When Attention Meets Fast Recurrence: Training Language Models with Reduced Compute

Tao Lei
ASAPP, Inc.
taoleics@gmail.com

Abstract

Large language models have become increasingly difficult to train because of the growing computation time and cost. In this work, we present SRU++, a highly-efficient architecture that combines fast recurrence and attention for sequence modeling. SRU++ exhibits strong modeling capacity and training efficiency. On standard language modeling tasks such as ENWIK8, WIKI-103 and BILLION WORD datasets, our model obtains better bits-per-character and perplexity while using 3x-10x less training cost compared to top-performing Transformer models. For instance, our model achieves a state-of-the-art result on the ENWIK8 dataset using 1.6 days of training on an 8-GPU machine. We further demonstrate that SRU++ requires minimal attention for near state-of-the-art performance. Our results suggest jointly leveraging fast recurrence with little attention as a promising direction for accelerating model training and inference.[1]

1 Introduction

Many recent advances in language modeling have come from leveraging ever larger datasets and model architectures. As a result, the associated computation cost for developing such models have grown enormously, requiring hundreds of GPU hours or days per experiment, and raising concerns about the environmental sustainability of current research (Schwartz et al., 2020). As a consequence, it has become imperative to build *computationally efficient* models that retain top modeling power while reducing computational costs.

The Transformer architecture (Vaswani et al., 2017) was proposed to accelerate model training and has become the predominant architecture in NLP. Specifically, it is built entirely upon self-attention and avoids the use of recurrence to enable strong parallelization. While this change has led to many empirical success and improved computational efficiency, we are interested in revisiting the architectural question: **Is attention all we need for modeling?**

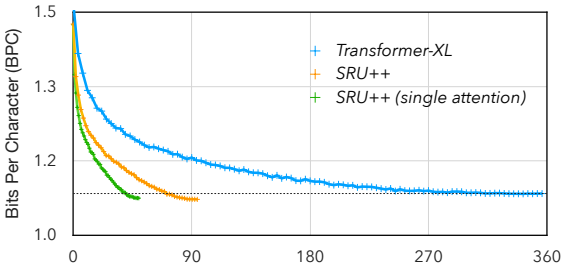


Figure 1: Bits-per-character on ENWIK8 dev set vs. GPU hours used for training. SRU++ obtains better BPC by using 1/8 of the resources. We compare with Transformer-XL as it is one of the strongest models on the datasets tested. Models are trained with single precision and comparable training settings.

The attention mechanism permits learning dependencies between any parts of the input, making it an extremely powerful neural component in many machine learning applications (Bahdanau et al., 2015; Lin et al., 2017). We hypothesize that this advantage can still be complemented with other computation that is directly designed for sequential modeling. Indeed, several recent works have studied and confirmed the same hypothesis by leveraging recurrence in conjunction with attention. For example, Merity (2019) demonstrates that single-headed attention LSTMs can produce results competitive to Transformer models in language modeling. Other work have incorporated RNNs into Transformer, and obtain better results in machine translation (Lei et al., 2018; Hao et al., 2019) and language understanding benchmarks (Huang et al., 2020). These results highlight one possibility – we could build more efficient models by combining attention and fast recurrent networks (Bradbury et al., 2017; Zhang and Sennrich, 2019).

[1]Our code, experimental setup and models are available at https://github.com/asappresearch/sru.

In this work, we validate this idea and present a self-attentive recurrent unit that achieves strong computational efficiency. Our work builds upon the SRU (Lei et al., 2018), a highly parallelizable RNN implementation that has been shown effective in language and speech applications (Park et al., 2018; Kim et al., 2019; Hsu et al., 2020; Shangguan et al., 2019). We incorporate attention into the SRU by simply replacing the linear transformation of input with a self-attention component. The proposed architecture, called SRU++, enjoys enhanced modeling capacity and remains equally parallelizable. Figure 1 compares its performance with the Transformer-XL model (Dai et al., 2019) on the ENWIK8 dataset. SRU++ achieves better results while using a fraction of the training resources needed by the baseline.

We evaluate SRU++ on standard language modeling benchmarks including the ENWIK8, WIKI-103 and BILLION WORD datasets. SRU++ consistently outperforms various Transformer models on these datasets, delivering better or on par results while using 3x-10x less computation. Our model do not use positional encoding, multi-head attention and other techniques useful to Transformer models. Furthermore, we demonstrate that a couple of attention layers are sufficient for SRU++ to obtain near state-of-the-art performance. These changes not only highlight the effectiveness of recurrence but also enable strong computation reduction in training and inference. Finally, we also showcase the effectiveness of SRU++ on the IWSLT'14 De→En translation task, and open source our implementation in Pytorch to facilitate future research.

2 Background: SRU

We first describe the Simple Recurrent Unit (SRU) in this section. A single layer of SRU involves the following computation:

$$\begin{aligned}
\mathbf{f}[t] &= \sigma\left(\mathbf{W}\mathbf{x}[t] + \mathbf{v} \odot \mathbf{c}[t\text{-}1] + \mathbf{b}\right) \\
\mathbf{r}[t] &= \sigma\left(\mathbf{W}'\mathbf{x}[t] + \mathbf{v}' \odot \mathbf{c}[t\text{-}1] + \mathbf{b}'\right) \\
\mathbf{c}[t] &= \mathbf{f}[t] \odot \mathbf{c}[t\text{-}1] + (1 - \mathbf{f}[t]) \odot (\mathbf{W}''\mathbf{x}[t]) \\
\mathbf{h}[t] &= \mathbf{r}[t] \odot \mathbf{c}[t] + (1 - \mathbf{r}[t]) \odot \mathbf{x}[t]
\end{aligned}$$

where $\odot$ is the element-wise multiplication, $\mathbf{W}$, $\mathbf{W}'$ and $\mathbf{W}''$ are parameter matrices and $\mathbf{v}$, $\mathbf{v}'$, $\mathbf{b}$ and $\mathbf{b}'$ are parameter vectors to be learnt during training. The SRU architecture consists of a light recurrence component which successively computes the hidden states $\mathbf{c}[t]$ by reading the input vector $\mathbf{x}[t]$ for each step t. The computation resembles other gated recurrent networks such as LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014). Specifically, the state vector $\mathbf{c}[t]$ is a weighted average between the previous state $\mathbf{c}[t\text{-}1]$ and a linear transformation of the input $\mathbf{W}''\mathbf{x}[t]$. The weighted aggregation is controlled by a forget gate $\mathbf{f}[t]$ which is a sigmoid function over the current input and hidden state. Once the internal state $\mathbf{c}[t]$ is produced, SRU uses a highway network to introduce a skip connection and compute the final output state $\mathbf{h}[t]$. Similarly, the information flow in the highway network is controlled by a reset gate $\mathbf{r}[t]$.

Two important code-level optimizations are performed to enhance the parallelism and speed of SRU. First, given the input sequence $\mathbf{X} = \{\mathbf{x}[1], \cdots, \mathbf{x}[L]\}$ where each $\mathbf{x}[t] \in \mathbb{R}^d$ is a d-dimensional vector, SRU combines the three matrix multiplications across all time steps as a single multiplication. This significantly improves the computation intensity (e.g. GPU utilization). Specifically, the batched multiplication is a linear projection of the input tensor $\mathbf{X} \in \mathbb{R}^{L \times d}$:

$$\mathbf{U}^\top = \begin{pmatrix} \mathbf{W} \\ \mathbf{W}' \\ \mathbf{W}'' \end{pmatrix} \mathbf{X}^\top , \qquad (1)$$

where $\mathbf{U} \in \mathbb{R}^{L \times 3 \times d}$ is the output tensor, L is the sequence length and d is the hidden state size.

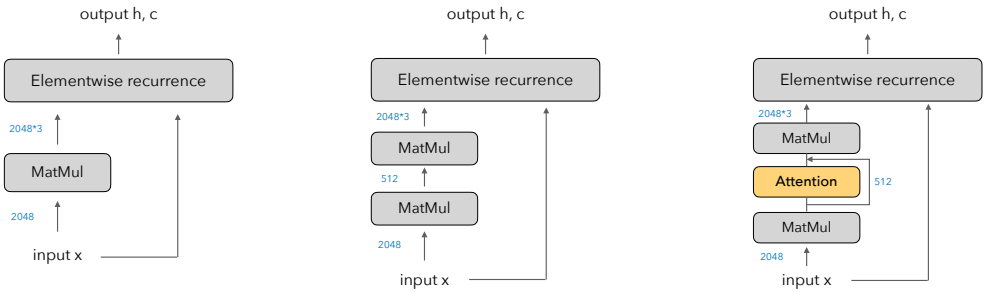
The second optimization performs all element-wise operations in an efficient way. This involves

$$\mathbf{f}[t] = \sigma(\mathbf{U}[t, 0] + \mathbf{v} \odot \mathbf{c}[t\text{-}1] + \mathbf{b}) \qquad (2)$$
$$\mathbf{r}[t] = \sigma(\mathbf{U}[t, 1] + \mathbf{v}' \odot \mathbf{c}[t\text{-}1] + \mathbf{b}') \qquad (3)$$
$$\mathbf{c}[t] = \mathbf{f}[t] \odot \mathbf{c}[t\text{-}1] + (1 - \mathbf{f}[t]) \odot \mathbf{U}[t, 2] \qquad (4)$$
$$\mathbf{h}[t] = \mathbf{r}[t] \odot \mathbf{c}[t] + (1 - \mathbf{r}[t]) \odot \mathbf{x}[t]. \qquad (5)$$

Similar to other built-in operations such as attention and cuDNN LSTM (Appleyard et al., 2016), SRU implements all these operations as a single CUDA kernel to accelerate computation. Note that each dimension of the hidden vectors is independent once $\mathbf{U}$ is computed. The computation can run in parallel across each hidden dimension (and each input sequence given a mini-batch of multiple sequences).

3 SRU++

The key modification of SRU++ is to incorporate more expressive non-linear operations into the re-



(a) SRU (b) SRU w/ projection trick (c) SRU++ w/ attention

Figure 2: An illustration of SRU and SRU++ networks: (a) the original SRU, (b) the SRU variant with projection to reduce the number of parameters, experimented in Lei et al. (2018) and (c) SRU++ proposed in this work. Numbers indicate the dimension of intermediate inputs/outputs given hidden size $d = 2048$ and attention size $d' = 512$.

current network. Note that the computation of $\mathbf{U}$ (Equation 1) is a linear transformation of the input sequence $\mathbf{X}$. We can replace this linear transformation with self-attention operation to enhance modeling capacity.

Specifically, given the input sequence represented as a matrix $\mathbf{X} \in \mathbb{R}^{L \times d}$, the attention component computes the query, key and value representations using the following multiplications,

$$\mathbf{Q} = \mathbf{W}^q \mathbf{X}^\top$$
$$\mathbf{K} = \mathbf{W}^k \mathbf{Q}$$
$$\mathbf{V} = \mathbf{W}^v \mathbf{Q}$$

where $\mathbf{W}^q \in \mathbb{R}^{d' \times d}$, $\mathbf{W}^k, \mathbf{W}^v \in \mathbb{R}^{d' \times d'}$ are model parameters. d' is the attention dimension that is typically much smaller than d. Note that the keys $\mathbf{K}$ and values $\mathbf{V}$ are computed using $\mathbf{Q}$ instead of $\mathbf{X}$ such that the weight matrices $\mathbf{W}^k$ and $\mathbf{W}^v$ are significantly smaller. We also tested another variant in which we first project $\mathbf{X}' = \mathbf{W}\mathbf{X}^\top$ into the lower dimension d', and then apply three independent d'-by-d' matrix multiplications over $\mathbf{X}'$ to obtain the query, key and value representations. This variant achieves similar results.

Next, we compute a weighted average output $\mathbf{A} \in \mathbb{R}^{d' \times L}$ using the scaled dot-product attention introduced in Vaswani et al. (2017),

$$\mathbf{A}^\top = \text{softmax}\left(\frac{\mathbf{Q}^\top \mathbf{K}}{\sqrt{d'}}\right) \mathbf{V}^\top.$$

The final output $\mathbf{U}$ required by the elementwise recurrence is obtained by another linear projection,

$$\mathbf{U}^\top = \mathbf{W}^o \left(\mathbf{Q} + \alpha \cdot \mathbf{A}\right).$$

where $\alpha \in \mathbb{R}$ is a learned scalar and $\mathbf{W}_o \in \mathbb{R}^{3d \times d'}$ is a parameter matrix. $\mathbf{Q} + \alpha \cdot \mathbf{A}$ is a residual connection which improves gradient propagation and stabilizes training. We initialize α to zero and as a result,

$$\mathbf{U}^\top = \mathbf{W}^o \mathbf{Q} = \left(\mathbf{W}^o \mathbf{W}^q\right) \mathbf{X}^\top$$

initially falls back to a linear transformation of the input $\mathbf{X}$ skipping the attention transformation. Intuitively, skipping attention encourages leveraging recurrence to capture sequential patterns during early stage of training. As $|\alpha|$ grows, the attention mechanism can learn long-range dependencies for the model. In addition, $\mathbf{W}^o\mathbf{W}^q$ can be interpreted as applying a matrix factorization trick with a small inner dimension $d' < d$, reducing the total number of parameters. Figure 2 (a)-(c) compares the differences of SRU, SRU with this factorization trick (but without attention), and SRU++ proposed in this section.

The last modification is adding layer normalization (Ba et al., 2016) to each SRU++ layer. In our implementation, we apply normalization after the attention operation and before the matrix multiplication with $\mathbf{W}^o$,

$$\mathbf{U}^\top = \mathbf{W}^o \text{ layernorm}(\mathbf{Q} + \alpha \cdot \mathbf{A}).$$

This implementation is post-layer normalization in which the normalization is added after the residual connection. Alternatively, pre-layer normalization (Xiong et al., 2020) only applies to the non-linear transformation. While pre-normalization tends to be less sensitive to different learning rates, we use post-normalization for better results following the observations in Liu et al. (2020b). We analyze the effectiveness of layer normalization in Appendix A.2.

Model	Batch size $B \times M$	BPC ↓
Trans-XL	24×512	1.06
SRU++	24×512	1.03
SRU++	16×768	1.02

Table 1: Test BPC of SRU++ and Transformer-XL on ENWIK8 dataset. We train SRU++ using the same setting as Transformer-XL base model. Numbers are smaller the better. B is the number of sequence. M is the unroll size (and additional context size).

Model	Param	BPC ↓	GPU hrs ↓
Trans-XL	41M	1.06	356
SHA-LSTM	54M	1.07	28†
$k = 1$		1.022	37†
$k = 2$		1.025	29†
$k = 5$	42M	1.032	24†
$k = 10$		1.033	22†
No attention		1.190	20†

Table 2: Results of SRU++ on ENWIK8 by enabling attention every k layers. We adjust the hidden size so the number of parameters are comparable. † indicates mixed precision training.

4 Experimental setup

Datasets We evaluate our model on four standard NLP benchmarks.

- ENWIK8 (Hutter, 2006) is a character-level language modeling dataset consisting of 100M tokens taken from Wikipedia. The vocabulary size of this dataset about 200. We use the standard 90M/5M/5M splits as the training, dev and test sets, and report bits-per-character (BPC) as the evaluation metric.

- WIKI-103 (Merity et al., 2017) is a word-level language modeling dataset. The training data contains 100M tokens extracted from Wikipedia articles. Following prior work, we use a vocabulary of 260K tokens, and adaptive embedding and softmax layers (Grave et al., 2017; Baevski and Auli, 2019).

- BILLION WORD (Chelba et al., 2013) is one of the largest language modeling datasets containing 768M tokens for training. Unlike WIKI-103 in which sentences in the same article are treated as consecutive inputs to model long context, the sentences in BILLION WORD are randomly shuffled. Following Baevski and Auli (2019), we use a vocabulary of 800K tokens, adaptive embedding and softmax layers.

- IWSLT'14 De→En is a low-resource machine translation dataset consists of 170K translation pairs. We showcase SRU++ can be applied to other tasks such as translation. We follow the same setup of Lin et al. (2020) and other previous work. The dataset uses a shared vocabulary of 14K BPE tokens.

Models All our language models are constructed with a word embedding layer, multiple layers of SRU++ and an output linear layer followed by softmax operation. We use single-head attention in each layer and 10 SRU++ layers for all our models. We use the same dropout probability for all layers and tune this value according to the model size and the results on the dev set. By default, we set the hidden dimension $d : d' = 4 : 1$. We report additional analysis and tune this ratio for best results in Section 5 and Appendix A.

For simplicity, SRU++ does not use recent techniques that are shown useful to Transformer such as multi-head attention, compressed memory (Rae et al., 2020), relative position (Shaw et al., 2018; Press et al., 2021), nearest-neighbor interpolation (Khandelwal et al., 2020) and attention variants to handle very long context (Sukhbaatar et al., 2019a; Roy et al., 2021).

We compare with previous Transformer models that incorporate one or several these techniques. However, we do not compare with results that use additional data or dynamic evaluation (Graves, 2013; Krause et al., 2018), for a fair comparison between all models.

Optimization We use RAdam (Liu et al., 2020a) with the default β values as our optimizer. RAdam is a variant of Adam optimizer (Kingma and Ba, 2014) that is reported less sensitive to the choice of learning rate and warmup steps while achieving similar results at the end. We use a fixed weight decay of 0.1 and an initial learning rate of 0.0003 in our experiments. These values are selected based on ENWIK8 dev set and used for other tasks. See Appendix A.3 for more details. We use a cosine learning rate schedule following Dai et al. (2019). We do not change the initial learning rate unless otherwise specified. See Appendix B for the detailed training configuration of each model.

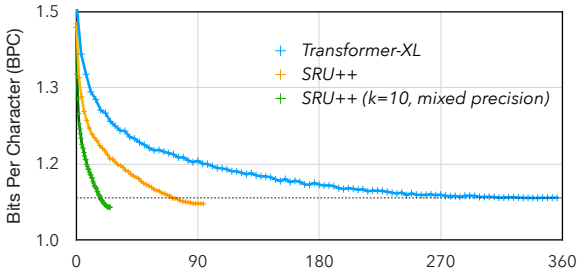


Figure 3: Dev BPC vs. total GPU hours used on ENWIK8 for each model. Using automatic mixed precision (amp) and only one attention sub-layer achieves 16x reduction. To compute the dev BPC, the maximum attention length is the same as the unroll size M during training.

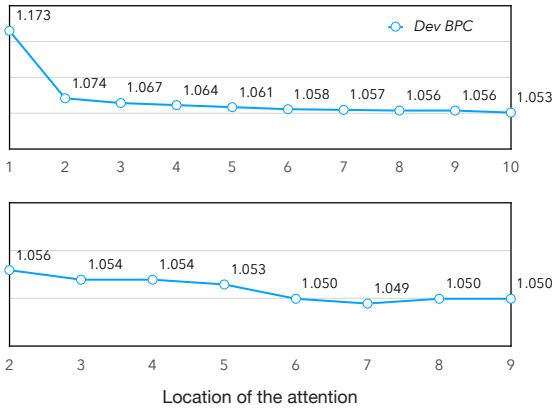


Figure 4: Analyzing where to apply attention. We enable only one attention layer (top figure) or two (bottom figure) in the SRU++ model. For the latter, we always apply attention in the last layer and move the location of the other. X-axis is the layer index. The layer closest to the input embedding layer has index 1.

Each training batch contains B sequences (i.e. the batch size) and M consecutive tokens for each sequence (i.e. the unroll size), which gives an effective size of $B \times M$ tokens per batch. Following standard practice, the previous training batch is provided as additional context for attention, which results in a maximum attention length of $2 \times M$. For ENWIK8 and WIKI-103 datasets, the training data is partitioned into B chunks by concatenating articles and ignoring the boundaries between articles. For BILLION WORD dataset, we follow Dai et al. (2019) and concatenate sentences to create the training batches. Sentences are randomly shuffled and separated by a special token <s> indicating sentence boundaries.

5 Results

Does recurrence improve upon attention-only model? We first conduct a comparison with the Transformer-XL model (Dai et al., 2019) on ENWIK8 dataset[2]. Their base model consists of 41M parameters and 12 Transformer layers. Following the official instructions, we reproduced the reported test BPC of 1.06 by training with 4 Nvidia 2080 Ti GPUs. The training took about 4 days or a total of 360 GPU hours equivalently.

We train a 10-layer SRU++ model with 42M parameters. For a fair comparison, we use the same hyperparameter setting including the effective batch size, attention context length, learning rate and the number of training iterations as the Transformer-XL base model. Notably, our base model can be trained using 2 GPUs due to less GPU memory usage. After training, we set the attention context length to 2048 for testing, similarly to the Transformer-XL baseline. Table 1 presents the results. Our model achieves a test BPC of 1.03, outperforming the baseline by a large margin. This result suggests that combining recurrence and attention can greatly outperform an attention-only model. We obtain a BPC of 1.02 by extending the attention context length from 512 to 768, while keeping the number of tokens per batch the same.

How much attention is needed? Merity (2019) demonstrated that using a single attention layer with LSTM retains most of the modeling capacity compared to using multiple attention layers. We conduct a similar analysis to understand how much attention is needed in SRU++. To do so, we only enable attention every k layers. The layers without attention become the variant with dimension projection illustrated in Figure 2 (b). Note that $k = 1$ gives the default SRU++ model with attention in every layer, and $k = 10$ means only the last layer has attention in a 10-layer model.

Table 2 presents the results by varying k. Our base model is the same 10-layer SRU++ model in Table 1. We see that using 50% less attention ($k = 2$) achieves almost no increase in test BPC. Moreover, using only a single attention module ($k = 10$) leads to a marginal loss of 0.01 BPC but reduces the training time by 40%. Our results still outperform Transformer-XL model and single-headed attention LSTM (Merity, 2019) greatly by 0.03 BPC. Figure 3 showcases the training efficiency of our model. SRU++ is

[2] https://github.com/kimiyoung/transformer-xl/tree/master/pytorch

Model	Parameters ↓	Test BPC ↓	GPU days ↓
Longformer 30L (Beltagy et al., 2020)	102M	0.99	104†
All-attention network 36L (Sukhbaatar et al., 2019b)	114M	0.98	64
Transformer-XL 24L (Dai et al., 2019)	277M	0.99	-
◦ Compressive memory (Rae et al., 2020)	-	0.97	-
Feedback Transformer (Fan et al., 2020)	77M	0.96	-
SRU++ Base	108M	0.97	6†
◦ only 2 attention layers ($k = 5$)	98M	0.98	4†
SRU++ Large	191M	0.96	12†
◦ $d = 8\,d'$	195M	0.95	13†

Table 3: Comparison with top-performing models on ENWIK8 dataset. We include the training cost (measured by the number of GPUs used $\times$ the number of days) if it is reported in the previous work. Our results are obtained using an AWS p3dn instance with 8 V100 GPUs. The reported training time of all-attention network is based on V100 GPUs while the training time of Longformer is based on RTX8000 GPUs (which is about 90% speed of V100). † indicates mixed precision training.

Ratio	Dimensions d, d'		Dev BPC ↓
4	3072	768	0.997
6	3840	640	0.992
8	4480	560	0.991
10	5040	504	0.992

Table 4: Dev BPC on ENWIK8 by changing the ratio $d : d'$ in the SRU++ model while fixing the number of parameters to 108M.

5x faster to reach the dev BPC obtained by the Transformer-XL model. Furthermore, using automatic mixed precision training and a single attention layer ($k = 10$) achieves 16x reduction on training cost.

Where to use attention? Next, we analyze if the location of attention in SRU++ makes a non-trivial difference. Figure 4 (top) compares the results by enabling attention in only one of the SRU++ layers. Applying attention in the first bottom layer achieves significantly worse result. We believe this is due to the lack of positional information for attention, since SRU++ does not use positional encoding. Enabling attention in subsequent layers gives much better and comparable results because recurrence can encode positional information.

Moreover, SRU++ consistently achieves worse results by moving the attention to lower layer closer to the input embedding. We also enable a second attention layer while fixing the first one in the 10th layer. The corresponding results are shown in Figure 4 (bottom). Similarly, SRU++ achieves worse results if the attention is added to one of the lower layers. In contrast, results are comparable once the attention is placed in a high-enough layer. These observations suggest that the model should first learn local features before attention plays a most effective role at capturing long-range dependencies. More analyses can be found in Appendix A.

Does the ratio $d : d'$ matter? Transformer models by default use a FFN dimension that is 4 times larger than the attention dimension (Vaswani et al., 2017). We analyze the ratio of recurrence dimension d to attention dimension d' for SRU++. A small value of d' can reduce the amount of computation and the number of parameters used in attention layers but may limit the modeling capacity. Table 4 compares the results of using different $d : d'$ ratio given a similar amount of model parameters. We fix the model size to around 108M and use 10 SRU++ layers. Changing this ratio from 4 to a higher value gives better result. The best dev result is obtained with a ratio of 8.

Given this observation, we report SRU++ result using a default ratio of 4 as well as a ratio of 8 in the subsequent result sections. This ensures we conduct a comparison that uses a setup similarly to the default of Transformer models, but also showcases stronger results SRU++ can achieve.

ENWIK8 Table 3 compares our model with other top-performing models on the ENWIK8 dataset. We train a base model with $d = 3072$ and a large model with $d = 4096$ using 400K training steps. The unroll size and attention context length are set to 1024 during training and 3072 during evalua-

Model	Parameters ↓	Test PPL ↓	GPU days ↓
All-attention network 36L (Sukhbaatar et al., 2019b)	133M	20.6	-
Feedback Transformer (Fan et al., 2020)	139M	18.2	214
Transformer (Baevski and Auli, 2019)	247M	18.7	22†
Transformer-XL 18L (Dai et al., 2019)	257M	18.3	-
◦ Compressive memory (Rae et al., 2020)	-	17.1	-
Routing Transformer (Roy et al., 2021)	-	15.8	-
kNN-LM (Khandelwal et al., 2020)	-	15.8	-
SRU++ Base	148M	18.3	8†
SRU++ Large	232M	17.4	14†
◦ $d = 8\,d'$	234M	17.1	15†
◦ only 2 attention layers ($k = 5$)	225M	17.3	11†

Table 5: Comparison with top-performing models on WIKI-103 dataset. We include the training cost (measured by the number of GPUs used $\times$ the number of days) if it is reported in the previous work. The reported training costs are based on V100 GPUs. Our results are similarly obtained using an AWS p3dn instance with 8 V100 GPUs. † indicates mixed precision training.

Model	Param	PPL ↓	Days ↓
Transformer	331M	25.6	57†
		25.2	147†
	465M	23.9	192†
SRU++	328M	25.1	36†
SRU++ ($k = 5$)	465M	23.5	63†

Table 6: Test perplexity and effective GPU days for training of SRU++ models and the Transformer models of Baevski and Auli (2019) on BILLION WORD dataset.

Model	Speed↑	PPL↓
kNNLM (Khandelwal et al.)	145	15.8
Trans (Baevski and Auli)	2.5k	18.7
Trans-XL (Dai et al.)	3.2k	18.3
Shortformer (Press et al.)	15k	18.2
SRU++ Large	15k	17.1
SRU++ Large ($k = 5$)	22k	17.3

Table 7: Inference speed (tokens/second) on WIKI-103 test set. Results of baselines are taken from Press et al. (2021). We use a single V100 GPU, a batch size of 1 and maximum attention length 2560 for consistency.

tion. To compare the computation efficiency we report the effective GPU days – the number of GPUs multiplied by the number of days needed to finish training. Our base model achieves better BPC and uses a fraction of the training cost reported in previous work. Furthermore, our large models achieve a new state-of-the-art result on this dataset, reaching a test BPC of 0.96 when $d = 4\,d'$ and 0.95 when $d = 8\,d'$.

WIKI-103 Table 5 presents the result of SRU++ models and other top results on the WIKI-103 dataset. We train one base model with 148M parameters and a few large models which contain about 230M parameters. As shown in the table, our base model obtains a test perplexity of 18.3 using 8 GPU days of training, about 3x reduction compared to the Transformer model in Baevski and Auli (2019) and over 10x reduction compared to Feedback Transformer (Fan et al., 2020). Again, changing the hidden size ratio to $d = 8\,d'$ improves the modeling capacity. Our big model achieves a test perplexity of 17.1. The required training cost remains significantly lower.

BILLION WORD We double our training iterations to 800K and use a learning rate of 0.0002 for the BILLION WORD dataset. We train a base model using $d = 4096$, $d' = 1024$ and an effective batch size of 65K tokens per gradient update. We also train a large model by increasing the hidden size d to 7616 and the batch size to 98K. In addition, we use only 2 attention layers ($k = 5$) for the large model. Table 6 reports the test perplexity and associated training cost. Our base and large model obtain a test perplexity of 25.1 and 23.5 respectively, outperforming the Transformer model of Baevski and Auli (2019) given similar model size. Moreover, SRU++ achieves 3-4x training cost reduction and is trained using 8 GPUs. In comparison, the Transformer model uses 32 or 64 V100 GPUs.

Model	Param	BLEU↑	Hrs↓
Transformer	20.1M	35.9±0.1	10.5
SRU++	20.4M	36.3±0.2	8.5
SRU++ ($k = 2$)	19.6M	36.1±0.1	7.5

Table 8: Results on IWSLT'14 De→En test set. We use a beam size of 5. BLEU scores and training time are averaged over 4 independent runs.

Inference speed Table 7 compares the inference speed of SRU++ with other top-performing models on WIKI-103 test set. We use a single V100 GPU for inference. Our large model runs at least 4.5x faster than all baseline models except Shortformer (Press et al., 2021). In addition, our model achieves 0.9-1.1 perplexity lower than Shortformer and runs 50% faster when using 2 attention layers ($k = 5$).

IWSLT Does SRU++ work well for other tasks? We study this question by evaluating SRU++ on the IWSLT'14 De→En translation task. We use the open-sourced training and evaluation code of Lin et al. (2020). The base model is an 8-layer Transformer model containing 20M parameters. We train SRU++ models using 6 layers and $d = 1024$, resulting in similar number of parameters. We use the original settings such as learning rate and batch size, except that we use RAdam optimizer for consistency and increase the number of training epochs to 50. Both architectures achieve much higher BLEU scores given more training epochs.[3] Table 8 presents the test results. Without additional hyperparameter tuning, SRU++ achieves 0.4 BLEU score higher and less training time compared to the Transformer model tuned in Lin et al. (2020).

Why does SRU++ reduce training cost in our experiments? Several factors contribute to the computation reduction observed in our experiments. First, combining attention and recurrence gives stronger modeling capacity. As shown in our experiments, SRU++ often achieves comparable results using fewer layers and/or fewer parameters. The required computation are much lower for shallower and smaller models.

We also observe higher training efficiency, requiring fewer training steps and smaller training batch compared to several Transformer models. For example, SRU++ uses a maximum effective batch size of 98K tokens and 800K training steps on the BILLION WORD dataset, while the Transformer model in comparison (Baevski and Auli, 2019) uses 128K tokens and near 1000K steps. The reduced batch size and gradient updates cut down the training cost.

Finally, model implementation is an important factor for computation saving. Our implementation is highly efficient for two reasons. First, the fast recurrence operation of SRU is a reusable module that is already optimized for speed (Lei et al., 2018). Second, since recurrence encodes positional information, we can use simple single-head attention and remove positional encoding.

On the contrary, advanced attention and positional encoding mechanism can generate non-trivial computation overhead. To see this, we measure the running time of SRU++ and Transformer-XL using Pytorch Profiler. Figure 5 (a) shows the average model forward time of a single batch. SRU++ runs 4-5x times faster compared to the Transformer-XL implementation. Figure 5 (b) breaks down the computation and highlights the most time-consuming operations in both models. The matrix multiplications are one of the most expensive operations for both models. Surprisingly, many operations in the relative attention of Transformer-XL are computationally expensive. For example, the relative attention requires shifting the attention scores and adding up different attention score matrices. Both require a lot of time but they are not needed in non-relative attention. In addition, the last column shows the running time of tensor transpose operators needed by batch matrix-matrix multiplications in attention. Again, the relative attention uses an order of magnitude more time compared to the simple single-head attention used in our model implementation.[4]

6 Related Work

Accelerating common architectures for NLP has become an increasingly important research topic recently (Tay et al., 2020; Sun et al., 2020; Lan et al., 2020). Our work is closely related to two lines of research under this topic.

[3] Lin et al. (2020) reports a test BLEU of 35.2. We obtain 35.9 for the same Transformer model by training longer.

[4] Note that this high latency of tensor transpose might be caused by sub-optimal implementation choices such as a poor arrangement of tensor axes in the open-sourced model. There is room for improvement. Nevertheless, relative attention and positional encoding are reported to be non-trivially slower in other works (Shaw et al., 2018; Tian et al., 2021).

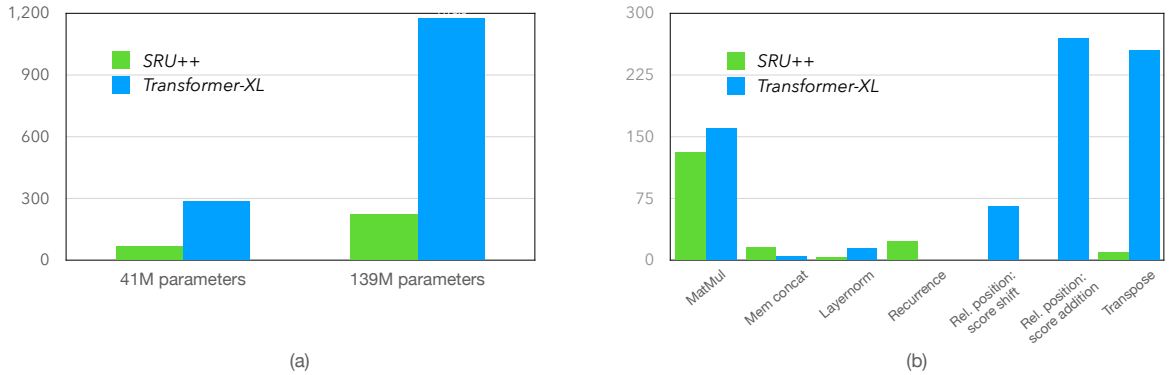


Figure 5: Profiling of SRU++ and Transformer-XL: (a) forward time (in milliseconds) of small and large models and (b) forward time used in various types of time-consuming operations. We use a single GPU for profiling to avoid extra overhead such as data synchronization between GPUs. We use an unroll size / context length $M = 512$ and 1024 respectively for small and large models. All models use a batch size $B = 16$ for profiling.

First, previous works have tackled the speed problem of recurrent neural networks (RNNs) and have proposed various fast RNN implementations (Diamos et al., 2016; Campos et al., 2018; Zhang and Sennrich, 2019). Notably, the Quasi-RNN (Bradbury et al., 2017) and SRU (Lei et al., 2018) have invented highly-parallelizable recurrence and combined them with convolutions or highway networks respectively. The resulting architectures achieve equivalent parallelism as convolutional and attention models. This advancement eliminates the need of avoiding recurrence computation to trade model training efficiency, a design choice made by the Transformer architecture. Our model builds on top of SRU.

Second, several recent works have argued that using attention alone is not the best architecture in terms of model expressiveness. For example, Dong et al. (2021) demonstrate theoretically and empirically that using pure attention results in performance degeneration. Gulati et al. (2020) have combined convolution and attention and obtained new state-of-the-art results for speech recognition. Moreover, RNNs have been incorporated into Transformer architectures, resulting in improved results in machine translation and language understanding tasks (Lei et al., 2018; Huang et al., 2020). Our work is built upon a similar hypothesis that recurrence and attention are complementary at sequence modeling. We demonstrate that jointly leveraging fast recurrence and attention not only achieves state-of-the-art modeling results but also obtain significant computation reduction.

Being orthogonal to our work, many recent works improve the efficiency of Transformer models by accelerating attention computation (Zaheer et al., 2020; Katharopoulos et al., 2020; Vyas et al., 2020; Peng et al., 2021). Examples include Longformer (Beltagy et al., 2020), Reformer (Kitaev et al., 2020), Linformer (Wang et al., 2020) and Routing Transformer (Roy et al., 2021). In contrast, our work optimizes computational efficiency using recurrence combined with minimal attention and our model can incorporate these attention variants for additional speed improvement.

7 Conclusion

We present a highly-efficient architecture combining fast recurrence and attention, and evaluate its effectiveness on various language modeling datasets. We demonstrate fast RNNs with little attention not only achieve top results but also reduce training cost significantly. Our work shares a different idea to accelerating attention, therefore providing an orthogonal direction to advancing state-of-the-art model architecture. As future work, we believe the model can be improved using stronger attention or recurrent implementations, better normalization or optimization techniques.

Acknowledgement

We would like to thank ASAPP Inc. for making this work possible. We thank Hugh Perkins, Joshua Shapiro, Sam Bowman, Danqi Chen and Yu Zhang for providing invaluable feedback for this work. Finally, we thank Jeremy Wohlwend, Jing Pan, Prashant Sridhar and Kyu Han for helpful discussions, and ASAPP Language Technology and Infra teams for the compute cluster setup for our research experiments.

References

Jeremy Appleyard, Tomas Kocisky, and Phil Blunsom. 2016. Optimizing performance of recurrent neural networks on gpus. *arXiv preprint arXiv:1604.01946*.

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

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations (ICLR)*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv:2004.05150*.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-Recurrent Neural Networks. In *International Conference on Learning Representations (ICLR)*.

Andrew Brock, Soham De, and Samuel L Smith. 2021. Characterizing signal propagation to close the performance gap in unnormalized resnets. In *International Conference on Learning Representations*.

Víctor Campos, Brendan Jou, Xavier Giró i Nieto, Jordi Torres, and Shih-Fu Chang. 2018. Skip rnn: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations (ICLR)*.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. Technical report, Google.

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)*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Greg Diamos, Shubho Sengupta, Bryan Catanzaro, Mike Chrzanowski, Adam Coates, Erich Elsen, Jesse Engel, Awni Hannun, and Sanjeev Satheesh. 2016. Persistent rnns: Stashing recurrent weights on-chip. In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*.

Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. 2021. Attention is not all you need: pure attention loses rank doubly exponentially with depth. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. 2020. Accessing higher-level representations in sequential transformers with feedback memory. *arXiv preprint arXiv:2002.09402*.

Edouard Grave, Armand Joulin, Moustapha Cissé, Hervé Jégou, et al. 2017. Efficient softmax approximation for gpus. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. 2020. Conformer: Convolution-augmented transformer for speech recognition. In *Proceedings of the 21st Annual Conference of the International Speech (INTERSPEECH)*.

Jie Hao, Xing Wang, Baosong Yang, Longyue Wang, Jinfeng Zhang, and Zhaopeng Tu. 2019. Modeling recurrence for transformer. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*.

Elad Hoffer, Itay Hubara, and Daniel Soudry. 2017. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*.

Yi-Te Hsu, Sarthak Garg, Yi-Hsiu Liao, and Ilya Chatsviorkin. 2020. Efficient inference for neural machine translation. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*.

Zhiheng Huang, Peng Xu, Davis Liang, Ajay Mishra, and Bing Xiang. 2020. Trans-blstm: Transformer with bidirectional lstm for language understanding. *arXiv preprint arXiv:2003.07000*.

Marcus Hutter. 2006. The human knowledge compression contest. *http://prize.hutter1.net/*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations (ICLR)*.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *International Conference on Learning Representations (ICLR)*.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. 2018. Dynamic evaluation of neural sequence models. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations (ICLR)*.

Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. 2018. Simple recurrent units for highly parallelizable recurrence. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Alexander Lin, Jeremy Wohlwend, Howard Chen, and Tao Lei. 2020. Autoregressive knowledge distillation through imitation learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. In *International Conference on Learning Representations (ICLR)*.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2020a. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations (ICLR)*.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020b. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Stephen Merity. 2019. Single headed attention rnn: Stop thinking with your head. *arXiv preprint arXiv:1911.11423*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*.

Jinhwan Park, Yoonho Boo, Iksoo Choi, Sungho Shin, and Wonyong Sung. 2018. Fully neural network based speech recognition on mobile and embedded devices. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. 2021. Random feature attention. In *International Conference on Learning Representations (ICLR)*.

Ofir Press, Noah A. Smith, and Mike Lewis. 2021. Shortformer: Better language modeling using shorter inputs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations (ICLR)*.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. 2020. Green AI. *Communications of the ACM*.

Yuan Shangguan, Jian Li, Qiao Liang, Raziel Alvarez, and Ian McGraw. 2019. Optimizing speech recognition for the edge. *arXiv preprint arXiv:1909.12408*.

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*.

Sheng Shen, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. 2020. PowerNorm: Rethinking batch normalization in transformers. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019a. Adaptive attention span in transformers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Herve Jegou, and Armand Joulin. 2019b. Augmenting self-attention with persistent memory. *arXiv preprint arXiv:1907.01470*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*.

Ran Tian, Joshua Maynez, and Ankur P Parikh. 2021. Shatter: An efficient transformer encoder with single-headed self-attention and relative sequence partitioning. *arXiv preprint arXiv:2108.13032*.

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 *Advances in Neural Information Processing Systems (NeurIPS)*.

Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. 2020. Fast transformers with clustered attention. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. 2020. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. 2019. Understanding and improving layer normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems (NeurIPS)*.

Biao Zhang and Rico Sennrich. 2019. A lightweight recurrent network for sequence modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

A Additional results

A.1 Detailed analysis of attention

Table 10 presents a more comprehensive analysis of attention in SRU++ models. First, we change the number of attention layers and their locations in the model. As shown in the top block of Table 10, using attention in 50% of the layers leads to no (or negligible) loss in model performance. This is consistent with the results in Table 2 using a smaller model. Enabling attention in higher layers performs slightly better than evenly distributing attention from the bottom to top layers.

We also experiment with using more than one attention head in each of the attention layer, as shown in the middle block of the table. Unlike Transformer models however, we do not observe a significant improvement using multiple heads. We hypothesize that the recurrence states can already carry different features or information that are present in different input positions, making redundant heads unnecessary.

Finally, changing the ratio $d : d'$ from 4 to 8 gives similar improvements regardless of using 2 attention layers or 10 attention layers. This suggests that the amount of attention and the hidden size ratio can be tuned independently for best model performance.

A.2 The effectiveness of layer normalization

In our experiments, we have always used layer normalization to stabilize training. However, we also found layer normalization to achieve worse generalization for larger models that are more prone to over-fitting. Figure 6 showcases our empirical observation on the ENWIK8 dataset. Using layer normalization achieves more rapid training progress and lower training loss, but results in higher dev loss in the case of training a 108M model. This generalization gap remains even if we tune the dropout rate carefully. In addition, although using layer normalization in the smaller model with 41M parameters gives slightly better dev results, we still observe a larger generalization gap (indicated by the difference between training loss and dev loss) compared to the run without layer normalization. Similar over-fitting patterns are observed on Wiki-103 dataset, and also in previous work (Xu et al., 2019).

On the other hand, turning off layer normalization can achieve better generalization but makes training sensitive to learning rate and parameter initialization. For example, we have to use a smaller learning rate of 0.00025 or lower to avoid sudden gradient explosion during training. These results suggest possible future work by improving the normalization method (Shen et al., 2020; Brock et al., 2021).

A.3 Tuning weight decay and learning rate

We find that tuning the weight decay and learning rate critical to the success of training SRU++ and achieving best results. Table 9 provides a sensitivity analysis by testing different learning rates and weight decay values. Increasing the weight decay consistently gives better results for all learning rates tested. Tuning the learning rate is also needed to reach the best result. The non-trivial effect of weight decay seems to be unique for SRU++.

On the other hand, the performance of SRU++ remains robust once the appropriate weight decay and learning rate are set. As shown in previous results and analyses, SRU++ achieves strong and relatively stable results to various hidden sizes, number of attention layers and datasets. In particular, using the same weight decay value generalize well for all datasets (including language modeling and translation tasks) and model configurations tested.

	0.10	0.01	0.00
3×10^{-4}	**1.014**	-	-
2×10^{-4}	1.022	1.035	1.047
1.5×10^{-4}	1.030	1.038	1.040

Table 9: Dev BPC of SRU++ given a learning rate $\in \{1.5, 2, 3\} \times 10^{-4}$ and a weight decay $\in \{0.1, 0.01, 0\}$. '-' means the training run diverged or got gradient explosion.

B Training details

Language modeling We use the RAdam optimizer[5] with the default hyperparameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for all our experiments. We use a cosine learning rate schedule with only 1 cycle for simplicity. For faster training, we also leverage the native automatic mixed precision (AMP) training and distributed data parallel (DDP) of Pytorch in all experiments, except those in Table 1 and Fig-

[5] https://github.com/LiyuanLucasLiu/RAdam

ure 1 for a fair comparison with the Transformer-XL implementation.

Table 11 shows the detailed training configuration of SRU++ models on ENWIK8 dataset. Most training options are kept the same for all models. We tune the dropout probability more carefully as we found training is more prone to over-fitting and under-fitting for this dataset. The large model is trained with 2x batch size. As a result, we increase the learning rate proportionally by a factor of $\sqrt{2}$ (Hoffer et al., 2017), which results in a rounded learning rate of 0.0004.

Table 12 presents the detailed training configuration on WIKI-103 dataset. Similarly we use $d = 3072$ and $d = 4096$ for the base and large model respectively for a hidden size ratio $d : d' = 4 : 1$. Following (Baevski and Auli, 2019), we use an adaptive word embedding layer and an adaptive softmax layer for our models, and we tie the weight matrices of the two layers. We keep the total number of parameters comparable when we use a different hidden size ratio $d : d' = 8 : 1$.

Machine translation We use the open-sourced code from Lin et al. (2020) for the IWSLT'14 De→En translation task. The Transformer model tuned by the original work uses 8 layers for both the encoder and decoder and a total of 20M parameters. Most of the training configuration remains the same as the original work[6], except for a couple of changes. First, we use RAdam optimizer and the same β values for consistency with the language model task. We use the same weight decay value of 0.1 for SRU++. The Transformer model uses a weight decay of 0 that is tuned based on dev set performance. Second, we increase the number of training epochs to 50 (or equivalently 64K training steps) since all models achieve better BLEU scores by training longer. This ensures we compare models when they reach the maximum performance.

Our SRU++ model uses a hidden size $d = 1024$, an attention size $d' = 256$ and 6 layers for the encoder and decoder, resulting in a similar number of parameters as the Transformer model in comparison. Let $\mathbf{X}_{src}$ be the output representation of the SRU++ encoder. Each SRU++ decoder layer make uses of $\mathbf{X}_{src}$ by simplying treating it as extra attention context. That is, the query, key and value representations are computed by concatenating the input of the current layer $\mathbf{X}_{tgt}$ with $\mathbf{X}_{src}$,

$$\begin{aligned}\mathbf{Q} &= [\mathbf{Q}_{src}, \mathbf{Q}_{tgt}] \\ &= \mathbf{W}^q \, [\mathbf{X}_{src}, \mathbf{X}_{tgt}]^\top \\ \mathbf{K} &= \mathbf{W}^k \, \mathbf{Q} \\ \mathbf{V} &= \mathbf{W}^v \, \mathbf{Q}\end{aligned}$$

The resulting representations $\mathbf{Q}_{tgt}$, $\mathbf{K}$ and $\mathbf{V}$ are used for the rest of the attention computation. The attention mask is set such that each target token can only attend to all source tokens and preceding target tokens.

[6] https://github.com/asappresearch/imitkd/blob/master/configs/iwslt/teacher.yaml

Layers that has attention	Num of heads	d	d'	Model size	Dev BPC
All layers	1	3072	768	108M	0.997
6,7,8,9,10				102M	0.997
2,4,6,8,10				102M	0.999
8,9,10		3136	784	103M	1.000
3,6,9					1.001
5,10	1	3072	768	98M	1.002
	2				1.002
10	1			97M	1.007
	2				1.006
All layers	1	3072	768	108M	0.997
5,10				98M	1.002
All layers		4480	560	109M	0.991
5,10				104M	0.997

Table 10: Results of 10-layer SRU++ models by varying the attention setting. We report the dev BPC on the ENWIK8 dataset. The first column indicates layers where the attention are located. Smaller index numbers represent layers that are closer to the input of the model.

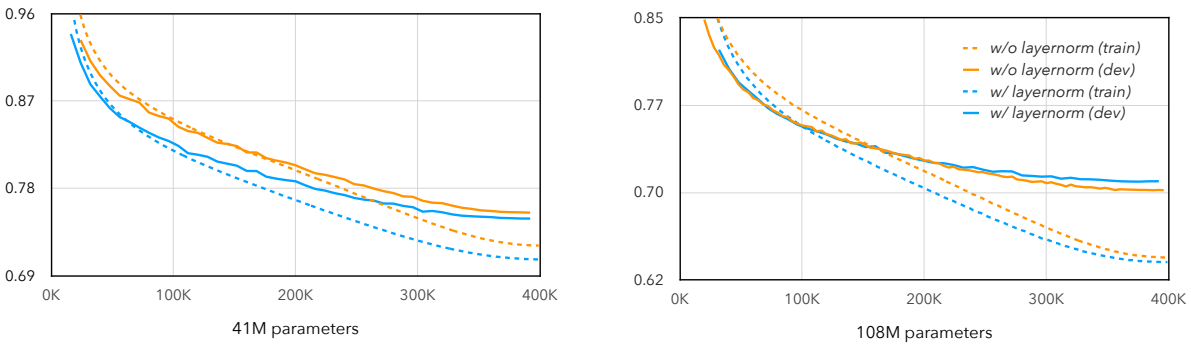


Figure 6: Understanding the empirical effect of layer normalization. We show the training and dev loss of SRU++ models using 41M parameters and 108M parameters on ENWIK8 dataset. The model with layer normalization fits the training data better, but achieves worse generalization.

	Base model ($k = 5$)	Base model	Large model	Large model
Attention / unroll size - train	1024	1024	1024	1024
Attention / unroll size - test	3072	3072	3072	3072
Batch size $\times$ Num of GPUs	4×8	4×8	8×8	8×8
Dropout	0.22	0.22	0.32	0.35
Gradient clipping	1.0	1.0	1.0	1.0
Hidden size ratio $d : d'$	4	4	4	8
Hidden size d	3072	3072	4096	6016
Hidden size d'	768	768	1024	752
Learning rate	0.0003	0.0003	0.0004	0.0004
LR warmup steps	16K	16K	16K	16K
Training steps	400K	400K	400K	400K
Weight decay	0.1	0.1	0.1	0.1
Model size	98M	108M	191M	195M
Dev BPC	1.002	0.997	0.985	0.974
Test BPC	0.980	0.974	0.963	0.953

Table 11: Training details of SRU++ models on ENWIK8 dataset.

	Base model	Large model	Large model ($k = 5$)	Large model
Attention / unroll size - train	768	1024	1024	1024
Attention / unroll size - test	2560	2560	2560	2560
Batch size $\times$ Num of GPUs	8×8	8×8	8×8	8×8
Dropout	0.15	0.2	0.2	0.2
Gradient clipping	1.0	1.0	1.0	1.0
Hidden size ratio $d : d'$	4	4	8	8
Hidden size d	3072	4096	5952	5952
Hidden size d'	768	1024	744	744
Learning rate	0.0003	0.0003	0.0003	0.0003
LR warmup steps	16K	16K	16K	16K
Training steps	400K	400K	400K	400K
Weight decay	0.1	0.1	0.1	0.1
Model size	148M	232M	225M	234M
Dev PPL	17.5	16.7	16.6	16.4
Test PPL	18.3	17.4	17.3	17.1

Table 12: Training details of SRU++ models on WIKI-103 dataset.