 En-De task.

The AAN consists of an average layer that averages preceding embeddings, a feed-forward network to perform context-aware encoding based on the averaged context embedding, and a gating layer to enhance the expressiveness.

With a simple addition and subtraction operation, Zhang et al. (2018b) introduce a twin-gated mechanism to build input and forget gates which are highly correlated, and present a heavily simplified ATR which has the smallest number of weight matrices among units of all existing gated RNNs. Despite this simplification, the essential non-linearities and capability of modelling long-distance dependencies are preserved.

As LN-LSTM and ATR lead to the out-of-memory issue when handling long sentences, we follow Zhang et al. (2018b) to use sentences no longer than 80 subwords for their training, but we keep the batch size and training steps the same as the others for fairness. Their training without excluding these long sentences is slower than we reported. Results are shown in Table 2.

Table 2 shows that the MHPLSTM is not only the fastest in both training and decoding, but also leads to the best performance compared to baselines. Surprisingly, MHPLSTM even surpasses LN-LSTM. We conjecture potential reasons that MHPLSTM surpasses both self-attention and LN-LSTM might be:

- The self-attention network relies on absolute positional embedding for position encoding, which has its drawbacks (Shaw et al., 2018; Wang et al., 2019; Chen et al., 2019a; Wang et al., 2020), while LSTMs seem to have natural advantages in (relative) positional encod-

Approach	BLEU		Para. (M)	Speed-Up	
	dev	test		Train	Decode
Transformer	24.00	27.55	62.37	1.00	1.00
MHPLSTM	24.65	28.37	62.80	1.16	1.69
- FFN	24.08	27.67	50.21	1.49	1.91

Table 3: The effects of decoder FFN.

Hidden	Gates	BLEU	
		dev	test
√	×	24.65	28.37
√	√	24.71	28.38
×	×	24.23	27.92
×	√	24.36	27.97

Table 4: Using 2-layer FFN computation.

ing (Chen et al., 2019b).

- LSTMs lack a mechanism to directly connect distant words, which may lead to overlooking neighboring information, while the use of a bag-of-words representation (Equation 9) enables MHPLSTM to connect tokens directly regardless of the distance, thus MHPLSTM is able to leverage both local (Equation 16) and global patterns (Xu et al., 2019). (Please refer to Section 4.7 for empirical verification.)

- Compared to the self-attention network, the MHPLSTM computation is more complex.

- The computation for the LSTM hidden state (Equation 14) and output gate (Equation 17) in MHPLSTM is enhanced compared to the LN-LSTM.

4.3 Effect of FFN Layers

We conducted ablation studies on the WMT 14 En-De task.

Since the LSTM hidden state computation may take the role of the position-wise Feed-Forward Network (FFN) sub-layer of decoder layers, we first study removing the FFN sub-layer in decoder layers. Results are shown in Table 3.

Table 3 shows that removing the FFN layer of the MHPLSTM-based decoder can lead to further acceleration while performing competitively with the Transformer baseline with fewer parameters. However, it hampers MHPLSTM performance, thus we keep the feed-forward layer in the other experiments.

We also study the effects of using a 1-layer or a 2-layer neural network for the computation of the MHPLSTM hidden states (Equations 13 and 14) and gates (Equations 11 and 12). Results are shown in Table 4.

Table 4 shows that using a 2-layer neural network for the computation of hidden states is important for the performance, but the impact of using a 2-layer neural network for the gate computation is neglectable. Thus we only apply the 2-layer network for the computation of the LSTM hidden states in the other experiments.

4.4 Number of MHPLSTM Heads

We examined the effects of the impact of the number of MHPLSTM heads on performance and efficiency with the base setting (input dimension: 512). Results are shown in Table 5.

Table 5 shows that reducing the number of heads increases both parameters and time consumption with small performance gains compared to using 8 heads (with a dimension of 64 per head). Using 16 heads significantly hampers the performance with only a small reduction in the number of parameters and a slight acceleration. Thus we use a head dimension of 64 (8 heads for the base setting, 16 for the big setting) in our experiments, consistent with the Transformer.

4.5 MHPLSTM for Encoding

We tested the performance of using a bidirectional MHPLSTM for encoding. Results are shown in Table 6.

Table 6 shows that using MHPLSTM for encoding leads to a significant performance drop with more parameters: it even underperforms the baseline, while slowing down both training and decoding.

We conjecture that the self-attention network has advantages in encoding compared to the MHPLSTM: it can collect and process bi-directional

# Heads	BLEU		Para. (M)	Speed-Up	
	dev	test		Train	Decode
2	24.71	28.43	73.42	0.98	1.51
4	24.67	28.41	66.35	1.04	1.57
8	24.65	28.37	62.80	1.16	1.69
16	24.21	28.03	61.03	1.27	1.76

Table 5: The effects of the number of MHPLSTM heads.

Approach	BLEU		Para. (M)	Speed-Up	
	dev	test		Train	Decode
Transformer	24.00	27.55	62.37	1.00	1.00
MHPLSTM	24.65	28.37	62.80	1.16	1.69
+ Encoder	23.59	27.12	69.98	0.83	1.38

Table 6: MHPLSTM for encoding.

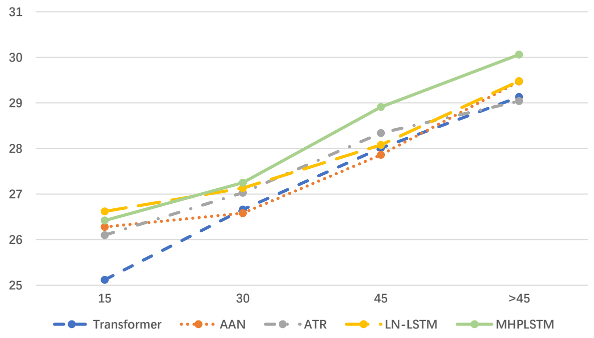

Figure 4: BLEU scores with respect to various input sentence length.

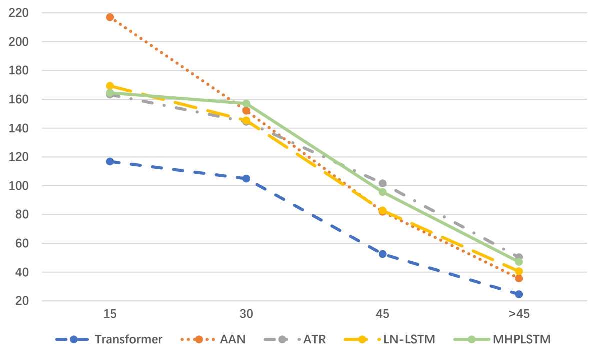

Figure 5: Decoding speed on a single GTX 1080Ti GPU with respect to various input sentence length. Y-axis: number of sentences / second. Beam size: 4.

context in one forward pass, while MHPLSTM has to compute 2 forward passes, one for the forward direction, another one for the reverse direction. For each direction, relevant context is processed separately in the recurrent models.

4.6 Length Analysis

To analyze the effects of MHPLSTM on performance with increasing input length, we conducted a length analysis on the news test set of the WMT 14 En-De task. Following Bahdanau et al. (2015); Tu et al. (2016); Xu et al. (2020b), we grouped sentences of similar lengths together and computed BLEU scores of the MHPLSTM and our baselines for each group. BLEU score results and decoding speed-up of each group are shown in Figure 4 and 5 respectively.

Figure 4 shows that MHPLSTM surpasses the other approaches in most length groups, and improvements of using an MHPLSTM based-decoder are more significant for long sentences than short sentences.

Figure 5 shows that all recurrent-based approaches are faster than the self-attention decoder in all length groups, and MHPLSTM achieves comparable decoding speed as LSTM and ATR. Even though the decoding speed of all approaches decreases very fast with increasing sentence length, the acceleration of MHPLSTM is more significant with long sentences (1.91 times faster than Transformer for sentences longer than 45) than with short sentences (1.41 times faster than Transformer for sentences no longer than 15).

4.7 Local / Global Pattern Learning Analysis

We compare the ability of the MHPLSTM and baselines in capturing dependencies of various distances with the linguistically-informed verb-subject agreement analysis on the *Lingeval97* dataset (Sennrich,

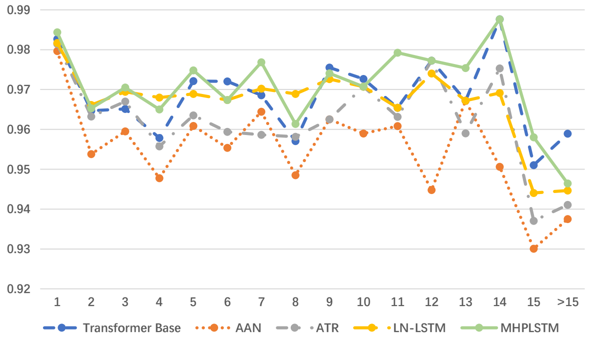

Figure 6: Subject-verb agreement analysis. X-axis and y-axis represent subject-verb distance in words and accuracy respectively.

2017).

In German, subjects and verbs must agree with one another in grammatical number and person. In *Lingeval97*, each contrastive translation pair consists of a correct reference translation, and a contrastive example that has been minimally modified to introduce one translation error. The accuracy of a model is the number of times it assigns a higher score to the reference translation than to the contrastive one, relative to the total number of predictions. Results are shown in Figure 6.

Figure 6 shows that the MHPLSTM outperforms baselines in almost all cases. For distances longer than 15, the self-attention network still performs best, indicating its strong ability in long-distance relation learning, but the MHPLSTM still surpasses the other recurrent approaches.

5 Related Work

Sequence-to-sequence neural machine translation models started with recurrent models (Sutskever et al., 2014; Bahdanau et al., 2015; Cho et al., 2014). But recurrent models cannot be parallelized at the sequence level. Convolutional models (Gehring et al., 2017; Wu et al., 2019) and the Transformer (Vaswani et al., 2017) have been proposed.

Due to the $O(n^2)$ self-attention network complexity, which slows down decoding, Zhang et al. (2018a) presented the average attention network to accelerate decoding. Even though LSTMs cannot be parallelized at the sequence level, its complexity is $O(n)$, and Chen et al. (2018) shows that using the layer normalization enhanced LSTM-based decoder can bring improvements in translation quality and accelerate decoding.

LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014) are the most popular recurrent models. To accelerate RNN models, Zhang et al. (2018b) propose a heavily simplified ATR network to have the smallest number of weight matrices among units of all existing gated RNNs.

Peter et al. (2016) investigate exponentially decaying bag-of-words input features for feed-forward NMT models. In addition to sequence-level parallelization, asynchronous optimization (Heigold et al., 2014) and data parallelization with a larger batch size (Ott et al., 2018; Chen et al., 2018; Xu et al., 2020a) can also accelerate training.

6 Conclusion

In this paper, we observe that the sequence-level parallelization issue of LSTM is due to the fact that its computation of gates and hidden states of the current step relies on the computation result of the preceding step, and linear transformations have to be propagated the same number of times as the sequence length. To improve the sequence-level parallelization of the LSTM, we propose to remove the dependency of the current step LSTM computation on the result of the previous step by computing hidden states and gates with the current input embedding and a bag-of-words representation of preceding tokens, and present the Highly Parallelized LSTM. To constrain the number of LSTM parameters, we compute several small HPLSTMs in parallel like multi-head self-attention.

In our experiments, we empirically show that the MHPLSTM model achieves better performance than self-attention networks, while being even slightly faster in training, and much faster in decoding, than the self-attention Transformer decoder.

Acknowledgments

We thank anonymous reviewers for their insightful comments. Hongfei Xu acknowledges the support of China Scholarship Council ([2018]3101, 201807040056). Josef van Genabith is supported by the German Federal Ministry of Education and Research (BMBF) under funding code 01IW20010 (CORA4NLP). Deyi Xiong is partially supported by the joint research center between GTCOM and Tianjin University and the Royal Society (London) (NAF\R1\180122). Meng Zhang is partially supported by MindSpore,[1] which is a new deep learning computing framework.

[1] https://www.mindspore.cn/.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kehai Chen, Rui Wang, Masao Utiyama, and Eiichiro Sumita. 2019a. Neural machine translation with reordering embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1787–1799, Florence, Italy. Association for Computational Linguistics.

Kehai Chen, Rui Wang, Masao Utiyama, and Eiichiro Sumita. 2019b. Recurrent positional embedding for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1361–1367, Hong Kong, China. Association for Computational Linguistics.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252, International Convention Centre, Sydney, Australia. PMLR.

G. Heigold, E. McDermott, V. Vanhoucke, A. Senior, and M. Bacchiani. 2014. Asynchronous stochastic optimization for sequence training of deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5587–5591.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395, Barcelona, Spain. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9, Brussels, Belgium. Association for Computational Linguistics.

Jan-Thorsten Peter, Weiyue Wang, and Hermann Ney. 2016. Exponentially decaying bag-of-words input features for feed-forward neural network in statistical machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 293–298, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich. 2017. How grammatical is character-level neural machine translation? assessing MT quality with contrastive translation pairs. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 376–382, Valencia, Spain. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 27, pages 3104–3112. Curran Associates, Inc.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–85, Berlin, Germany. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, and Jakob Grue Simonsen. 2020. Encoding word order in complex embeddings. In *International Conference on Learning Representations*.

Xing Wang, Zhaopeng Tu, Longyue Wang, and Shuming Shi. 2019. Self-attention with structural position representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1403–1409, Hong Kong, China. Association for Computational Linguistics.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*.

Hongfei Xu, Josef van Genabith, Deyi Xiong, and Qiuhui Liu. 2020a. Dynamically adjusting transformer batch size by monitoring gradient direction change. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3519–3524, Online. Association for Computational Linguistics.

Hongfei Xu, Josef van Genabith, Deyi Xiong, Qiuhui Liu, and Jingyi Zhang. 2020b. Learning source phrase representations for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 386–396, Online. Association for Computational Linguistics.

Hongfei Xu and Qiuhui Liu. 2019. Neutron: An Implementation of the Transformer Translation Model and its Variants. *arXiv preprint arXiv:1903.07402*.

Hongfei Xu, Qiuhui Liu, Josef van Genabith, Deyi Xiong, and Jingyi Zhang. 2020c. Lipschitz constrained parameter initialization for deep transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 397–402, Online. Association for Computational Linguistics.

Mingzhou Xu, Derek F. Wong, Baosong Yang, Yue Zhang, and Lidia S. Chao. 2019. Leveraging local and global patterns for self-attention networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3069–3075, Florence, Italy. Association for Computational Linguistics.

Biao Zhang, Deyi Xiong, and Jinsong Su. 2018a. 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, Melbourne, Australia. Association for Computational Linguistics.

Biao Zhang, Deyi Xiong, Jinsong Su, Qian Lin, and Huiji Zhang. 2018b. Simplifying neural machine translation with addition-subtraction twin-gated recurrent networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4273–4283, Brussels, Belgium. Association for Computational Linguistics.