arXiv:1709.02755v1 [cs.CL] 8 Sep 2017

Training RNNs as Fast as CNNs

Tao Lei
ASAPP Inc.
tao@asapp.com

Yu Zhang
MIT CSAIL
yzhang87@csail.mit.edu

Abstract

Recurrent neural networks scale poorly due to the intrinsic difficulty in parallelizing their state computations. For instance, the forward pass computation of h_t is blocked until the entire computation of h_{t-1} finishes, which is a major bottleneck for parallel computing. In this work, we propose an alternative RNN implementation by deliberately simplifying the state computation and exposing more parallelism. The proposed recurrent unit operates as fast as a convolutional layer and 5-10x faster than cuDNN-optimized LSTM. We demonstrate the unit's effectiveness across a wide range of applications including classification, question answering, language modeling, translation and speech recognition. We open source our implementation in PyTorch and CNTK[1].

1 Introduction

Many recent advances in deep learning have come from increased model capacity and associated computation. This often involves using larger and deeper networks that are tuned with extensive hyper-parameter settings. The growing model sizes and hyper-parameters, however, have greatly increased the training time. For instance, training a state-of-the-art translation or speech recognition system would take several days to complete (Vaswani et al., 2017; Wu et al., 2016b; Sak et al., 2014). Apparently, computation has become a major bottleneck for deep learning research.

To counter the dramatically increased computation, parallelization such as GPU-accelerated training has been predominately adopted to scale deep learning (Diamos et al., 2016; Goyal et al., 2017). While operations such as convolution and attention are well-suited for multi-threaded / GPU computation, recurrent neural nets, however, remain less amenable to parallelization. In a typical implementation, the computation of output state h_t is suspended until the entire computation of h_{t-1} completes. This constraint impedes independent computation and largely slows down sequence processing. Figure 1 illustrates the processing time of cuDNN-optimized LSTMs (Appleyard et al., 2016) and word-level convolutions using `conv2d`. The difference is quite significant – even the very optimized LSTM implementation performs over 10x slower.

In this work, we introduce the Simple Recurrent Unit (SRU) which operates significantly faster than existing recurrent implementations. The recurrent unit simplifies state computation and hence exposes the same parallelism as CNNs, attention and feed-forward nets. Specifically, while the update of internal state c_t still makes use of the previous state c_{t-1}, the dependence on h_{t-1} in a recurrence step has been dropped. As a result, all matrix multiplications (i.e. `gemm`) and element-wise operations in the recurrent unit can be easily parallelized across different dimensions and steps. Similar to the implementation of `cuDNN LSTM` and `conv2d`, we perform CUDA-level optimizations for SRU by compiling all element-wise operations into a single kernel function call. As shown in Figure 1, our implementation achieves the same speed of its `conv2d` counterpart.

Of course, an alternative implementation that fails to deliver comparable or better accuracy would have no applicability. To this end, we evaluate SRU on a wide range of applications including classifi-

[1] `https://github.com/taolei87/sru`

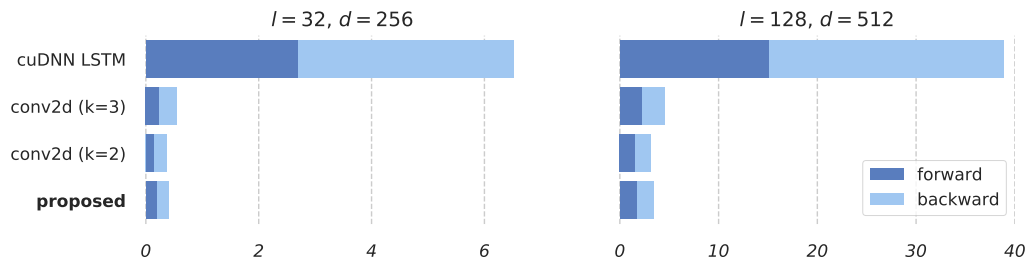

Figure 1: Average processing time (in milliseconds) of a batch of 32 samples using `cuDNN LSTM`, word-level convolution `conv2d`, and the proposed RNN implementation. l: number of tokens per sequence, d: feature dimension and k: feature width. Numbers reported are based on PyTorch with an Nvidia GeForce GTX 1070 GPU and Intel Core i7-7700K Processor.

cation, question answering, language modeling, translation and speech recognition. Experimental results confirm the effectiveness of SRU – it achieves better performance compared to recurrent (or convolutional) baseline models across these tasks, while being able to train much faster.

2 Method

We present Simple Recurrent Unit (SRU) in this section. We start with a basic gated recurrent neural network implementation, and then present necessary modifications for the speed-up. The modifications can be adopted to other gated recurrent neural nets, being not restricted to this particular instance.

2.1 SRU implementation

Most top-performing recurrent neural networks such as long short-term memory (LSTMs) (Hochreiter and Schmidhuber, 1997) and gated recurrent unit (GRUs) (Cho et al., 2014) make use of *neural gates* to control the information flow and alleviate the gradient vanishing (or explosion) problem. Consider a typical implementation,

$$\begin{aligned}\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{x_t} \\ &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \tilde{\mathbf{x}}_t\end{aligned}$$

where $\mathbf{f}_t$ and i_t are sigmoid gates referred as the *forget gate* and *input gate*. $\tilde{\mathbf{x}}_t$ is the transformed input at step t. We choose the coupled version $\mathbf{i}_t = 1 - \mathbf{f}_t$ here for simplicity. The computation of $\tilde{\mathbf{x}}_t$ also varies in different RNN instances. We use the simplest version that performs a linear transformation over the input vector $\tilde{\mathbf{x}}_t = \mathbf{W}\mathbf{x}_t$ (Lei et al., 2017; Lee et al., 2017). Finally, the internal state $\mathbf{c}_t$ is passed to an activation function $g(\cdot)$ to produce the output state $\mathbf{h}_t = g(\mathbf{c}_t)$.

We include two additional features in our implementation. First, we add skip connections between recurrent layers since they are shown quite effective for training deep networks (He et al., 2016; Srivastava et al., 2015; Wu et al., 2016a). Specifically, we use highway connections (Srivastava et al., 2015) and the output state $\mathbf{h}'_t$ is computed as,

$$\mathbf{h}'_t = \mathbf{r}_t \odot \mathbf{h}_t + (1 - \mathbf{r}_t) \odot \mathbf{x}_t \tag{1}$$

$$= \mathbf{r}_t \odot g(\mathbf{c}_t) + (1 - \mathbf{r}_t) \odot \mathbf{x}_t \tag{2}$$

where $\mathbf{r}_t$ is the output of a *reset gate*. Second, we implement variational dropout (Gal and Ghahramani, 2016) in addition to the standard dropout for RNN regularization. Variational dropout uses a shared dropout mask across different time steps t. The mask is applied over input $\mathbf{x}_t$ during every matrix multiplication in RNN (i.e., $\mathbf{W} \cdot \text{drop}(\mathbf{x}_t)$). Standard dropout is performed on $\mathbf{h}_t$ before it is given to the highway connection.

2.2 Speeding-up the recurrence

Existing RNN implementations use the previous output state $\mathbf{h}_{t-1}$ in the recurrence computation. For instance, the forget vector would be computed by $\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{R}_f\mathbf{h}_{t-1} + \mathbf{b}_f)$. The inclusion of

$\mathbf{R}\mathbf{h}_{t-1}$ breaks independence and parallelization: each dimension of the hidden state depends on the other, hence the computation of $\mathbf{h}_t$ has to wait until the entire $\mathbf{h}_{t-1}$ is available.

We propose to completely drop the connection (between $\mathbf{h}_{t-1}$ and the neural gates of step t). The associated equations of SRU are given below,

$$\tilde{\mathbf{x}_t} = \mathbf{W}\mathbf{x}_t \tag{3}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{b}_f) \tag{4}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{b}_r) \tag{5}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \tilde{\mathbf{x}_t} \tag{6}$$

$$\mathbf{h}_t = \mathbf{r}_t \odot g(\mathbf{c}_t) + (1 - \mathbf{r}_t) \odot \mathbf{x}_t \tag{7}$$

Given a sequence of input vectors $\{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$, $\{\tilde{\mathbf{x}_t}, \mathbf{f}_t, \mathbf{r}_t\}$ for different $t = 1 \cdots n$ are independent and hence all these vectors can be computed in parallel. Our formulation is similar to the recently proposed Quasi-RNN (Bradbury et al., 2017). While we drop $\mathbf{h}_{t-1}$ in the linear transformation terms of Eq (3) to (5), Quasi-RNN uses k-gram `conv2d` operations to substitute the linear terms. The computation bottleneck of our network is simply the three matrix multiplications in (3)$\sim$(5). After computing $\tilde{\mathbf{x}_t}$, $\mathbf{f}_t$ and $\mathbf{r}_t$, Eq (6) and (7) can be computed quite easily and fast since all operations are element-wise.

One open question is whether the simplification reduces the representational capability of the recurrent model. A theoretical analysis regarding the representational characteristics of (a broader class of) such recurrent architectures is presented in (Lei et al., 2017). In our experimental section, we empirically demonstrate that SRU can achieve the same or better performance by stacking the same number of or more layers. In fact, training a deeper network with SRU is much easier since each layer enjoys less computation and higher processing speed.

2.3 CUDA level optimization

A naive implementation of SRU in existing deep learning libraries can already achieve over 5x speed-up over the naive LSTM implementation. This is still sub-optimal since implementation on top of DL libraries introduces a lot of computation overhead such as data copy and kernel launching latencies. In contrast, convolution `conv2d` and `cuDNN LSTM` (Appleyard et al., 2016) have been optimized as CUDA kernel functions to accelerate their computation. To demonstrate the potential and compare with these implementations, we implement a version with CUDA-level optimizations in PyTorch.

Optimizing SRU is similar to but much easier than the `cuDNN LSTM` (Appleyard et al., 2016). Our RNN formulation permits two optimizations that become possible for the first time in RNNs. First, the matrix multiplications across all time steps can be batched, which can significantly improve the computation intensity and hence GPU utilization. Second, all element-wise operations of the sequence can be fused (compiled) into one kernel function and be parallelized across hidden dimensions. Without the fusion, operations such as addition $+$ and sigmoid activation $\sigma()$ would invoke a separate function call. This brings additional kernel launching latency and also adds data moving cost.

Specifically, the matrix multiplications in Eq (3)$\sim$(5) are grouped into one single multiplication, which can be formulated as,

$$\mathbf{U}^\top = \begin{pmatrix} \mathbf{W} \\ \mathbf{W}_f \\ \mathbf{W}_r \end{pmatrix} [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] \tag{8}$$

where $\mathbf{U} \in \mathbb{R}^{n \times 3d}$ is the resulting matrix. When the input is a mini-batch of k sequences, $\mathbf{U}$ would be a tensor of size $(n, k, 3d)$. We choose a length-major representation instead of a batch-major version here. The pseudocode of the fused kernel function is presented in Algorithm 1.

3 Experiments

We evaluate SRU on a diverse set of benchmarks. These benchmarks are chosen to have a broad coverage of application scenarios and computation difficulties. Specifically, we train models on text

Algorithm 1 Mini-batch version of the forward pass defined in Eq (3) to (7).

Input: $\mathbf{x}[l, i, j]$, $\mathbf{U}[l, i, j']$, $\mathbf{b}_f[j]$ and $\mathbf{b}_r[j]$; initial state $\mathbf{c}_0[i, j]$.
$l = 1, \cdots, n$, $i = 1, \cdots, k$, $j = 1, \cdots, d$ and $j' = 1, \cdots, 3d$

```
Initialize h[·,·,·] and c[·,·,·] as two n × k × d tensors.
for i = 1, ··· , k; j = 1, ··· , d do                    // parallelize over i and j
    c = c_0[i, j]
    for l = 1, ··· , n do
        f = σ ( U[l, i, j + d] + b_f[j] )
        r = σ ( U[l, i, j + d × 2] + b_r[j] )
        c = f × c + (1 − f) × U[l, i, j]
        h = r × g(c) + (1 − r) × x[l, i, j]
        c[l, i, j] = c
        h[l, i, j] = h
    end for
end for
return h[·,·,·] and c[·,·,·]
```

classification, question answering, language modeling, machine translation and speech recognition tasks. Training time on these benchmarks ranges from a couple of minutes (for classification) to several days (for speech).

We are primarily interested in whether SRU achieves better results and better performance-speed trade-off compared to other (recurrent) alternatives. To this end, we stack multiple layers of SRU as a direct substitute of other recurrent (or convolutional) modules in a model. We minimize hyper-parameter tuning and architecture engineering for a fair comparison with prior work, since such effort has a non-trivial impact on the results. The model configurations are made (mostly) consistent with prior work.

3.1 Classification

Dataset We use 6 classification datasets from (Kim, 2014)[2]: movie reviews (MR) (Pang and Lee, 2005), subjectivity data (SUBJ) (Pang and Lee, 2004), customer reviews (CR) (Hu and Liu, 2004), TREC questions (Li and Roth, 2002), opinion polarity from MPQA data (Wiebe et al., 2005) and Stanford sentiment treebank (SST) (Socher et al., 2013)[3]. All these datasets contain several thousand annotated sentences. We use the `word2vec` embeddings trained on 100 billion tokens from Google News, following (Kim, 2014). The word vectors are normalized to unit vectors and are fixed during training.

Setup We train RNN encoders and use the last hidden state to predict the class label for a given input sentence. For most datasets, a 2-layer RNN encoder with 128 hidden dimensions suffices to produce good results. We experiment with 4-layer RNNs for SST dataset since the amount of annotation is an order of magnitude larger than other datasets. In addition, we train the same CNN model of (Kim, 2014) under our setting as a reference. We use the same filter widths and number of filters as (Kim, 2014). All models are trained using default `Adam` optimizer with a maximum of 100 epochs. We tune dropout probability among $\{0.1, 0.3, 0.5, 0.7\}$ and report the best results.

Results Table 1 presents the test accuracy on the six benchmarks. Our model achieves better accuracy consistently across the datasets. More importantly, our implementation processes data significantly faster than `cuDNN LSTM`. Figure 2 plots the validation curves of our model, `cuDNN LSTM` and the wide CNNs of (Kim, 2014). On the movie review dataset for instance, our model completes 100 training epochs within 40 seconds, while `cuDNN LSTM` takes more than 450 seconds.

[2] `https://github.com/harvardnlp/sent-conv-torch`

[3] We use the binary version of Stanford sentiment treebank.

Model	CR	SUBJ	MR	TREC	MPQA	SST
wide CNNs	82.2 ±2.2	92.9 ±0.7	79.1 ±1.5	93.2 ±0.5	88.8 ±1.2	85.3 ±0.4
cuDNN LSTM	82.7 ±2.9	92.4 ±0.6	80.3 ±1.5	93.1 ±0.9	89.2 ±1.0	87.9 ±0.6
SRU	84.8 ±1.3	93.4 ±0.8	82.2 ±0.9	93.9 ±0.6	89.7 ±1.1	89.1 ±0.3

Table 1: Test accuracies on classification benchmarks. Wide CNNs refer to the sentence convolutional model (Kim, 2014) using 3, 4, 5-gram features (i.e. filter width 3, 4, 5). We perform 10-fold cross validation when there is no standard train-dev-test split. The result on SST is averaged over 5 independent trials. All models are trained using `Adam` optimizer with default learning rate $= 0.001$ and weight decay $= 0$.

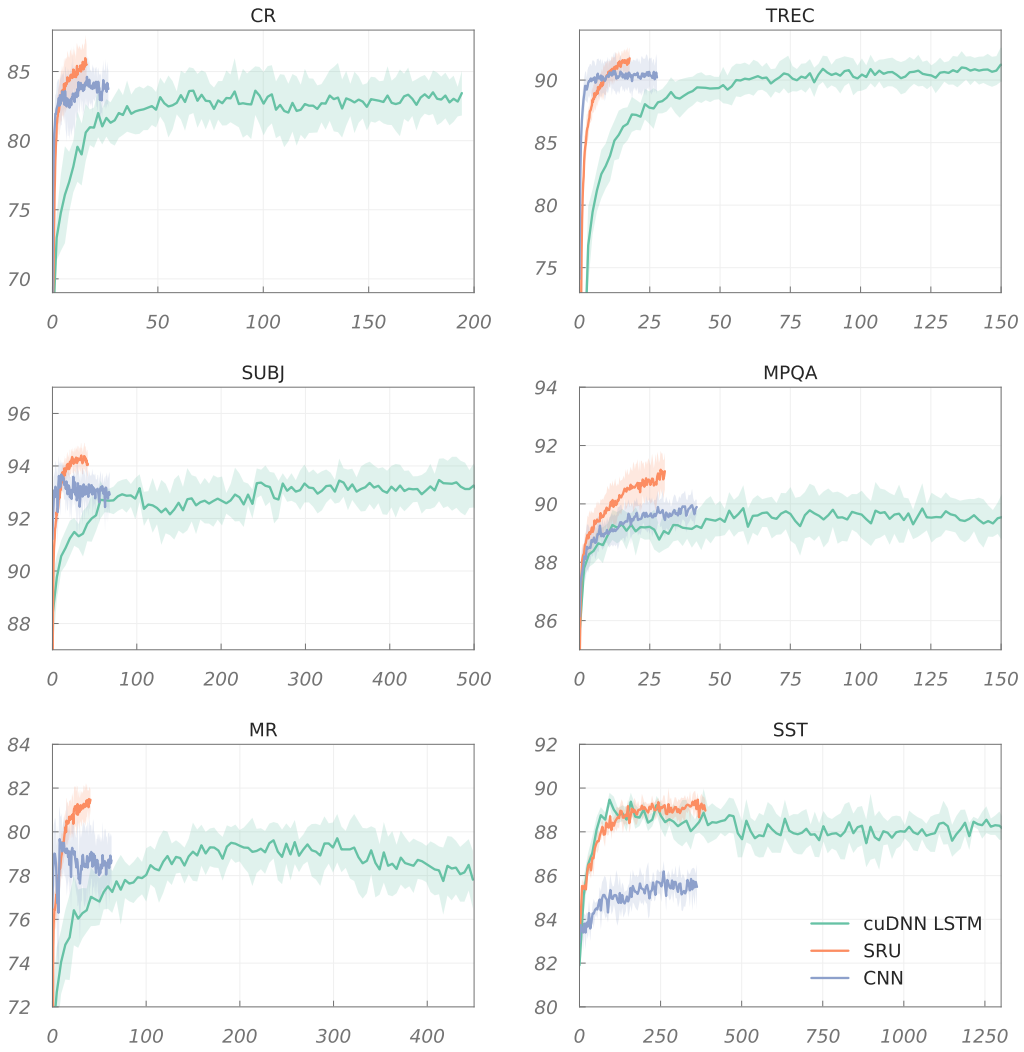

Figure 2: Mean validation accuracies (y-axis) of LSTM, CNN and SRU for the first 100 epochs on 6 classification benchmarks. X-axis: training time (in seconds) relative to the first iteration. Timings are performed on PyTorch and a desktop machine with a single Nvidia GeForce GTX 1070 GPU, Intel Core i7-7700K Processor, CUDA 8 and cuDNN 6021.

Model	# layers	d	Size	Dev EM	Dev F1	Time / epoch	
						RNN	Total
(Chen et al., 2017)	3	128	4.1m	69.5	78.8	-	-
Bi-LSTM	3	128	4.1m	69.6	78.7	534s	670s
Bi-LSTM	4	128	5.8m	69.6	78.9	729s	872s
Bi-SRU	3	128	2.0m	69.1	78.4	60s	179s
Bi-SRU	4	128	2.4m	69.7	79.1	74s	193s
Bi-SRU	5	128	2.8m	70.3	79.5	88s	207s

Table 2: EM (exact match) and F1 scores of various models on SQuAD. We also report the total processing time per epoch and the time used in RNNs. SRU achieves better results and operates more than 6 times faster than `cuDNN LSTM`. Timings are performed on a desktop machine with a single Nvidia GeForce GTX 1070 GPU and Intel Core i7-7700K Processor.

3.2 Question answering

Dataset We use Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) as our benchmark. It is one of the largest machine comprehension dataset, consisting over 100,000 question-answer pairs extracted from Wikipedia articles. We use the standard train and dev sets provided on the official website.

Setup We train the Document Reader model as described in (Chen et al., 2017) and compare the model variants which use LSTM (original setup) and SRU (our setup). We use the open source PyTorch re-implementation[4] of the Document Reader model. Due to minor implementation differences, this version obtains 1% worse performance compared to the results reported in (Chen et al., 2017) when using the same training options. Following the suggestions of the authors, we use a smaller learning rate (0.001 instead of 0.002 for Adamax optimizer) and re-tune the dropout rates of word embeddings and RNNs. This gives us results comparable to the original paper.

All models are trained for a maximum of 50 epochs, batch size 32, a fixed learning rate of 0.001 and hidden dimension 128. We use a dropout of 0.5 for input word embeddings, 0.2 for SRU layers and 0.3 for LSTM layers.

Results Table 2 summarizes our results on SQuAD. LSTM models achieve 69.6% exact match and 78.9% F1 score, being on par with the results in the original work (Chen et al., 2017). SRU obtains better results than LSTM, getting 70.3% exact match and 79.5 F1 score. Moreover, SRU exhibits 6x to 10x speed-up and hence more than 69% reduction in total training time.

3.3 Language modeling

Dataset We use the Penn Treebank corpus (PTB) as the benchmark for language modeling. The processed data along with train, dev and test splits are taken from (Mikolov et al., 2010), which contains about 1 million tokens with a truncated vocabulary of 10k. Following standard practice, the training data is treated as a long sequence (split into a few chunks for mini-batch training), and hence the models are trained using truncated back-propagation-through-time (BPTT).

Setup Our training configuration largely follows prior work (Zaremba et al., 2014; Gal and Ghahramani, 2016; Zoph and Le, 2016). We use a batch size of 32 and truncated back-propagation with 35 steps. The dropout probability is 0.75 for the input embedding and output softmax layer. The standard dropout and variational dropout probability is 0.2 for stacked RNN layers. SGD with an initial learning rate of 1 and gradient clipping are used for optimization. We train a maximum of 300 epochs and start to decrease the learning rate by a factor of 0.98 after 175 epochs. We use the same configuration for models with different layers and hidden dimensions.

[4] `https://github.com/hitvoice/DrQA`

Model	# layers	Size	Dev	Test	Time / epoch	
					RNN	Total
LSTM (Zaremba et al., 2014)	2	66m	82.2	78.4		
LSTM (Press and Wolf, 2017)	2	51m	75.8	73.2		
LSTM (Inan et al., 2016)	2	28m	72.5	69.0		
RHN (Zilly et al., 2017)	10	23m	67.9	65.4		
KNN (Lei et al., 2017)	4	20m	-	63.8		
NAS (Zoph and Le, 2016)	-	25m	-	64.0		
NAS (Zoph and Le, 2016)	-	54m	-	62.4		
cuDNN LSTM	2	24m	73.3	71.4	53s	73s
cuDNN LSTM	3	24m	78.8	76.2	64s	79s
SRU	3	24m	68.0	64.7	21s	44s
SRU	4	24m	65.8	62.5	23s	44s
SRU	5	24m	63.9	61.0	27s	46s
SRU	6	24m	63.4	60.3	28s	47s

Table 3: Perplexities on PTB language modeling dataset. Models in comparison are trained using similar regularization and learning strategy: variational dropout is used except for (Zaremba et al., 2014), (Press and Wolf, 2017) and cuDNN LSTM; input and output word embeddings are tied except for (Zaremba et al., 2014); SGD with learning rate decaying is used for all models. Timings are performed on a desktop machine with a single Nvidia GeForce GTX 1070 GPU and Intel Core i7-7700K Processor.

Results Table 3 shows the results of our model and prior work. We use a parameter budget of 24 million for a fair comparison. `cuDNN LSTM` implementation obtains a perplexity of 71.4 at the speed of 73$\sim$79 seconds per epoch. The perplexity is worse than most of those numbers reported in prior work and we attribute this difference to the lack of variational dropout support in `cuDNN` implementation. In contrast, SRU obtains better perplexity compared to `cuDNN LSTM` and prior work, reaching 64.7 with 3 recurrent layers and 60.3 with 6 layers[5]. SRU also achieves better speed-perplexity trade-off, being able to run 47 seconds per epoch given 6 RNN layers.

3.4 Machine translation

Dataset We select WMT'14 English$\rightarrow$German translation task as our evaluation benchmark. Following standard practice (Peitz et al., 2014; Li et al., 2014; Jean et al., 2015), the training corpus was pre-processed and about 4 million translation pairs are left after processing. The news-test-2014 data is used as the test set and the concatenation of news-test-2012 and news-test-2013 data is used as the development set.

Setup We use OpenNMT (Klein et al., 2017), an open-source machine translation system for our experiments. We take the Pytorch version of this system[6] and extend it with our SRU implementation. The system trains a seq2seq model using a recurrent encoder-decoder architecture with attention (Luong et al., 2015). By default, the model feeds $\mathbf{h}_{t-1}$ (i.e. the hidden state of decoder at step $t-1$) as an additional input to the RNN decoder at step t. Although this can potentially improve translation quality, it also impedes parallelization and hence slows down the training procedure. We choose to disable this option unless otherwise specified. All models are trained with hidden and word embedding size 500, 15 epochs, SGD with initial learning rate 1.0 and batch size 64. Unlike OpenNMT's default setting, we use a smaller standard dropout rate of 0.1 and a weight decay of 10^{-5}. This leads to better results for both RNN implementations.

Results Table 4 presents the translation results. We obtain better BLEU scores compared to the results presented in the report of OpenNMT system (Klein et al., 2017). SRU with 10 stacking layers

[5]These results may be improved. As recently demonstrated by (Melis et al., 2017), the LSTM can achieve a perplexity of 58 via better regularization and hyper-parameter tuning. We leave this for future work.

[6]`https://github.com/OpenNMT/OpenNMT-py`

OpenNMT default setup	# layers	Size		Test BLEU	Time in RNNs
(Klein et al., 2017)	2	-	-	17.60	
(Klein et al., 2017) + BPE	2	-	-	19.34	
cuDNN LSTM (wd $= 0$)	2	85m	10m	18.04	149 min
cuDNN LSTM (wd $= 10^{-5}$)	2	85m	10m	19.99	149 min
Our setup					
cuDNN LSTM	2	84m	9m	19.67	46 min
cuDNN LSTM	3	88m	13m	19.85	69 min
cuDNN LSTM	5	96m	21m	20.45	115 min
SRU	3	81m	6m	18.89	12 min
SRU	5	84m	9m	19.77	20 min
SRU	6	85m	10m	20.17	24 min
SRU	10	91m	16m	20.70	40 min

Table 4: English-German translation results using OpenNMT system. We show the total number of parameters and the number excluding word embeddings. Our setup disables $\mathbf{h}_{t-1}$ feeding (i.e. `-input_feed 0`), which significantly reduces the training time. Adding one LSTM layer in encoder and decoder costs an additional 23 min in a training epoch, while SRU costs 4 min. Timings are performed on a single Nvidia Titan X Pascal GPU.

achieves a BLEU score of 20.7 while `cuDNN LSTM` achieves 20.45 using more parameters and more training time. Our implementation is also more scalable: a SRU layer in encoder and decoder adds only 4 min per training epoch. In comparison, the rest of the operations (e.g. attention and softmax output) costs about 95 min and a LSTM layer costs 23 min per epoch. As a result, we can easily stack many layers of SRU without increasing much of the training time. During our experiments, we do not observe an over-fitting on the dev set even using 10 layers.

3.5 Speech recognition

Dataset We use Switchboard-1 corpus (Godfrey et al., 1992) for our experiments. 4,870 sides of conversations (about 300 hours speech) from 520 speakers are used as training data, and 40 sides of Switchboard-1 conversations (about 2 hours speech) from the 2000 Hub5 evaluation are used as testing data.

Setup We use Kaldi (Povey et al., 2011) for feature extraction, decoding, and training of initial HMM-GMM models. Maximum likelihood-criterion context-dependent speaker adapted acoustic models with Mel-Frequency Cepstral Coefficient (MFCC) features are trained with standard Kaldi recipes. Forced alignment is performed to generate labels for neural network acoustic model training.

For speech recognition task, we use Computational Network Toolkit (CNTK) (Yu et al., 2014) instead of PyTorch for neural network training. Following (Sainath et al., 2015), all weights are randomly initialized from the uniform distribution with range $[-0.05, 0.05]$, and all biases are initialized to 0 without generative or discriminative pretraining (Seide et al., 2011). All neural network models, unless explicitly stated otherwise, are trained with a cross-entropy (CE) criterion using truncated back-propagation-through-time (BPTT) (Williams and Peng, 1990) for optimization. No momentum is used for the first epoch, and a momentum of 0.9 is used for subsequent epochs (Zhang et al., 2015). $L2$ constraint regularization (Hinton et al., 2012) with weight 10^{-5} is applied.

To train the uni-directional model, we unroll 20 frames and use 80 utterances in each mini-batch. We also delayed the output of LSTM by 10 frames as suggested in (Sak et al., 2014) to add more context for LSTM. The ASR performance can be further improved by using bidirectional model and state-level Minimum Bayes Risk (sMBR) training (Kingsbury et al., 2012). To train the bidirectional model, the latency-controlled method described in (Zhang et al., 2015) was applied. We set $N_c = 80$ and $N_r = 20$ and processed 40 utterances simultaneously. To train the recurrent model with sMBR criterion (Kingsbury et al., 2012), we adopted the two-forward-pass method described in (Zhang et al., 2015), and processed 40 utterances simultaneously.

Model	# layers	# Parameters	**WER**	Speed*
LSTM	5	47M	11.9	10.0K
LSTM + Seq	5	47M	10.8	-
Bi-LSTM	5	60M	11.2	5.0K
Bi-LSTM + Seq	5	60M	10.4	-
LSTM with highway (remove $\mathbf{h}$)	12	56M	12.5	6.5K
LSTM with highway	12	56M	12.2	4.6K
SRU	12	56M	11.6	12.0K
SRU + sMBR	12	56M	10.0	-
Bi-SRU	12	74M	10.5	6.2K
Bi-SRU + sMBR	12	74M	**9.5**	-
Very Deep CNN + sMBR (Saon et al., 2016)	10		10.5	-
LSTM + LF-MMI (Povey et al., 2016)	3		10.3	-
Bi-LSTM + LF-MMI (Povey et al., 2016)	3		9.6	-

Table 5: WER of different neural models. *Note the speed numbers reported here are based on a naive implementation of SRU in CNTK. No CUDA-level optimizations are performed.

The input features for all models are 80-dimensional log Mel filterbank features computed every 10 ms, with an additional 3-dimensional pitch features unless explicitly stated. The output targets are 8802-context-dependent triphone states, of which the numbers are determined by the last HMM-GMM training stage.

Results Table 5 summaries the results using SRU and other published results on SWBD corpus. We achieve state of the art results on this dataset. Note that LF-MMI for sequence training, i-vectors for speaker adaptation, and speaker perturbation for data augmentation have been applied in (Povey et al., 2016). All of these techniques can also been used for SRU. Moreover, we believe different highway variants such as grid LSTM (Hsu et al., 2016) can also further boost our model. If we also apply the same highway connection to LSTM, the performance is slightly worse than the baseline. Removing the dependency of $\mathbf{h}$ in LSTM can improve the speed but no gain for WER. Here we didn't use our customized kernel for SRU because CNTK has a special batching algorithm for RNNs. We can see without any kernel optimization, the SRU is already faster than LSTM using the same amount of parameters. More detailed experimental results about different highway structure and number of layers are refer to Appendix A.1.

4 Conclusion

This work presents Simple Recurrent Unit (SRU), a recurrent module that runs as fast as CNNs and scales easily to over 10 layers. We perform an extensive evaluation on NLP and speech recognition tasks, demonstrating the effectiveness of this recurrent unit. We open source our implementation to facilitate future NLP and deep learning research.

5 Acknowledgement

We thank Alexander Rush and Yoon Kim for their help on the machine translation experiments, and Danqi Chen for her help on SQuAD experiments. We also thank Adam Yala and Yoav Artzi for useful discussions and comments. A special thanks to Hugh Perkins for his support on the experimental environment setup, Runqi Yang for answering questions about his code, and the PyTorch community for enabling flexible neural module implementation.

References

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

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. In *ICLR*, 2017.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*, 2017.

Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 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, 2014.

Greg Diamos, Shubho Sengupta, Bryan Catanzaro, Mike Chrzanowski, Adam Coates, Erich Elsen, Jesse Engel, Awni Hannun, and Sanjeev Satheesh. Persistent rnns: Stashing recurrent weights on-chip. In *International Conference on Machine Learning*, pages 2024–2033, 2016.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016.

J. J. Godfrey, E. C. Holliman, and J. McDaniel. Switchboard: Telephone speech corpus for research and development. In *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 517–520, 1992.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. In *arXiv*, 2012.

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

W. Hsu, Y. Zhang, and J. Glass. A prioritized grid long short-term memory rnn for speech recognition. In *Proc. SLT*, 2016.

Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015.

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 2014.

Brian Kingsbury, Tara Sainath, and Hagen Soltau. Scalable Minimum Bayes Risk Training of Deep Neural Network Acoustic Models Using Distributed Hessian-free Optimization. In *INTERSPEECH*, 2012.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, 2017.

Kenton Lee, Omer Levy, and Luke Zettlemoyer. Recurrent additive networks. *arXiv preprint arXiv:1705.07393*, 2017.

Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. *ICML*, 2017.

Liangyou Li, Xiaofeng Wu, Santiago Cortes Vaillo, Jun Xie, Andy Way, and Qun Liu. The dcu-ictcas mt system at wmt 2014 on german-english translation task. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, 2014.

Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 2002.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 2010.

Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 115–124. Association for Computational Linguistics, 2005.

Stephan Peitz, Joern Wuebker, Markus Freitag, and Hermann Ney. The rwth aachen german-english machine translation system for wmt 2014. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, 2014.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannenmann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The Kaldi Speech Recognition Toolkit. In *Automatic Speech Recognition and Understanding Workshop*, 2011.

Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for asr based on lattice-free mmi. In *INTERSPEECH*, pages 2751–2755, 2016.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2017.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

Tara N. Sainath, Oriol Vinyals, Andrew Senior, and Hasim Sak. Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2015.

Hasim Sak, Andrew Senior, and Francoise Françoise. Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling. In *INTERSPEECH*, 2014.

George Saon, Tom Sercu, Steven Rennie, and Hong-Kwang J. Kuo. The ibm 2016 english conversational telephone speech recognition system. In *https://arxiv.org/abs/1604.08242*, 2016.

Frank Seide, Gang Li, Xie Chen, and Dong Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 24–29. IEEE, 2011.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, October 2013.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 2005.

Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

Huijia Wu, Jiajun Zhang, and Chengqing Zong. An empirical exploration of skip connections for sequential tagging. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, December 2016a.

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

D. Yu, A. Eversole, M. Seltzer, K. Yao, B. Guenter, O. Kuchaiev, F. Seide, H. Wang, J. Droppo, Z. Huang, Y. Zhang, G. Zweig, C. Rossbach, J. Currey, J. Gao, A. May, A. Stolcke, and M. Slaney. An introduction to computational networks and the computational network toolkit. Technical Report MSR, Microsoft Research, 2014. http://cntk.codeplex.com.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Yu Zhang, Dong Yu, Michael L Seltzer, and Jasha Droppo. Speech recognition with prediction-adaptation-correction recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5004–5008. IEEE, 2015.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A Additional results and analyses

A.1 Speech recognition

Baseline Table 6 compared different LSTM baseline model. For all the models, we follow the configurations reported in the paper Sak et al. (2014)[7]. We found that more than 5-layer will significantly increase the word error rate (WER). We can observed that our best LSTM model (with least parameters) has 5-layer and each layer contains 1024 memory cells. We will use it as a baseline model in Section 3.5.

Model	# layers	#Parameters	**WER**
LSTM with projection (Sak et al., 2014)	5	28M	12.2
LSTM	3	30M	12.5
LSTM (S)	5	28M	12.5
LSTM	5	47M	11.9
LSTM (L)	5	94M	12.0
LSTM	6	56M	12.3

Table 6: LSTM baseline on SWBD corpus. LSTM has 1024 cells for each layer. LSTM (S) has 750 cells for each layer. LSTM (L) has 1560 cells for each layers. LSTM with projection contains 1024 cells and a 512-node linear projection layer is added on top of each layer's output.

Effect of Highway Transform for SRU The dimensions of $\mathbf{x}_t$ and $\mathbf{h}_t$ must be equal in Eq. 2. If this is not the case (e.g., the first layer of the SRU), we can perform a linear projection $\mathbf{W}_h^l$ by the highway connections to match the dimensions at layer l:

$$\mathbf{h}'_t = \mathbf{r}_t \odot g(\mathbf{c}_t) + (1 - \mathbf{r}_t) \odot \mathbf{W}_h^l \mathbf{x}_t.$$

We can also use a square matrix $\mathbf{W}_h^l$ for every layer. As illustrated in Table 7, adding this transformation significantly improved the WER from 12.6% to 11.8% when we use the same amount of parameters. Note that all the highway transform are outside the recurrent loop, therefore the computation can be very efficient.

Model	# layers	#Parameters	**WER**
SRU (no $\mathbf{W}_h^l, l > 1$)	16	56M	12.6
SRU	12	56M	11.8

Table 7: Comparison of the effect of highway transform. SRU (no $\mathbf{W}_h^l, l > 1$) mean only add the transform in the first layer because of the dimension changed. SRU means adding the transform to every layer.

Effect of Depth for SRU Table 8 shows a comparison of the layers to different RNN models. It can be seen that the 10-layer SRU already outperform our best LSTM model using the same amount of parameters. The speed is also 1.4x faster even using a straight forward implementation in CNTK. [8] We can see without any kernel optimization, the SRU is already faster than LSTM because it require less "small" matrix multiplication. 12-layer SRU seems works best in this corpus which is also faster than the LSTM model.

[7] We removed the projection layer because we found the vanilla LSTM model give us better performance as illustrated in Table 6.

[8] Here we didn't use our customized SRU kernel because CNTK has some constraints to incorporate with it.

Model	# layers	#Parameters	**WER**	Speed
LSTM	5	47M	11.9	10.0K
SRU	10	47M	11.8	14.0K
SRU	12	56M	11.5	12.0K
SRU	16	72M	11.5	9.3K
SRU	20	89M	11.8	8.0K

Table 8: Comparison of the effect of the depth for SRU model.