A Lightweight Recurrent Network for Sequence Modeling

Biao Zhang[1] **Rico Sennrich**[1,2]

[1]School of Informatics, University of Edinburgh
B.Zhang@ed.ac.uk, rico.sennrich@ed.ac.uk

[2]Institute of Computational Linguistics, University of Zurich

Abstract

Recurrent networks have achieved great success on various sequential tasks with the assistance of complex recurrent units, but suffer from severe computational inefficiency due to weak parallelization. One direction to alleviate this issue is to shift heavy computations outside the recurrence. In this paper, we propose a lightweight recurrent network, or *LRN*. LRN uses input and forget gates to handle long-range dependencies as well as gradient vanishing and explosion, with all parameter-related calculations factored outside the recurrence. The recurrence in LRN only manipulates the weight assigned to each token, tightly connecting LRN with self-attention networks. We apply LRN as a drop-in replacement of existing recurrent units in several neural sequential models. Extensive experiments on six NLP tasks show that LRN yields the best running efficiency with little or no loss in model performance.[1]

1 Introduction

Various natural language processing (NLP) tasks can be categorized as sequence modeling tasks, where recurrent networks (RNNs) are widely applied and contribute greatly to state-of-the-art neural systems (Yang et al., 2018; Peters et al., 2018; Zhang et al., 2018; Chen et al., 2018; Kim et al., 2019). To avoid the optimization bottleneck caused by gradient vanishing and/or explosion (Bengio et al., 1994), Hochreiter and Schmidhuber (1997) and Cho et al. (2014) develop gate structures to ease information propagation from distant words to the current position. Nevertheless, integrating these traditional gates inevitably increases computational overhead which is accumulated along token positions due to the sequential nature of RNNs. As a result, the weak parallelization of RNNs makes the benefits from improved model capacity expensive in terms of computational efficiency.

Recent studies introduce different solutions to this issue. Zhang et al. (2018) introduce the addition-subtraction twin-gated recurrent unit (ATR), which reduces the amount of matrix operations by developing parameter-shared twin-gate mechanism. Lei et al. (2018) introduce the simple recurrent unit (SRU), which improves model parallelization by moving matrix computations outside the recurrence. Nevertheless, both ATR and SRU perform affine transformations of the previous hidden state for gates, though SRU employs a vector parameter rather than a matrix parameter. In addition, SRU heavily relies on its highway component, without which the recurrent component itself suffers from weak capacity and generalization (Lei et al., 2018).

In this paper, we propose a lightweight recurrent network (LRN), which combines the strengths of ATR and SRU. The structure of LRN is simple: an input gate and a forget gate are applied to weight the current input and previous hidden state, respectively. LRN has fewer parameters than SRU, and compared to ATR, removes heavy calculations outside the recurrence, generating gates based on the previous hidden state without any affine transformation. In this way, computation inside each recurrent step is highly minimized, allowing better parallelization and higher speed.

The gate structure endows LRN with the capability of memorizing distant tokens as well as handling the gradient vanishing and explosion issue. This ensures LRN's expressiveness and performance on downstream tasks. In addition, decomposing its recurrent structure discovers the correlation of input/forget gate with key/query in self-attention networks (Vaswani et al., 2017), where

[1]Source code is available at https://github.com/bzhangGo/lrn.

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 1538–1548
Florence, Italy, July 28 - August 2, 2019.

these two gates together manipulate the weight assigned to each token. We also reveal how LRN manages long-term and short-term memories with the decomposition.

We carry out extensive experiments on six NLP tasks, ranging from natural language inference, document classification, machine translation, question answering and part-of-speech tagging to language modeling. We use LRN as a drop-in replacement of existing recurrent units in different neural models without any other modification of model structure. Experimental results show that LRN outperforms SRU by 10%∼20% in terms of running speed, and is competitive with respect to performance and generalization compared against all existing recurrent units.

2 Related Work

Past decades have witnessed the rapid development of RNNs since the Elman structure was proposed (Elman, 1990). Bengio et al. (1994) point out that the gradient vanishing and explosion issue impedes the optimization and performance of RNNs. To handle this problem, Hochreiter and Schmidhuber (1997) develop LSTM where information and gradient from distant tokens can successfully pass through the current token via a gate structure and a memory cell. Unfortunately, the enhanced expressivity via complex gates comes at the cost of sacrificing computational efficiency, which becomes more severe when datasets are scaled up. Simplifying computation but keeping model capacity in RNNs raises a new challenge.

One direction is to remove redundant structures in LSTM. Cho et al. (2014) remove the memory cell and introduce the gated recurrent unit (GRU) with only two gates. Lee et al. (2017) introduce an additive structure to generate hidden representations with linear transformed inputs directly, though we empirically observe that non-linear activations can stabilize model training. Zhang et al. (2018) propose a twin-gate mechanism where input and forget gate are simultaneously produced from the same variables. We extend this mechanism by removing the affine transformation of previous hidden states.

Another direction is to shift recurrent matrix multiplications outside the recurrence so as to improve the parallelization of RNNs. Bradbury et al. (2016) propose the quasi-recurrent network (QRNN). QRNN factors all matrix multiplications out of the recurrence and employs a convolutional network to capture local input patterns. A minimal recurrent pooling function is used in parallel across different channels to handle global input patterns. Lei et al. (2017) apply the kernel method to simplify recurrence and show improved model capacity with deep stacked RNNs. This idea is extended to SRU (Lei et al., 2018) where a minimal recurrent component is strengthened via an external highway layer. The proposed LRN falls into this category with the advantage over SRU of the non-dependence on the highway component.

Orthogonal to the above work, recent studies also show the potential of accelerating matrix computation with low-level optimization. Diamos et al. (2016) emphasize persistent computational kernels to exploit GPU's inverted memory hierarchy for reusing/caching purpose. Appleyard et al. (2016) upgrade NIVIDIA's cuDNN implementation through exposing parallelism between operations within the recurrence. Kuchaiev and Ginsburg (2017) reduce the number of model parameters by factorizing or partitioning LSTM matrices. In general, all these techniques can be applied to any recurrent units to reduce computational overhead.

Our work is closely related with ATR and SRU. Although recent work shows that novel recurrent units derived from weighted finite state automata are effective without the hidden-to-hidden connection (Balduzzi and Ghifary, 2016; Peng et al., 2018), we empirically observe that including previous hidden states for gates is crucial for model capacity which also resonates with the evolution of SRU. Unlike ATR and SRU, however, we demonstrate that the affine transformation on the previous hidden state for gates is unnecessary. In addition, our model has a strong connection with self-attention networks.

3 Lightweight Recurrent Network

Given a sequence of input $\mathbf{X} = [\mathbf{x}_1^\intercal; \mathbf{x}_2^\intercal; \ldots; \mathbf{x}_n^\intercal] \in \mathbb{R}^{n \times d}$ $\mathbb{R}^{n \times d}$ with length of n, LRN operates as follows[2]:

$$\mathbf{Q}, \mathbf{K}, \mathbf{V} = \mathbf{X}\mathbf{W}_g, \mathbf{X}\mathbf{W}_k, \mathbf{X}\mathbf{W}_v \quad (1)$$

$$\mathbf{i}_t = \sigma(\mathbf{k}_t + \mathbf{h}_{t-1}) \in \mathbb{1}\times d \quad (2)$$

$$\mathbf{f}_t = \sigma(\mathbf{q}_t - \mathbf{h}_{t-1}) \in \mathbb{1}\times d \quad (3)$$

$$\mathbf{h}_t = g(\mathbf{i}_t \odot \mathbf{v}_t + \mathbf{f}_t \odot \mathbf{h}_{t-1}) \in \mathbb{1}\times d \quad (4)$$

[2]Bias terms are removed for clarity.

where $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d\times d}$ are model parameters and $g(\cdot)$ is an activation function, such as *identity* and *tanh*. $\odot$ and $\sigma(\cdot)$ indicate the element-wise multiplication and sigmoid activation function, respectively. $\mathbf{q}_t, \mathbf{k}_t$ and $\mathbf{v}_t$ correspond to the t-th row of the projected sequence representation $\mathbf{Q}, \mathbf{K}, \mathbf{V}$. We use the term q, k and v to denote the implicit correspondence to *query*, *key* and *value* in self-attention networks which is elaborated in the next section.

As shown in Eq. (1), all matrix-related operations are shifted outside the recurrence and can be pre-calculated, thereby reducing the complexity of the recurrent computation from $\mathcal{O}(d^2)$ to $\mathcal{O}(d)$ and easing model parallelization. The design of the input gate $\mathbf{i}_t$ and forget gate $\mathbf{f}_t$ is inspired by the twin-gate mechanism in ATR (Zhang et al., 2018). Unlike ATR, however, we eschew the affine transformation on the previous hidden state. By doing so, the previous hidden state directly offers positive contribution to the input gate but negative to the forget gate, ensuring adverse correlation between these two gates.

The current hidden state $\mathbf{h}_t$ is a weighted average of the current input and the previous hidden state followed by an element-wise activation. When *identity* function is employed, our model shows analogous properties to ATR. However, we empirically observe that this leads to gradually increased hidden representation values, resulting in optimization instability. Unlike SRU, which controls stability through a particular designed scaling term, we replace the *identity* function with the *tanh* function, which is simple but effective.

4 Structure Decomposition

In this section, we show an in-depth analysis of LRN by decomposing the recurrent structure. With an *identity* activation, the t-th hidden state can be expanded as follows:

$$\mathbf{h}_t = \sum_{k=1}^{t} \mathbf{i}_k \odot \left(\prod_{l=1}^{t-k} \mathbf{f}_{k+l} \right) \odot \mathbf{v}_k, \qquad (5)$$

where the representation of the current token is composed of all previous tokens with their contribution distinguished by both input and forget gates.

Relation with self-attention network. After grouping these gates, we observe that:

$$\mathbf{h}_t = \sum_{k=1}^{t} \underbrace{\mathbf{i}_k}_{\text{key}(\mathbf{K})} \odot \underbrace{\mathbf{f}_{k+1} \odot \cdots \odot \mathbf{f}_t}_{\text{query}(\mathbf{Q})} \odot \underbrace{\mathbf{v}_k}_{\text{value}(\mathbf{V})}. \qquad (6)$$

Each weight can be regarded as a query from the current token $\mathbf{f}_t$ to the k-th input token $\mathbf{i}_k$. This query chain can be decomposed into two parts: a *key* represented by $\mathbf{i}_k$ and a *query* represented by $\prod_{l=1}^{t-k} \mathbf{f}_{k+l}$. The former is modulated through the weight matrix $\mathbf{W}_k$, and tightly associated with the corresponding input token. Information carried by the key remains intact during the evolution of time step t. In contrast, the latter, induced by the weight matrix $\mathbf{W}_q$, highly depends on the position and length of this chain, which dynamically changes between different token pairs.

The weights generated by keys and queries are assigned to *value*s represented by $\mathbf{v}_k$ and manipulated by the weight matrix $\mathbf{W}_v$. Compared with self-attention networks, LRN shows analogous weight parameters and model structure. The difference is that weights in self-attention networks are normalized across all input tokens. Instead, weights in LRN are unidirectional, unnormalized and spanned over all channels.

Memory in LRN Alternatively, we can view the gating mechanism in LRN as a memory that gradually forgets information.

Given the value representation at k-th time step $\mathbf{v}_k$, the information delivered to later time step t ($k < t$) in LRN is as follows:

$$\underbrace{\mathbf{i}_k}_{\text{short term}} \odot \underbrace{\mathbf{f}_{k+1} \odot \cdots \odot \mathbf{f}_t}_{\text{forget chain (long term)}} \odot \mathbf{v}_k. \qquad (7)$$

The input gate $\mathbf{i}_k$ indicates the moment that LRN first accesses the input token $\mathbf{x}_k$, whose value reflects the amount of information or knowledge allowed from this token. A larger input gate corresponds to a stronger input signal, thereby a large change of activating short-term memory. This information is then delivered through a forget chain where memory is gradually decayed by a forget gate at each time step. The degree of memory decaying is dynamically controlled by the input sequence itself. When a new incoming token is more informative, the forget gate would increase so that previous knowledge is erased so as to make way for new knowledge in the memory. By contrast, meaningless tokens would be simply ignored.

Model		#Params	Base		+LN		+BERT		+LN+BERT	
			ACC	Time	ACC	Time	ACC	Time	ACC	Time
Rocktäschel et al. (2016)		250K	83.50	-	-	-	-	-	-	-
This Work	LSTM	8.36M	84.27	0.262	86.03	0.432	89.95	0.544	**90.49**	0.696
	GRU	6.41M	**85.71**	0.245	**86.05**	0.419	**90.29**	0.529	90.10	0.695
	ATR	2.87M	84.88	0.210	85.81	0.307	90.00	0.494	90.28	0.580
	SRU	5.48M	84.28	0.258	85.32	0.283	89.98	0.543	90.09	0.555
	LRN	4.25M	84.88	**0.209**	85.06	**0.223**	89.98	**0.488**	89.93	**0.506**

Table 1: Test accuracy (ACC) on SNLI task. "#Params": the parameter number of Base. *Base* and *LN* denote the baseline model and layer normalization respectively. *Time*: time in seconds per training batch measured from 1k training steps on GeForce GTX 1080 Ti. Best results are highlighted in bold.

5 Gradient Analysis

Gradient vanishing and explosion are the bottleneck that impedes training of vanilla RNNs (Pascanu et al., 2013). Consider a vanilla RNN formulated as follows:

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1}). \tag{8}$$

The gradient back-propagated from the t-th step heavily depends on the following one-step derivation:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{U}^T g'. \tag{9}$$

BPTT的梯度消失/爆炸是因为连乘U导致的.

Due to the *chain rule*, the recurrent weight matrix $\mathbf{U}$ will be repeatedly multiplied along the sequence length. Gradient vanishing/explosion results from a weight matrix with small/large norm (Pascanu et al., 2013).

In LRN, however, the recurrent weight matrix is removed. The current hidden state is generated by directly weighting the current input and the previous hidden state. The one-step derivation of Eq. (2-4) is as follows:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \underbrace{\left(\sigma_i' \odot \mathbf{v}_t - \mathbf{h}_{t-1} \odot \sigma_f' + \mathbf{f}_t\right)}_{\mathbf{A}} \odot g' \tag{10}$$

A的诸数依赖于输入.
这种依赖性避免了梯度消失/爆炸.

where σ_i' and σ_f' denote the derivation of Eq. (2) and Eq. (3), respectively. The difference between Eq. (9) and Eq. (10) is that the recurrent weight matrix is substituted by a more expressive component denoted as $\mathbf{A}$ in Eq. (10). Unlike the weight matrix $\mathbf{U}$, the norm of $\mathbf{A}$ is input-dependent and varies dynamically along different positions. The dependence on inputs provides LRN with the capability of avoiding gradient vanishing/explosion.

6 Experiments

We verify the effectiveness of LRN on six diverse NLP tasks. For each task, we adopt (near) state-of-the-art neural models with RNNs handling sequence representation. We compare LRN with several cutting-edge recurrent units, including LSTM, GRU, ATR and SRU. For all comparisons, we keep the neural architecture intact and only alter the recurrent unit.[3] All RNNs are implemented without specialized cuDNN kernels. Unless otherwise stated, different models on the same task share the same set of hyperparameters.

6.1 Natural Language Inference

Settings Natural language inference reasons about the entailment relationship between a premise sentence and a hypothesis sentence. We use the Stanford Natural Language Inference (SNLI) corpus (Bowman et al., 2015) and treat the task as a three-way classification task. This dataset contains 549,367 premise-hypothesis pairs for training, 9,842 pairs for developing and 9,824 pairs for testing. We employ accuracy for evaluation.

We implement a variant of the word-by-word attention model (Rocktäschel et al., 2016) using Tensorflow for this task, where we stack two additional bidirectional RNNs upon the final sequence representation and incorporate character embedding for word-level representation. The pretrained GloVe (Pennington et al., 2014) word vectors are used to initialize word embedding. We also integrate the base BERT (Devlin et al., 2018) to improve contextual modeling.

[3] Due to possible dimension mismatch, we include an additional affine transformation on the input matrix for the highway component in SRU. In addition, we only report and compare speed statistics when all RNNs are optimally implemented where computations that can be done before the recurrence are moved outside.

Model		#Params	AmaPolar		Yahoo		AmaFull		YelpPolar	
			ERR	Time	ERR	Time	ERR	Time	ERR	Time
Zhang et al. (2015)		-	6.10	-	29.16	-	40.57	-	5.26	-
This Work	LSTM	227K	**4.37**	0.947	**24.62**	1.332	37.22	1.003	3.58	1.362
	GRU	176K	4.39	0.948	24.68	1.242	**37.20**	0.982	**3.47**	1.230
	ATR	74K	4.78	0.867	25.33	1.117	38.54	0.836	4.00	1.124
	SRU	194K	4.95	0.919	24.78	1.394	38.23	0.907	3.99	1.310
	LRN	151K	4.98	**0.731**	25.07	**1.038**	38.42	**0.788**	3.98	**1.022**

Table 2: Test error (ERR) on document classification task. "#Params": the parameter number in AmaPolar task. *Time*: time in seconds per training batch measured from 1k training steps on GeForce GTX 1080 Ti.

We set the character embedding size and the RNN hidden size to 64 and 300 respectively. Dropout is applied between consecutive layers with a rate of 0.3. We train models within 10 epochs using the Adam optimizer (Kingma and Ba, 2014) with a batch size of 128 and gradient norm limit of 5.0. We set the learning rate to $1e^{-3}$, and apply an exponential moving average to all model parameters with a decay rate of 0.9999. These hyperparameters are tuned according to development performance.

Results Table 1 shows the test accuracy and training time of different models. Our implementation outperforms the original model where Rocktäschel et al. (2016) report an accuracy of 83.50. Overall results show that LRN achieves competitive performance but consumes the least training time. Although LSTM and GRU outperform LRN by 0.3∼0.9 in terms of accuracy, these recurrent units sacrifice running efficiency (about 7%∼48%) depending on whether LN and BERT are applied. No significant performance difference is observed between SRU and LRN, but LRN has fewer model parameters and shows a speedup over SRU of 8%∼21%.

Models with layer normalization (LN) (Ba et al., 2016) tend to be more stable and effective. However, for LSTM, GRU and ATR, LN results in significant computational overhead (about 27%∼71%). In contrast, quasi recurrent models like SRU and LRN only suffer a marginal speed decrease. This is reasonable because layer normalization is moved together with matrix multiplication out of the recurrence.

Results with BERT show that contextual information is valuable for performance improvement. LRN obtains additional 4 percentage points gain with BERT and reaches an accuracy of around 89.9. This shows the compatibility of LRN with existing pretrained models. In addition, although the introduction of BERT brings in heavy matrix computation, the benefits from LRN do not disappear. LRN is still the fastest model, outperforming other recurrent units by 8%∼27%.

6.2 Document Classification

Settings Document classification poses challenges in the form of long-range dependencies where information from distant tokens that contribute to the correct category should be captured. We use Amazon Review Polarity (AmaPolar, 2 labels, 3.6M/0.4M for training/testing), Amazon Review Full (AmaFull, 5 labels, 3M/0.65M for training/testing), Yahoo! Answers (Yahoo, 10 labels, 1.4M/60K for training/testing) and Yelp Review Polarity (YelpPolar, 2 labels, 0.56M/38K for training/testing) from Zhang et al. (2015) for experiments. We randomly select 10% of training data for validation. Models are evaluated by test error.

We treat a document as a sequence of words. Our model is a bidirectional RNN followed by an attentive pooling layer. The word-level representation is composed of a pretrained GloVe word vector and a convolutional character vector. We use Tensorflow for implementation and do not use layer normalization. We set character embedding size to 32, RNN hidden size to 64 and dropout rate to 0.1. Model parameters are tuned by Adam optimizer with initial learning rate of $1e^{-3}$. Gradients are clipped when their norm exceeds 5. We limit the maximum document length to 400 and maximum training epochs to 6. Parameters are smoothed by an exponential moving average with a decay rate of 0.9999. These hyperparameters are tuned according to development performance.

Results Table 2 summarizes the classification results. LRN achieves comparable classification performance against ATR and SRU, but slightly

Model	#Params	BLEU	Train	Decode
GNMT	-	24.61	-	-
GRU	206M	26.28	2.67	45.35
ATR	122M	25.70	1.33	**34.40**
SRU	170M	25.91	1.34	42.84
LRN	143M	26.26	**0.99**	36.50
oLRN	164M	**26.73**	1.15	40.19

Table 3: Case-insensitive tokenized BLEU score on WMT14 English-German translation task. *Train*: time in seconds per training batch measured from 0.2k training steps on Tesla P100. *Decode*: time in milliseconds used to decode one sentence measured on newstest2014 dataset.

Model	#Params	Base	+Elmo
rnet*	-	71.1/79.5	-/-
LSTM	2.67M	**70.46**/78.98	75.17/82.79
GRU	2.31M	70.41/**79.15**	75.81/83.12
ATR	1.59M	69.73/78.70	75.06/82.76
SRU	2.44M	69.27/78.41	74.56/82.50
LRN	2.14M	70.11/78.83	**76.14/83.83**

Table 4: Exact match/F1-score on SQuad dataset. "#Params": the parameter number of Base. *rnet**: results published by Wang et al. (2017).

underperforms LSTM and GRU (-0.45∼-1.22). This indicates that LRN is capable of handling long-range dependencies though not as strong as complex recurrent units. Instead, the simplification endows LRN with less computational overhead than these units. Particularly, LRN accelerates the training over LSTM and SRU by about 20%, or several days of training time on GeForce GTX 1080 Ti.[4]

6.3 Machine Translation

Settings Machine translation is the task of transforming meaning from a source language to a target language. We experiment with the WMT14 English-German translation task (Bojar et al., 2014) which consists of 4.5M training sentence pairs.[5] We use newstest2013 as our development set and newstest2014 as our test set. Case-sensitive tokenized BLEU score is used for evaluation.

We implement a variant of the GNMT system (Wu et al., 2016) using Tensorflow, enhanced with residual connections, layer normalization, label smoothing, a context-aware component (Zhang et al., 2017) and multi-head attention (Vaswani et al., 2017). Byte-pair encoding (Sennrich et al., 2016) is used to reduce the vocabulary size to 32K. We set the hidden size and embedding size to 1024. Models are trained using Adam optimizer with adaptive learning rate schedule (Chen et al., 2018). We cut gradient norm to 1.0 and set the token size to 32K. Label smoothing rate is set to 0.1.

Model Variant Apart from LRN, we develop an improved variant for machine translation that includes an additional output gate. Formally, we change the Eq. (4) to the following one:

$$\mathbf{c}_t = \mathbf{i}_t \odot \mathbf{v}_t + \mathbf{f}_t \odot \mathbf{h}_{t-1} \quad (11)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t - \mathbf{c}_t) \quad (12)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t \quad (13)$$

We denote this variant *oLRN*. Like LRN, the added matrix transformation in oLRN can be shifted out of the recurrence, bringing in little computational overhead. The design of this output gate $\mathbf{o}_t$ is inspired by the LSTM structure, which acts as a controller to adjust information flow. In addition, this gate helps stabilize the hidden activation to avoid value explosion, and also improves model fitting capacity.

Results The results in Table 3 show that translation quality of LRN is slightly worse than that of GRU (-0.02 BLEU). After incorporating the output gate, however, oLRN yields the best BLEU score of 26.73, outperforming GRU (+0.45 BLEU). In addition, the training time results in Table 3 confirm the computational advantage of LRN over all other recurrent units, where LRN speeds up over ATR and SRU by approximately 25%. For decoding, nevertheless, the autoregressive schema of GNMT disables position-wise parallelization. In this case, the recurrent unit with the least computation operations, i.e. ATR, becomes the fastest. Still, both LRN and oLRN translate sentences faster than SRU (+15%/+6%).

6.4 Reading Comprehension

Settings Reading comprehension aims at providing correct answers to a query based on a

[4] We notice that ATR operates faster than SRU. This is because though in theory SRU can be highly optimized for parallelization, computational framework like Tensorflow can not handle it automatically and the smaller amount of calculation in ATR has more advantage in practice.

[5] Preprocessed data is available at (Zhang et al., 2018): https://drive.google.com/open?id=15WRLfle66CO1zIGKbyz0FsFmUcINyb4X.

Model		#Params	PTB			WT2		
			Base	+Finetune	+Dynamic	Base	+Finetune	+Dynamic
Yang et al. (2018)		22M	55.97	54.44	47.69	63.33	61.45	40.68
This Work	LSTM	22M	63.78	62.12	**53.11**	**69.78**	**68.68**	**44.60**
	GRU	17M	69.09	67.61	60.21	73.37	73.05	49.77
	ATR	9M	66.24	65.86	58.29	75.36	73.35	48.65
	SRU	13M	69.64	65.29	60.97	85.15	84.97	57.97
	LRN	11M	**61.26**	**61.00**	54.45	69.91	68.86	46.97

Table 5: Test perplexity on PTB and WT2 language modeling task. "#Params": the parameter number in PTB task. *Finetune*: fintuning the model after convergence. *Dynamic* dynamic evaluation. Lower perplexity indicates better performance.

Model	#Params	NER
LSTM*	-	90.94
LSTM	245K	**89.61**
GRU	192K	89.35
ATR	87K	88.46
SRU	161K	88.89
LRN	129K	88.56

Table 6: F1 score on CoNLL-2003 English NER task. "#Params": the parameter number in NER task. *LSTM** denotes the reported result (Lample et al., 2016).

given document, which involves complex sentence matching, reasoning and knowledge association. We use the SQuAD corpus (Rajpurkar et al., 2016) for this task and adopt span-based extraction method. This corpus contains over 100K document-question-answer triples. We report exact match (EM) and F1-score (F1) on the development set for evaluation.

We employ the public available rnet model (Wang et al., 2017)[6] in Tensorflow. We use the default model settings: character embedding size 8, hidden size 75, batch size 64, and Adadelta optimizer (Zeiler, 2012) with initial learning rate of 0.5. Gradient norm is cut to 5.0. We also experiment with Elmo (Peters et al., 2018), and feed the Elmo representation in before the encoding layer and after the matching layer with a dropout of 0.5.

Results Table 4 lists the EM/F1 score of different models. In this task, LRN outperforms ATR and SRU in terms of both EM and F1 score. After integrating Elmo for contextual modeling, the performance of LRN reaches the best (76.14 EM and 83.83 F1), beating both GRU and LSTM (+0.33EM, +0.71F1). As recent studies show that cases in SQuAD are dominated by local pattern matching (Jia and Liang, 2017), we argue that LRN is good at handling local dependencies.

6.5 Named Entity Recognition

Settings Named entity recognition (NER) classifies named entity mentions into predefined categories. We use the CoNLL-2003 English NER dataset (Tjong Kim Sang and De Meulder, 2003) and treat NER as a sequence labeling task. We use the standard train, dev and test split. F1 score is used for evaluation.

We adopt the bidirectional RNN with CRF inference architecture (Lample et al., 2016). We implement different models based on the public codebase in Tensorflow.[7] We use the default hyperparameter settings. Word embedding is initialized by GloVe vectors.

Results As shown in Table 6[8], the performance of LRN matches that of ATR and SRU, though LSTM and GRU operate better (+1.05 and +0.79). As in the SQuAD task, the goal of NER is to detect local entity patterns and figure out the entity boundaries. However, the performance gap between LSTM/GRU and LRN in NER is significantly larger than that in SQuAD. We ascribe this to the weak model architecture and the small scale NER dataset where entity patterns are not fully captured by LRN.

6.6 Language Modeling

Settings Language modeling aims to estimate the probability of a given sequence, which re-

[6] https://github.com/HKUST-KnowComp/R-Net

[7] https://github.com/Hironsan/anago

[8] Notice that our implementation falls behind the original model (Lample et al., 2016) because we do not use specifically trained word embedding.

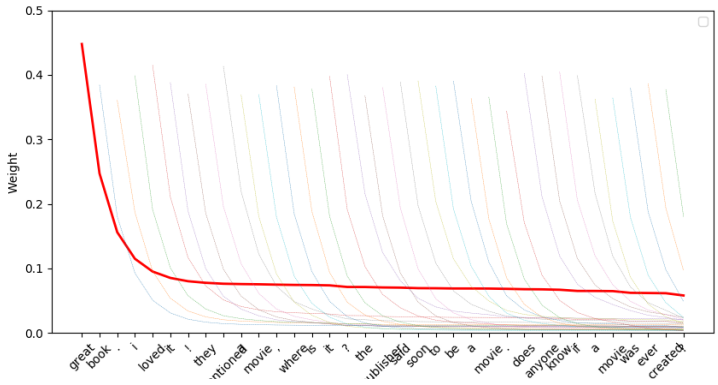

Figure 1: The decay curve of each token modulated by input and forget gates along the token position. Notice how the memory of term "*great*" flows to the final state shown in red, and contributes to a *Positive* decision. *Weight* denotes the averaged activation of $\mathbf{i}_k \odot \left(\prod_{l=1}^{t-k} \mathbf{f}_{k+l}\right)$ as shown in Eq. (5).

quires models to memorize long-term structure of language. We use two widely used datasets, Penn Treebank (PTB) (Mikolov et al., 2010) and WikiText-2 (WT2) (Merity et al., 2016) for this task. Models are evaluated by perplexity.

We modify the mixture of softmax model (MoS) (Yang et al., 2018)[9] in PyTorch to include different recurrent units. We apply weight dropout to all recurrent-related parameters instead of only hidden-to-hidden connection. We follow the experimental settings of MoS, and manually tune the initial learning rate based on whether training diverges.

Results Table 5 shows the test perplexity of different models.[10] In this task, LRN significantly outperforms GRU, ATR and SRU, and achieves near the same perplexity as LSTM. This shows that in spite of its simplicity, LRN can memorize long-term language structures and capture a certain degree of language variation. In summary, LRN generalizes well to different tasks and can be used as a drop-in replacement of existing recurrent units.

6.7 Ablation Study

Part of LRN can be replaced with some alternatives. In this section, we conduct ablation analysis to examine two possible designs:

gLRN The twin-style gates in Eq. (2-3) can be replaced with a general one:

$$\mathbf{f}_t = \sigma(\mathbf{q}_t - \mathbf{h}_{t-1}), \mathbf{i}_t = 1 - \mathbf{f}_t. \quad (14)$$

In this way, input and forget gate are inferable from each other with the key weight parameter removed.

eLRN The above design can be further simplified into an extreme case where the forget gate is only generated from the previous hidden state without the query vector:

$$\mathbf{f}_t = \sigma(-\mathbf{h}_{t-1}), \mathbf{i}_t = 1 - \mathbf{f}_t. \quad (15)$$

Model	SNLI	PTB
LRN	**85.06**	**61.26**
gLRN	84.72	92.49
eLRN	83.56	169.81

Table 7: Test accuracy on SNLI task with *Base+LN* setting and test perplexity on PTB task with *Base* setting.

We experiment with SNLI and PTB tasks. Results in Table 7 show that although the accuracy on SNLI is acceptable, gLRN and eLRN perform significantly worse on the PTB task. This suggests that these alternative structures suffer from weak generalization.

6.8 Structure Analysis

In this section, we provide a visualization to check how the gates work in LRN.

We experiment with a unidirectional LRN on the AmaPolar dataset, where the last hidden state

[9] https://github.com/zihangdai/mos

[10] Our re-implementation of LSTM model is worse than the original model (Yang et al., 2018) because the system is sensitive to hyperparameters, and we apply weight dropout to all LSTM parameters which makes the original best choices not optimal.

is used for document classification. Figure 1 shows the decay curve of each token along the token position. The memory curve of each token decays over time. However, important clues that contribute significantly to the final decision, as the token "*great*" does, decrease slowly, as shown by the red curve. Different tokens show different decay rate, suggesting that input and forget gate are capable of learning to propagate relevant signals. All these demonstrate the effectiveness of our LRN model.

7 Conclusion and Future Work

This paper presents LRN, a lightweight recurrent network that factors matrix operations outside the recurrence and enables higher parallelization. Theoretical and empirical analysis shows that the input and forget gate in LRN can learn long-range dependencies and avoid gradient vanishing and explosion. LRN has a strong correlation with self-attention networks. Experiments on six different NLP tasks show that LRN achieves competitive performance against existing recurrent units. It is simple, effective and reaches better trade-off among parameter number, running speed, model performance and generalization.

In the future, we are interested in testing low-level optimizations of LRN, which are orthogonal to this work, such as dedicated cuDNN kernels.

Acknowledgments

We thank the reviewers for their insightful comments. Biao Zhang acknowledges the support of the Baidu Scholarship. This work has been performed using resources provided by the Cambridge Tier-2 system operated by the University of Cambridge Research Computing Service (http://www.hpc.cam.ac.uk) funded by EPSRC Tier-2 capital grant EP/P020259/1.

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*.

David Balduzzi and Muhammad Ghifary. 2016. Strongly-typed recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1292–1300. JMLR.org.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2016. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*.

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

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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 *International Conference on Machine Learning*, pages 2024–2033.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

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

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031. Association for Computational Linguistics.

Seonhoon Kim, Jin-Hyuk Hong, Inho Kang, and Nojun Kwak. 2019. Semantic sentence matching with densely-connected recurrent and co-attentive information. *AAAI19*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Oleksii Kuchaiev and Boris Ginsburg. 2017. Factorization tricks for lstm networks. *arXiv preprint arXiv:1703.10722*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270. Association for Computational Linguistics.

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

T. Lei, W. Jin, R. Barzilay, and T. Jaakkola. 2017. Deriving neural architectures from sequence and graph kernels. In *International Conference on Machine Learning (ICML)*.

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*, pages 4470–4481. Association for Computational Linguistics.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318.

Hao Peng, Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. Rational recurrences. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1214. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392. Association for Computational Linguistics.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomas Kocisky, and Phil Blunsom. 2016. Reasoning about entailment with neural attention. In *International Conference on Learning Representations (ICLR)*.

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

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 142–147, Stroudsburg, PA, USA. Association for Computational Linguistics.

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

Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. 2017. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 189–198. Association for Computational Linguistics.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus

Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, ukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. 2018. Breaking the softmax bottleneck: A high-rank rnn language model. *ICLR*.

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

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

Biao Zhang, Deyi Xiong, and Jinsong Su. 2018. Neural machine translation with deep attention. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1.

Biao Zhang, Deyi Xiong, Jinsong Su, and Hong Duan. 2017. A context-aware recurrent encoder for neural machine translation. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 25(12):2424–2432.

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

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 649–657, Cambridge, MA, USA. MIT Press.