The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Lattice Recurrent Unit: Improving Convergence and Statistical Efficiency for Sequence Modeling

Chaitanya Ahuja
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
cahuja@andrew.cmu.edu

Louis-Philippe Morency
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
morency@cs.cmu.edu

Abstract

Recurrent neural networks have shown remarkable success in modeling sequences. However low resource situations still adversely affect the generalizability of these models. We introduce a new family of models, called Lattice Recurrent Units (LRU), to address the challenge of learning deep multi-layer recurrent models with limited resources. LRU models achieve this goal by creating distinct (but coupled) flow of information inside the units: a first flow along time dimension and a second flow along depth dimension. It also offers a symmetry in how information can flow horizontally and vertically. We analyze the effects of decoupling three different components of our LRU model: *Reset Gate*, *Update Gate* and *Projected State*. We evaluate this family of new LRU models on computational convergence rates and statistical efficiency. Our experiments are performed on four publicly-available datasets, comparing with Grid-LSTM and Recurrent Highway networks. Our results show that LRU has better empirical computational convergence rates and statistical efficiency values, along with learning more accurate language models.

1 Introduction

Recurrent Neural Networks have been shown to be turing complete (Siegelmann and Sontag 1995) and hence can approximate any given function. Even though they can theoretically represent any form of sequential data, these networks are hard to optimize by gradient methods as gradients start diminishing if backpropagated over large number of time-steps (Bengio, Simard, and Frasconi 1994; Pascanu, Mikolov, and Bengio 2013; Hochreiter et al. 2001; Pascanu, Mikolov, and Bengio 2013). This is overcome by the use of gating mechanisms in Long Short-Term Memory (LSTMs) (Hochreiter and Schmidhuber 1997) and more recently Gated Recurrent Units (GRUs) (Cho et al. 2014). Gates ensures a constant flow of backpropagated error along the temporal dimension, hence making neural sequence models trainable (or optimizable). GRUs, LSTMs and variants of gated recurrent networks seem to capture temporal dependencies and have been successful in tasks such as language modeling (Jozefowicz et al. 2016), machine translation (Sutskever, Vinyals, and Le 2014), handwriting recognition (Graves and Schmidhuber 2009) and generation (Graves 2013), image captioning (Vinyals et al. 2015) and video captioning (Venugopalan et al. 2015).

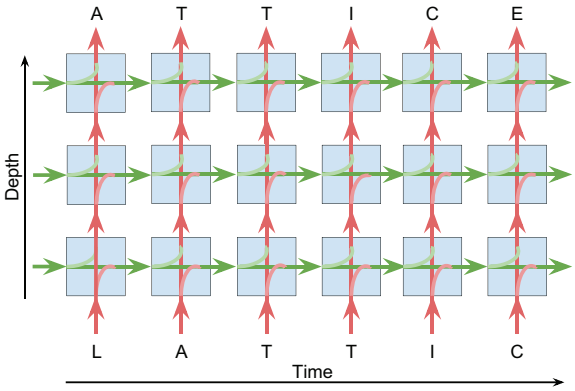

Figure 1: A depiction of how our proposed Lattice Recurrent Unit (LRU) can create a distinct flows of information for both time and depth dimensions. LRU enables better convergence and accuracy, especially with limited resource settings. We exemplify LRU with a character prediction task. The figure shows LRU predicting the word "LATTICE", using the "L" character as the starting point.

The depth of a neural network makes modeling of the data exponentially more efficient (Bianchini and Scarselli 2014), but it is a challenge to optimize the parameters of multi-layer (or deep) models, especially under low resource settings. GridLSTM (Kalchbrenner, Danihelka, and Graves 2016) and Recurrent Highway Networks (RHN) (Zilly et al. 2017) were introduced to improve training of deep LSTM models as it is impractical to train very deep stacked LSTM or GRU models due to the vanishing and exploding gradient problem along depth.

In this paper we propose a family of models, called Lattice Recurrent Units (LRUs), to address the challenge of deep recurrent models. LRU and its variants (Projected State LRU, Reset Gate LRU) are an adaptation of GRUs to a lattice multi-dimensional architecture. The structural differences amongst the variants lies in coupling (or decoupling) of some or all weights. To observe the effects of decoupling weights, we perform experiments on four language modeling datasets and compare perfor-

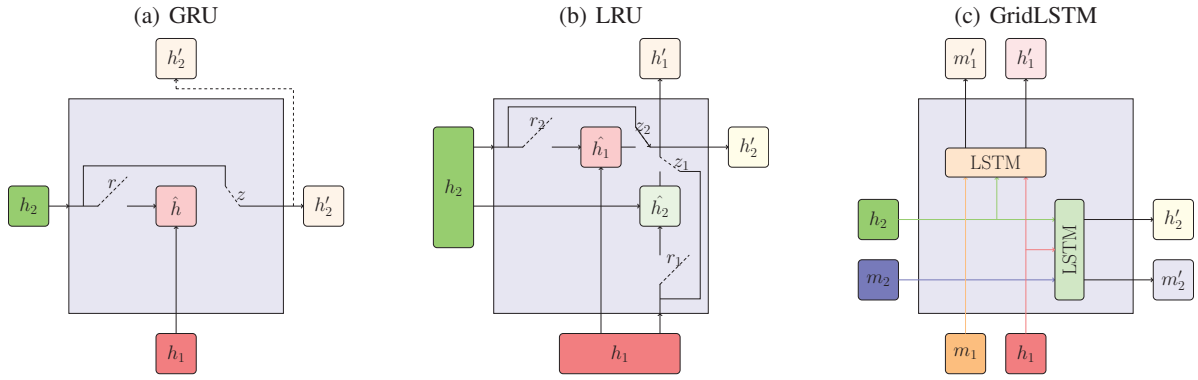

Figure 2: Comparison of our Lattice Recurrent Unit (LRU) model (shown in the middle) compared with the simpler Gated Recurrent Unit (GRU) model and the more recent GridLSTM model. LRU units create a distinct flow of information along time and depth dimensions unlike a GRU unit. We also propose two more variants of our model: Reset Gate LRU (RG-LRU) which couples the update gates (z_1 and z_2), and the Projected State LRU (PS-LRU) which couples the Reset Gates (r_1 and r_2). Decoupling gates in the LRU model allows adaptation to different information propagated along depth and time which gives a boost in accuracy, convergence rates and statistical efficiency.

mancess of GRUs to all the proposed variants of LRUs. As there has been an increasing interest in training speeds of RNNs by parallelizing computations (Bradbury et al. 2017; Vaswani et al. 2017) and reducing complexity models (Joulin et al. 2017), we also compare all the models with *computational convergence rates* and *statistical efficiency* (Bojanowski et al. 2016) as metrics. Also, as a comparison to state-of-the art recurrent units, we perform the same set of experiments on LSTMs, GridLSTMs and RHNs.

2 Background

This section introduces the technical background related to our new Lattice Recurrent Unit (LRU) models. First, we describe the Gated Recurrent Units (GRUs) (Cho et al. 2014) and their multi-layer extensions called Stacked GRU. Our LRU models builds upon this family of recurrent units to enable deeper representations. Second, we describe Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) models and their multi-layer extensions, specifically the Grid LSTM model (Kalchbrenner, Danihelka, and Graves 2016) and Recurrent Highway Networks (Zilly et al. 2017) given their relevance to our research goals. These models will also be used in our experiments as our baselines.

Notation for Sequence Modeling

Consider an ordered input-output sequence $\{(x^{(i)}, y^{(i)})\}_N$ of length N where $x^{(i)} \in \mathbb{R}^d$ is the input, while $y^{(i)} \in \mathbb{R}^d$ is the corresponding output. By an ordered sequence, we mean that $x^{(i+1)}$ comes after $x^{(i)}$ in time. Language modeling is a classic example where the inputs could be characters (or words) and the sequence is a sentence. The order of the characters (or words) is important to retain the meaning of the sentence. Hence, to model the dependencies of a sample $(x^{(n)}, y^{(n)})$, it would be useful to have some information from the previous time-step samples $\{(x^{(n-1)}, y^{(n-1)}), (x^{(n-2)}, y^{(n-2)}) \dots (x^{(1)}, y^{(1)})\}$.

Gated Recurrent Unit

The original Recurrent Neural Networks (RNNs) (Elman 1990; Rumelhart et al. 1988) were designed to model sequential data but it can be hard to optimize its parameters (Pascanu, Mikolov, and Bengio 2013; Bengio, Simard, and Frasconi 1994). Some gated variations were proposed(Yao et al. 2015; Chung et al. 2015; Greff et al. 2016) to counter the effects of vanishing gradients. For the purpose of this paper, we will consider GRU as described in (Cho et al. 2014).

The output of a GRU is also called *hidden state* which is encoded with memory of the sequence till the current time step. GRUs transform the *current input* $x^{(n)}$ and input *hidden state* $h^{(n-1)} \in \mathbb{R}^m$ to $h^{(n)}$ (output *hidden state*) which is used as implicit memory for the next input $x^{(n+1)}$. m is the *hidden state*'s size.

For consistency with notation from later part of this paper, we define *hidden state* flow through horizontal direction (or time dimension) as h_2. Similarly, *hidden state* flow through vertical direction (or depth dimension) is defined as h_1. In case of a GRU model, h_2 represents input hidden state and h'_2 represents output *hidden state*. $x^{(n)}$ is seen as a vertical input, which means that $h_1 = x^{(n)}$. It's important to note that GRU does not have a dedicated output for the vertical dimension. Multi-layer GRUs simply replicate the horizontal output for the vertical dimension: $h'_2 = h'_1$. Formulation of each unit is as follows (biases have been excluded for brevity).

$$z = \sigma\left(\begin{bmatrix}\mathbf{W^z} & \mathbf{U^z}\end{bmatrix}\begin{bmatrix}h_1 \\ h_2\end{bmatrix}\right) \quad (1)$$

$$r = \sigma\left(\begin{bmatrix}\mathbf{W^r} & \mathbf{U^r}\end{bmatrix}\begin{bmatrix}h_1 \\ h_2\end{bmatrix}\right) \quad (2)$$

$$\hat{h} = \tanh\left(\begin{bmatrix}\mathbf{W^h} & \mathbf{U^h}\end{bmatrix}\begin{bmatrix}h_1 \\ r \odot h_2\end{bmatrix}\right) \quad (3)$$

$$h'_2 = z \odot h_2 + (1 - z) \odot \hat{h} \quad (4)$$

where, $\mathbf{W^z}, \mathbf{W^r}, \mathbf{W^h}$ and $\mathbf{U^z}, \mathbf{U^r}, \mathbf{U^h}$ are transform matrices for input and hidden states. σ and $\tanh$ are the usual logistic sigmoid and hyperbolic tangent functions respectively, while $\odot$ is element-wise product.

Gating unit r is often called *Reset Gate* and controls the flow of information coming from input *hidden state* h_2. This transformation decides which features from the previous hidden state, alongside input state h_1, are projected onto a common space to give $\hat{h}$. We refer to this common space $\hat{h}$ as (*Projected State*). Gating unit z is often referred to as *Update Gate* and decides the fraction of h_2 and $\hat{h}$ that is passed to the next time step.

To increase the capacity of GRU networks (Hermans and Schrauwen 2013), recurrent layers can be stacked on top of each other. Since GRU does not have two output states, the same output *hidden state* h_2' is passed to the next vertical layer. In other words, the h_1 of the next layer will be equal to h_2'. This forces GRU to learn transformations that are useful along depth as well as time. While the model can be potentially trained to learn such transformations with a very large training set. When the training set size is limited, a more natural way would be to have two output states, one for the time dimension and a second for the depth dimension.

Long Short-Term Memory

Long Short-Term Memory (LSTM) models were introduced in 1997 to address the issue of vanishing gradient with recurrent neural networks (Hochreiter and Schmidhuber 1997). GRUs and LSTMs have been compared extensively (Jozefowicz, Zaremba, and Sutskever 2015; Greff et al. 2016) on numerous tasks. GRUs performances are generally on par with LSTMs. The major difference in formulations of GRUs and LSTMs is the presence of dedicated *Memory State* in an LSTM cell, unlike GRUs which have *Memory State* encoded in the *Hidden State* itself.

Grid-LSTM

Grid-LSTM (Kalchbrenner, Danihelka, and Graves 2016) can produce N distinct *Memory States* as outputs where $N \geq 1$. To help understand the Gird-LSTM model, let's consider a unit in two dimensional configuration (i.e. N=2; one dimension for time and other for depth). Borrowing notation from (Kalchbrenner, Danihelka, and Graves 2016), we can write:

$$h_1', m_1' = \mathbf{LSTM}(h_1, h_2, m_1, W_1) \quad (5)$$
$$h_2', m_2' = \mathbf{LSTM}(h_1, h_2, m_2, W_2) \quad (6)$$

where (h_1, h_2) and (m_1, m_2) are input *hidden* and *memory* states while(h_1', h_2') and (m_1', m_2') are output *hidden* and *memory* states. W_1 and W_2 are the set of parameters for each dimension, where the subscripts denote the direction of flow of information. **LSTM** function is the typical LSTM recurrent unit described in (Hochreiter and Schmidhuber 1997). A Grid-LSTM unit spits out 2 distinct hidden and *Memory States*, each of which is fed to different dimensions. Both time and depth have their very own forget gates, and this clever trick allows for reduction in the effect of vanishing gradient during optimization.

Recurrent Highway Network

Recurrent Highway Networks (RHN) (Zilly et al. 2017) increase the number of non-linear transformations in one recurrent unit to increase the capacity of the model. This combined with a gated sum of previous and current hidden states (Srivastava, Greff, and Schmidhuber 2015) ensures trainability of the resulting network, even if it were very deep. Even though, RHNs have more capacity they lack the ability of transferring intermediate hidden states along depth to the subsequent time step.

3 Lattice Recurrent Unit

In this section we introduce a family of models, Lattice Recurrent Units, designed to have distinct flow of information for through time and depth dimensions. LRU models can be seen as an expansion of the GRU model. As shown in Equations (1-3), GRU has three main components: *Projected State* $\hat{h}$, *Reset Gate* r and *Update Gate* z. In this paper, we created three members of the LRU family to study an important research question: what components of the GRU model should be decoupled to enable two channels of information?

The first LRU model, named *Projected State* LRU (PS-LRU) will decouple only *projected states* for each dimension. The second model, named *Reset Gate* LRU (RG-LRU), will go one step further by also decoupling *reset gates*. Finally the *Update Gate* LRU (UG-LRU), which we also call LRU, will decouple all three components, including the *update gate*. We decouple one gate at a time and formulate 3 different members of LRU family. The following subsections describe each LRU model in more detail.

Projected State LRU (PS-LRU)

As a first model PS-LRU, we decouple *projected state* $\hat{h}$ to create two new projected states: $\hat{h}_1$ and $\hat{h}_2$. Each of them is used to compute a separate output state: h_1' and h_2'. Formally, PS-LRU is defined using the following update functions,

$$\hat{h}_1 = \tanh\left(\begin{bmatrix}\mathbf{W_1^h} & \mathbf{U_1^h}\end{bmatrix}\begin{bmatrix}h_1 \\ r \odot h_2\end{bmatrix}\right) \quad (7)$$
$$\hat{h}_2 = \tanh\left(\begin{bmatrix}\mathbf{W_2^h} & \mathbf{U_2^h}\end{bmatrix}\begin{bmatrix}r \odot h_1 \\ h_2\end{bmatrix}\right) \quad (8)$$
$$h_1' = z \odot h_1 + (1 - z) \odot \hat{h}_2 \quad (9)$$
$$h_2' = z \odot h_2 + (1 - z) \odot \hat{h}_1 \quad (10)$$

PS-LRU model uses the same *update* and *reset* gates for both output states (see Equation 1 and 2). Note that Equation 4 splits up into Equations 9 and 10 as there are two distinct outputs in this model.

Reset Gate LRU (RG-LRU)

We go one step further and decouple the *Reset Gate* (which was originally defined in Equation 2) to give us two new gates, r_1 and r_2. This model is called RG-LRU and is defined as

follows,

$$r_1 = \sigma\left(\begin{bmatrix}\mathbf{W_1^r} & \mathbf{U_1^r}\end{bmatrix}\begin{bmatrix}h_1\\h_2\end{bmatrix}\right) \quad (11)$$

$$r_2 = \sigma\left(\begin{bmatrix}\mathbf{W_2^r} & \mathbf{U_2^r}\end{bmatrix}\begin{bmatrix}h_1\\h_2\end{bmatrix}\right) \quad (12)$$

$$\hat{h}_1 = \tanh\left(\begin{bmatrix}\mathbf{W_1^h} & \mathbf{U_1^h}\end{bmatrix}\begin{bmatrix}h_1\\r_2 \odot h_2\end{bmatrix}\right) \quad (13)$$

$$\hat{h}_2 = \tanh\left(\begin{bmatrix}\mathbf{W_2^h} & \mathbf{U_2^h}\end{bmatrix}\begin{bmatrix}r_1 \odot h_1\\h_2\end{bmatrix}\right) \quad (14)$$

Note that Equation 3 splits up into Equations 13 and 14 as r has now been decoupled.

Update Gate LRU (UG-LRU or LRU)

In our final model, UG-LRU (which we also refer as LRU for simplicity), we decouple all three main components of GRU, including *Update Gate* (originally defined in Equation 1) which give us gates z_1 and z_2 defined as:

$$z_1 = \sigma\left(\begin{bmatrix}\mathbf{W_1^z} & \mathbf{U_1^z}\end{bmatrix}\begin{bmatrix}h_1\\h_2\end{bmatrix}\right) \quad (15)$$

$$z_2 = \sigma\left(\begin{bmatrix}\mathbf{W_2^z} & \mathbf{U_2^z}\end{bmatrix}\begin{bmatrix}h_1\\h_2\end{bmatrix}\right) \quad (16)$$

$$h_1' = z_1 \odot \hat{h}_2 + (1 - z_1) \odot h_1 \quad (17)$$

$$h_2' = z_2 \odot \hat{h}_1 + (1 - z_2) \odot h_2 \quad (18)$$

Note that Equations 9 and 10 needed a slight modification as z has now been decoupled.

4 Experiments

One of the objectives is to study the effects of coupling gates and control flow of information, hence we split experiments into two parts. The first one is a comparative study on **GRU** and **LRU** family (**PS-LRU**, **RG-LRU**, **LRU**), while the second one compares **LRU** with baseline models including **LSTM**, **GRU**, **RHN** and **Grid-LSTM**[3]. Our LRU models and all baseline models were all trained and tested with the same methodology. In other words, we are not copying result numbers from other papers but instead testing every model to be sure that we have fair comparisons. This focus on fair comparisons (in contrast with other approach to only focus on state-of-the-art performance, even if the experimental environment is different) motivates our choice to not include dropout or variational layers in any of the models.

Models are compared on 3 evaluation metrics,

(1) **Accuracy** - Categorical Cross Entropy (CCE) is used as the loss function and is also used to compare all the models. Lower is better. Model with the best validation loss is used to compute this loss.

(2) **Computational Convergence Rate** - Number of epochs (traversal over complete dataset) taken to converge to the best possible model based on validation scores. Lesser number of epochs to complete optimization is often a desirable trait. It is especially useful in cases where new data is continuously being added and the model needs to be trained multiple times (e.g. active learning).

(3) **Statistical Efficiency** - Generalizing capacity with increasing the size of the dataset (Bojanowski et al. 2016). For e.g. we grab 20% 40% 60% and 80% of a particular dataset and then train models on them independently. We would expect models to perform better with increase in size of the dataset. The model that performs consistently better (Test losses are almost always the best regardless of the size of the dataset), is loosely considered more efficient.

[3]In a Grid-LSTM model it is possible to couple weights across all or some dimensions, but for the sake of comparison we couple the weights across depth.

Task

Character level language modeling is a well-established task for evaluating sequence models (Kalchbrenner, Danihelka, and Graves 2016; Zilly et al. 2017). Character level modeling entails predicting the next character token, given you have k previous character tokens.

This is equivalent to estimating the conditional distribution

$$\mathcal{P}\left(x^{(n)}|\{x^{(n-k)},\ldots,x^{(n-1)}\}\right)$$
$$\forall\{x^{(n-k)},\ldots,x^{(n-1)},x^{(n)}\} \in \mathcal{V}^k$$

where $\mathcal{V}$ is the set of all character tokens. All the sequential models in the upcoming experiments are optimized to estimate this conditional distribution.

Datasets

We use Penn Treebank Dataset (henceforth **PTB**) (Taylor, Marcus, and Santorini 2003) with pre-processing in (Mikolov et al. 2010) and the War and Peace Dataset (henceforth **WP**) as the standard benchmarks for character-level language modeling. PTB contains a set of collected 2499 stories designed to allow the extraction of simple predicate and argument structure. **WP** is basically the book "War and Peace" by Leo Tolstoy. Being a novel, it brings new challenges to the domain of language modeling because of a huge set of punctuation marks in the data. For example, quotation marks (“ ") come in pairs, forcing the model to learn long range dependencies. Both these datasets are relatively small and have around 5 million characters. We selected these datasets to represent scenarios with relatively low resources.

Among bigger datasets, we use **enwik8** and **text8** from the Hutter Prize dataset (Hutter 2012). Both these datasets contain 100 million characters from pages on Wikipedia. While **enwik8** has XML markups, special characters, latin/non-latin characters adding up to 205 unicode symbols, **text8** is a much cleaner dataset with only 27 unicode symbols. The shear size of both these datasets is enough to make the task of language modeling challenging. Following common practice, we chose first 90% for training, next 5% for validation and last 5% for testing for all datasets.

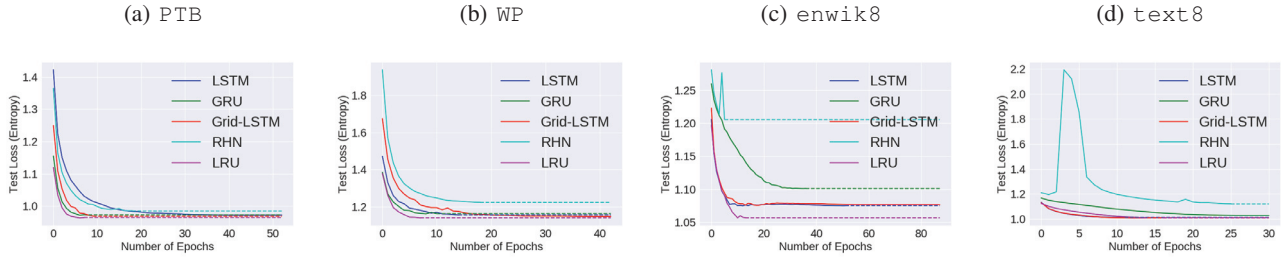

Figure 3: Visual representation of computational convergence rates of LSTM, GRU, RHN, Grid-LSTM and LRU on datasets: PTB, WP, enwik8 and text8. Solid lines denote that actual training curves, while dotted lines are just an extension to visually compare the test score. For example in Figure (a) LRU training ends at around 10 epochs, while LSTM's training continues on till around 50 epochs. LRU converges *faster* to a *lower CCE* loss on unseen data than the other models. All the models have 10M parameters[2]. Y-axis is loss as Categorical Crossentropy Error (CCE) on test sets, while X-axis is number of epochs.

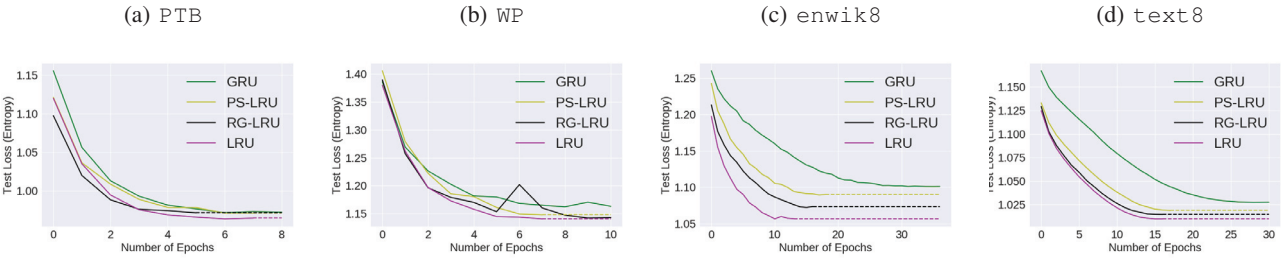

Figure 4: Visual representation of computational convergence rates of GRU, PS-LRU, RG-LRU and LRU on datasets: PTB, WP, enwik8 and text8. LRU and GRU are at two extreme ends of the coupled-weights spectrum. These training curves show that systematically decoupling weights in a GRU (to formulate PS-LRU, RG-LRU and eventually LRU) gives significant boosts to the *convergence rate* and *accuracy* of the model. All the models have 10M parameters.

Training Details

To make the comparison[4] fair, we fixed number of parameters to 10M and 24M based on insights in (Collins, Sohl-Dickstein, and Sussillo 2017). For example, all baseline and LRU models will have number of parameters as close as possible to 10 million in 10M experiments. The same is done for the 24M experiments. All models are either 2 or 4 layers deep, except RHNs which were trained with the transition depth of 5 following the protocol in (Zilly et al. 2017).

Batch size was fixed to 250 and all the models are trained by backpropagating the error up till 50 time steps. We use the optimizer Adam(Kingma and Ba 2015) with an exponentially (factor of 0.9) decaying learning rate of 0.001, $\beta_1 = 0.1$ and $\beta_2 = 0.001$. All weights were initialized using Glorot initialization (Glorot and Bengio 2010).

Evaluation

We optimize our models with Categorical Cross Entropy (CCE) as the loss function and report the same as part of our evaluation. Lower is better. For the purpose of analysis we store loss values on the held out test set and validation set at every epoch. But the checkpoint with the smallest validation loss is considered our best model, so we report the Test Loss obtained on the best model.

[4]Source code available at: https://github.com/chahuja/lru

5 Results and Discussion

GRU and LRU Family

As we mentioned earlier, we tested all recurrent units in the same training environment with parameter budgets of 10 and 24 million parameters. First we compare GRU, PS-LRU, RG-LRU and LRU on all evaluation metrics.

Accuracy: In Table 1, we observe that GRU < PS-LRU < RG-LRU < LRU[5] when CCE loss is the validations metric. LRU is consistently the best performing model, and GRU is the worst. It is interesting to note that number of gates is also in the order GRU < PS-LRU < RG-LRU < LRU. This seems to indicate a correlation between the number of gates and performance.

Figure 4 has Test Losses plot against number of epochs. At the end of the first epoch, it seems that the better performing model is already doing better on the held-out set. From then on the nature of the test curve is similar across GRU and LRU family, which prevents the worse model to catch up.

Computational Convergence Rate: We also look at the time taken by different networks to converge to the best model (checkpoint at which validation loss is the lowest) expressed in *number of epochs*. The choice of library and

[4]Models with 24M parameters were omitted due to space constraints

[5]except for WP where PS-LRU is slightly better than RG-LRU

Table 1: This table compares modeling capacity of LSTM, GRU, Grid-LSTM, RHN, PS-LRU, RG-LRU and LRU on the task of character level language modeling. For each dataset, we report losses on the test-split as Categorical Crossentropy Error (CCE) and Time in number of epochs. Lower is better.

		PTB		WP		enwik8		text8	
Models	**Num. of Params.**	**Time (in epochs)**	**Loss (CCE)**	**Time (in epochs)**	**Loss (CCE)**	**Time (in epochs)**	**Loss (CCE)**	**Time (in epochs)**	**Loss (CCE)**
LSTM	10M	53	0.972	15	1.158	52	1.075	12	1.013
GRU	10M	9	0.972	11	1.163	37	1.101	17	1.047
Grid-LSTM	10M	10	0.970	43	1.148	59	1.077	15	**1.009**
RHN	10M	19	0.986	19	1.225	6	1.206	26	1.121
PS-LRU	10M	8	0.971	8	1.148	19	1.090	17	1.019
RG-LRU	10M	6	0.971	11	1.143	17	1.074	17	1.014
LRU	10M	8	**0.965**	8	**1.141**	14	**1.057**	17	1.010
LSTM	24M	8	0.968	7	1.159	11	1.035	12	0.994
GRU	24M	9	0.980	8	1.164	57	1.121	59	1.089
Grid-LSTM	24M	7	0.971	8	1.171	13	**1.034**	13	**0.986**
RHN	24M	18	0.979	21	1.199	34	1.092	18	1.108
PS-LRU	24M	5	0.969	5	1.152	25	1.070	32	1.058
RG-LRU	24M	4	0.969	5	1.155	23	1.065	19	1.018
LRU	24M	5	**0.967**	6	**1.150**	17	1.041	16	1.010

recurrent unit implementations[6] make a huge difference to the speed (per second) of applying gradient descent updates, hence we choose *number of epochs* as the unit for comparison. For a compact overview we consider all[7] the experiments we conducted and make a box plot (Figure 6). It is interesting to note that all variants of LRU, at an average, take around 12 epochs to converge.

Statistical Efficiency: We grab 20%, 40%, 60% and 80% of PTB and text8 to construct smaller mini-datasets. After running experiments with GRU, and LRU family on the new datasets we calculate the losses on the held-out test set and choose the best models for each mini-dataset. All the models have a parameter budget of 10M. After LRU performed the best amongst its family and GRU, it was not a surprise when it consistently had the best loss for all mini-datasets (Figure 5). The graphs are an indication of the best generalizing capability and empirical statistical efficiency of LRU (amongst its family and GRU).

LRU Family and Baseline Models

Accuracy: Next, we compare LSTM, GRU, RHN and Grid-LSTM to LRU in Table 1. We see that LRUs perform consistently better than all other recurrent units especially on smaller datasets WP and PTB, which suggests that LRU has better modeling power in scenarios with small training datasets. Figure 3 has Test Losses plot against number of epochs. An interesting behaviour is seen RHNs curves where the test loss rises at first and then starts decreasing, while other models converge almost monotonically to their minimum value. LRU almost always is the best model at the end of one epoch, and it continues to maintain the lead with time.

Computational Convergence Rate: It is evident from Figure 6, that Grid-LSTMs, RHNs and GRUs, on an average, take around 22 epochs to converge which is almost double of that of the LRU family. LSTMs, at an average, require even more epoch of around 30 through the complete dataset to converge. Also, LSTMs have a high standard deviation of around 28 epochs. So, LSTMs could potentially take a large number of epochs to converge, while LRUs which have a low standard deviation of around 6 epochs have a more stable convergence rate over different datasets.

Statistical Efficiency: As expected, test losses for each model decrease monotonically with the increase in amount of data with an exception of the transition from 80% to 100% in text8. One possible reason could be that last 20% of text8 is much harder to model than the rest. LRUs perform just as good as Grid-LSTMs and LSTMs with increase in the amount of data available for training, if not better. For PTB, the difference between the losses of different models increase with the decrease in data (Figure 5), with the best being LRU and the worst being LSTM. All the models perform equally well on text8 with no significant difference across models on its mini-datasets. With the evidence in hand, LRUs seem to have the better empirical statistical efficiency, especially in cases with lesser data.

Vanishing Gradients

To demonstrate the effectiveness of two distinct streams of information for depth and time, we train a 10 layer deep language model on PTB with LRU and GRU as the recurrent units. As is evident in Figure 7, the average gradient norms

[6] We use PyTorch implementations of LSTM and GRU which are highly optimized with a C backend in contrast to user-implemented recurrent units

[7] All experiments include all training instances run on all the models on all datasets discussed in this paper

(a) PTB (b) text8

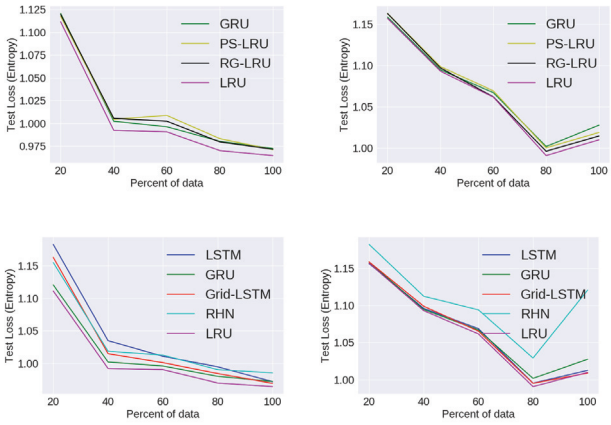

Figure 5: Visual representation of Statistical Efficiency of LSTM, GRU, RHN, Grid-LSTM, PS-LRU, RG-LRU and LRU on PTB and text8. Even though decreasing the amount of data negatively impacts the accuracy of the language model, LRU performs better when compared to other models. A LRU network trained on 40% of the data is better than other models at 60% of the data which indicates superiority in low-resource scenarios. Y-axis is test loss for the best set of parameters on validation sets, while X-axis is percent of data used for training. All models have 10M parameters.

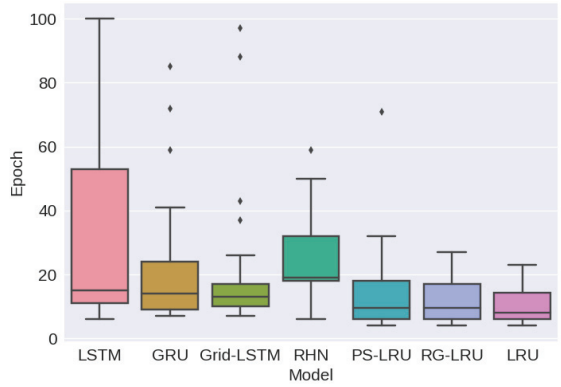

Figure 6: Box plot of *number of epochs* required for convergence, across all models and conducted experiments. At an average the family of LRU models converge faster with a low standard deviation in convergence epochs across experiments. LSTM, GRU, Grid-LSTM and RHN networks have a much higher mean and standard deviation in convergence epochs along with many outliers than can take as much as 5 times more than a LRU network to train. Hence, LRU models demonstrate a faster and more stable convergence rate.

are concentrated close the last layers in a GRU model, while they are spread across more evenly in a LRU model. This indicates that gradients are backpropagated deep enough for

(a) GRU (b) LRU

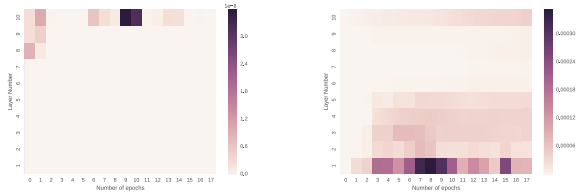

Figure 7: Gradient Norms across layers and number of epochs as a heatmap. It represents the distribution of gradients across layers and number of epochs for each model. Darker values are bigger. Note that the color bar is not the same for both the figures as the mean values of the gradients are far apart from each other. GRUs have extremely low gradients while LRU has high gradients across most layers.

gradient based optimization to work.

6 Conclusion

We introduced a new family of models, called Lattice Recurrent Units (LRU), to address the challenge of learning deep multi-layer recurrent models with limited resources. Our experiments are performed on four publicly-available datasets, comparing LRU with Grid-LSTM, Recurrent Highway networks, LSTMs and GRUs. Results indicated that LRU has the best accuracy, convergence rate, and statistical efficiency amongst all the baseline models when training language models, especially, when dealing with small datasets. We also analyzed the effects of decoupling three different components of our LRU model: *Reset Gate*, *Update Gate* and *Projected State*. Amongst GRU, PS-LRU, RG-LRU and LRU, LRU was the best in all the 3 metrics of evaluation mentioned above. In fact, a trend was observed: Decoupling of gates leads to better performance. Hence, LRU models achieved the goal of learning multi-layer networks with limited resources by creating distinct (but coupled) flow of information along time and depth.

7 Acknowledgements

This project was partially supported by Oculus Research Grant. The authors would like to thank Volkan Cirik, Rama Kumar Pasumarthi and Bhuwan Dhingra for their valuable inputs.

References

Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5(2):157–166.

Bianchini, M., and Scarselli, F. 2014. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems* 25(8):1553–1565.

Bojanowski, P.; Grave, E.; Joulin, A.; and Mikolov, T. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Bradbury, J.; Merity, S.; Xiong, C.; and Socher, R. 2017. Quasi-recurrent neural networks. *ICLR*.

Cho, K.; Van Merriënboer, B.; Bahdanau, D.; and Bengio, Y. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation at EMNLP* 103–111.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2015. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, 2067–2075.

Collins, J.; Sohl-Dickstein, J.; and Sussillo, D. 2017. Capacity and trainability in recurrent neural networks. *ICLR*.

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

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256.

Graves, A., and Schmidhuber, J. 2009. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, 545–552.

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

Greff, K.; Srivastava, R. K.; Koutník, J.; Steunebrink, B. R.; and Schmidhuber, J. 2016. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*.

Hermans, M., and Schrauwen, B. 2013. Training and analysing deep recurrent neural networks. In *Advances in Neural Information Processing Systems*, 190–198.

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

Hochreiter, S.; Bengio, Y.; Frasconi, P.; Schmidhuber, J.; et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A field guide to dynamical recurrent neural networks. IEEE Press*.

Hutter, M. 2012. The human knowledge compression contest. *http://prize.hutter1.net/*.

Joulin, A.; Grave, E.; Bojanowski, P.; and Mikolov, T. 2017. Bag of tricks for efficient text classification. *EACL* 427–431.

Jozefowicz, R.; Vinyals, O.; Schuster, M.; Shazeer, N.; and Wu, Y. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Jozefowicz, R.; Zaremba, W.; and Sutskever, I. 2015. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2342–2350.

Kalchbrenner, N.; Danihelka, I.; and Graves, A. 2016. Grid long short-term memory. *ICLR*.

Kingma, D., and Ba, J. 2015. Adam: A method for stochastic optimization. *ICLR*.

Mikolov, T.; Karafiát, M.; Burget, L.; Cernockỳ, J.; and Khudanpur, S. 2010. Recurrent neural network based language model. In *Interspeech*, volume 2, 3.

Pascanu, R.; Mikolov, T.; and Bengio, Y. 2013. On the difficulty of training recurrent neural networks. *ICML* 28:1310–1318.

Rumelhart, D. E.; Hinton, G. E.; Williams, R. J.; et al. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5(3):1.

Siegelmann, H. T., and Sontag, E. D. 1995. On the computational power of neural nets. *Journal of computer and system sciences* 50(1):132–150.

Srivastava, R. K.; Greff, K.; and Schmidhuber, J. 2015. Highway networks. *arXiv preprint arXiv:1505.00387*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Taylor, A.; Marcus, M.; and Santorini, B. 2003. The penn treebank: an overview. In *Treebanks*. Springer. 5–22.

Vaswani, A.; Shazeer, N.; Parmar, N.; Jones, L.; Uszkoreit, J.; Gomez, A. N.; and Kaiser, L. u. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*. 5994–6004.

Venugopalan, S.; Rohrbach, M.; Donahue, J.; Mooney, R.; Darrell, T.; and Saenko, K. 2015. Sequence to sequence-video to text. In *Proceedings of the IEEE International Conference on Computer Vision*, 4534–4542.

Vinyals, O.; Toshev, A.; Bengio, S.; and Erhan, D. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3156–3164.

Yao, K.; Cohn, T.; Vylomova, K.; Duh, K.; and Dyer, C. 2015. Depth-gated lstm. *Jelinek Summer Workshop on Speech and Language Technology*.

Zilly, J. G.; Srivastava, R. K.; Koutník, J.; and Schmidhuber, J. 2017. Recurrent highway networks. In *ICML*.