Initial Explorations on Regularizing the SCRN Model

Capstone Project By Olzhas Kabdolov
Supervisor: Zhenisbek Assylbekov, Second Reader: Rustem Takhanov

Abstract

Recurrent neural networks are very powerful sequence models which are used for language modeling as well. Under correct regularization such as naive dropout these models are able to achieve substantial improvement in their performance. We regularize the Structurally Constrained Recurrent Network (SCRN) model and show that despite its simplicity it can achieve the performance comparable to the ubiquitous LSTM model in language modeling task while being smaller in size and up to 2x faster to train. Further analysis shows that regularizing *both* context and hidden states of the SCRN is crucial.

Keywords: recurrent neural networks, words embeddings, regularization, naive dropout.

1 Introduction

Recurrent neural networks (RNN) have demonstrated tremendous success in sequence modeling tasks in general and in machine translation, speech recognition, time series in particular.

The most basic RNN (Elman, 1990) suffers from the problem of vanishing and exploiding gradients, (Bengio et al., 1994) and is hard to train efficiently. One of the most widespread and efficient alternatives to the basic RNN is the Long-Short Term Memory (LSTM) model (Hochreiter and Schmidhuber, 1997), which effectively addresses the problem of vanishing gradients. However, LSTM is a fairly complex model with excessive number of parameters and its inner functionality is not obvious. This complexity has motivated some of the researchers to find more apparent and less complex alternatives. One of such alternative models is a Structurally Constrained Recurrent Network (SCRN) proposed by Mikolov et al. (2015). They encouraged some of the hidden units to change their state slowly by making part of the recurrent weight matrix close to identity, thus forming a kind of longer term memory and showed that their SCRN model can outperform the simple RNN and achieve the performance comparable with the LSTM under no regularization and small parameter budget. It is natural to try to regularize the SCRN model under larger budgets: *Will it approach the performance of LSTM under the same regularization technique and the same parameter budget?* Our experiments show that the answer is positive for the case of naïve dropout (Zaremba et al., 2014) and 5M/20M budgets.

Recently Ororbia II et al. (2017) introduced Delta-RNN architecture for which SCRN serves as a predecessor. They showed that, when regularized, Delta-RNN performs comparably to LSTM and GRU (Chung et al., 2014). However, they did not compare to regularized SCRN, and our paper fills this gap.

Moreover, we have done experiments on modifications of the original model to see how it will perform and analyzed the possible reasons.

2 Related work

There are many variations of recurrent neural networks which has been proposed in the recent time and used the field of language modeling.

The SCRN architecture itself can be viewed as the combination of old and less stable variations of RNNs (Mozer, 1993), (Jordan, 1990) which are more simple and do not introduce the constraint which is close to identity.

One of the most similar model that was inspired by SCRNs is Delta RNN (Ororbia II et al., 2017). The key difference between them is that Delta RNN does not make the partition of slow and fast connection, but combines them. One state of Delta-RNN inputs data at the current time-step and function which recaps the previous states. This state is called the fast interpolation gate. Another state is slow and data-independent. This model achieves results comparable to LSTM under the same paremeter budget. However, it need more operations than SCRNs.

The other model which was introduced just recently is called Simple Recurrent Unit (SRU) (Lei and Zhang, 2017). The idea of this model is to use a gate which is a simple one-layer feed-forward network with sigmoid activation function. This gate is fast to be computed because it can be parallelized. Its output is used in a point-wise multiplication in a state which combines the previous state and the output feed-forward layer at the current time step. However, one important thing which needs to be mentioned is that this model also exploits the idea of skip connections (Zilly et al., 2016). The model has the reset gate which uses the highway connection by applying this gate directly to the input at current time step. The use of skip connections is probably the reason why the model achieve high performance under the same parameter budget.

A final model which is related to the scope of our project is called Recurrent Additive Networks (RAN) (Lee et al., 2017). The model architecture is similar to SRU but does not exploit the skip connections. The authors of the work declared that the model has small number of parameters and achieves high performance. However, the paper by (Lee et al., 2017) does not specify the whole number of parameters but instead skips those of softmax and word embeddings. That's why it hard to make clear comparison between RANs and SCRNs.

3 Baseline SCRN Model

Let $\mathcal{W}$ be a finite vocabulary of words. We assume that words have already been converted into indices. Based on one-hot word embeddings $\mathbf{x_{1:k}} = \mathbf{x_1}, \ldots, \mathbf{x_k}$ for a sequence of words $w_{1:k}$, the baseline SCRN model (Mikolov et al., 2015) produces two sequences of states, $\mathbf{s_{1:k}}$ and $\mathbf{h_{1:k}}$, according to[1]

$$\mathbf{s_t} = (1-\alpha)\mathbf{x_t}\mathbf{B} + \alpha\mathbf{s_{t-1}}, \tag{1}$$

$$\mathbf{h_t} = \sigma(\mathbf{x_t}\mathbf{P} + \mathbf{s_t}\mathbf{A} + \mathbf{h_{t-1}}\mathbf{R}), \tag{2}$$

[1]Vectors are assumed to be row vectors, which are right multiplied by matrices ($\mathbf{xW} + \mathbf{b}$). This choice is somewhat non-standard but it maps better to the way networks are implemented in code using matrix libraries such as `TensorFlow`.

where $\mathbf{B} \in \mathbb{R}^{|\mathcal{W}| \times d_s}$, $\mathbf{P} \in \mathbb{R}^{|\mathcal{W}| \times d_h}$, $\mathbf{A} \in \mathbb{R}^{d_s \times d_h}$, $\mathbf{R} \in \mathbb{R}^{d_h \times d_h}$, d_s and d_h are dimensions of $\mathbf{s_t}$ and $\mathbf{h_t}$, $\sigma(\cdot)$ is the logistic sigmoid function. Mikolov et al. (2015) refer to $\mathbf{s_t}$ as a slowly changing *context state*, and to $\mathbf{h_t}$ as a quickly changing *hidden state*. The last couple of states $(\mathbf{s_k}, \mathbf{h_k})$ is assumed to contain information on the whole sequence $w_{1:k}$ and is further used for predicting the next word w_{k+1} of a sequence according to the probability distribution

$$\Pr(w_{k+1}|w_{1:k}) = \operatorname{softmax}(\mathbf{s_k}\mathbf{U} + \mathbf{h_k}\mathbf{V}), \tag{3}$$

where $\mathbf{U} \in \mathbb{R}^{d_s \times |\mathcal{W}|}$ and $\mathbf{V} \in \mathbb{R}^{d_h \times |\mathcal{W}|}$ are output embedding matrices. For the sake of simplicity we omit bias terms in (2) and (3).

Training the model involves minimizing the negative log-likelihood over the corpus $w_{1:K}$:

$$-\textstyle\sum_{k=1}^{K} \log \Pr(w_k|w_{1:k-1}) \longrightarrow \min_{\Theta}, \tag{4}$$

which is usually done by truncated backpropagation through time (Werbos, 1990). Here Θ denotes the set of all model parameters.

Notice that SCRN is a slight modification of the vanilla RNN model, and its simplicity is in stark contrast with the complexity of the widespread LSTM model.

Dense embeddings: The original model spends

$$2 \cdot |\mathcal{W}| \cdot (d_s + d_h) \tag{5}$$

parameters to embed words into dense vectors at input and at output. We believe that it is more beneficial to first embed words w_t into dense vectors $\mathbf{w_t} = \mathbf{x_t}\mathbf{E} \in \mathbb{R}^{d_h}$ using only one embedding matrix $\mathbf{E} \in \mathbb{R}^{\mathcal{W} \times d_h}$ and then use $\mathbf{w_t}$ instead of $\mathbf{x_t}$ in (1) and (2) with the appropriate change of shapes for matrices: $\mathbf{B} \in \mathbb{R}^{d_h \times d_s}$ and $\mathbf{P} \in \mathbb{R}^{d_h \times d_h}$. In this case, the model spends $|\mathcal{W}| \cdot (2d_h + d_s)$ parameters on input/output embeddings, which is $d_s \cdot |\mathcal{W}|$ parameters less than (5). E.g., on the Penn Tree Bank (PTB) dataset (Marcus et al., 1993), where $|\mathcal{W}| = 10,000$, the reduction is 1M parameters for the SCRN model with $d_s = 100$.

4 Regularizing and Stacking SCRN

Due to high complexity of deep neural networks the regularization techniques are crucial for good generalization performance. Zaremba et al. (2014) proposed one of the ways how dropout (Srivastava et al., 2014) can be used to regularize recurrent neural networks. They apply dropout only to non-recurrent connections, while keeping the recurrent connections without change. This method, usually referred to as *naïve dropout*, improves the baseline results of the LSTM model without any other modifications. We investigate how application of the same dropout technique benefits the SCRN model, and, to the best of our knowledge, this was not done previously.

It is well known that stacking at least two layers in recurrent neural networks is beneficial, but between two and three layers the results are mixed (Karpathy et al., 2016; Laurent and von Brecht, 2017). To make our results comparable to the previous works on using the naïve dropout in RNN language modeling (Zaremba et al., 2014; Kim et al., 2016), we also use a two-layered architecture: the output of the first layer (1, 2) is concatenated $[\mathbf{s_t}; \mathbf{h_t}]$ and is fed as input into the second layer.

In what follows the superscript index in round brackets denotes the layer index, $\boldsymbol{\xi}(p) = [\xi_1, \ldots, \xi_d]$ is a random vector (dropout mask) with $\xi_i \sim \mathrm{Bernoulli}(1-p)$, d is the dimensionality of the corresponding layer, p is a dropout rate, and $\odot$ is the element-wise (Hadamard) product. Regularized and stacked SCRN model is fully specified by the following equations:

- Input embedding layer + dropout:

$$\mathbf{w_t} = \mathbf{x_t}\mathbf{E} \odot \boldsymbol{\xi}^{(0)}_{\mathbf{t}}(p_i).$$

- First SCRN layer + dropout:

$$\begin{aligned}\mathbf{s}^{(1)}_{\mathbf{t}} &= (1-\alpha)\mathbf{w_t}\mathbf{B}^{(1)} + \alpha\mathbf{s}^{(1)}_{\mathbf{t-1}},\\ \mathbf{h}^{(1)}_{\mathbf{t}} &= \sigma\left(\mathbf{w_t}\mathbf{P}^{(1)} + \mathbf{s}^{(1)}_{\mathbf{t}}\mathbf{A}^{(1)} + \mathbf{h}^{(1)}_{\mathbf{t-1}}\mathbf{R}^{(1)}\right),\\ \mathbf{y}^{(1)}_{\mathbf{t}} &= [\mathbf{s}^{(1)}_{\mathbf{t}}; \mathbf{h}^{(1)}_{\mathbf{t}}] \odot \boldsymbol{\xi}^{(1)}_{\mathbf{t}}(p_o).\end{aligned} \tag{6}$$

- Second SCRN layer + dropout:

$$\begin{aligned}\mathbf{s}^{(2)}_{\mathbf{t}} &= (1-\alpha)\mathbf{y}^{(1)}_{\mathbf{t}}\mathbf{B}^{(2)} + \alpha\mathbf{s}^{(2)}_{\mathbf{t-1}},\\ \mathbf{h}^{(2)}_{\mathbf{t}} &= \sigma\left(\mathbf{y}^{(1)}_{\mathbf{t}}\mathbf{P}^{(2)} + \mathbf{s}^{(2)}_{\mathbf{t}}\mathbf{A}^{(2)} + \mathbf{h}^{(2)}_{\mathbf{t-1}}\mathbf{R}^{(2)}\right),\\ \mathbf{y}^{(2)}_{\mathbf{t}} &= [\mathbf{s}^{(2)}_{\mathbf{t}}; \mathbf{h}^{(2)}_{\mathbf{t}}] \odot \boldsymbol{\xi}^{(2)}_{\mathbf{t}}(p_o).\end{aligned} \tag{7}$$

- Softmax prediction:

$$\Pr(w_{t+1}|w_{1:t}) = \mathrm{softmax}\left(\mathbf{y}^{(2)}_{\mathbf{t}} \begin{bmatrix}\mathbf{U}\\ \mathbf{V}\end{bmatrix}\right).$$

In the equations above, p_i and p_o are input and output dropout rates, the matrices $\mathbf{U}$ and $\mathbf{V}$ are concatenated along the first dimension to produce a $(d_s + d_h) \times |\mathcal{W}|$ matrix.

5 Experimental setup

We use perplexity (PPL) to evaluate the performance of the language models. Perplexity of a model over a sequence $[w_1, \ldots, w_T]$ is given by

$$\mathrm{PPL} = \exp\left(-\frac{1}{T}\sum_{k=1}^{T}\log\Pr(w_k|w_{1:k-1})\right).$$

Data sets: All configurations are trained and evaluated on the PTB (Marcus et al., 1993) and the WikiText-2 (Merity et al., 2017) data sets. For the PTB we utilize the standard training (0-20), validation (21-22), and test (23-24) splits along with pre-processing per Mikolov et al. (2010). WikiText-2 is an alternative to PTB, which is approximately two times as large in size and three times as large in vocabulary.

Baseline Model: To reproduce the results of the baseline (single-layer and non-regularized) SCRN model we use the original `Torch` implementation[2] released by Mikolov et al. (2015). We have spent fair amount of time and effort to make their script run, as several of its dependencies have not been updated for few years, and are not compatible with the up-to-date versions of the others. To simplify the path for other researchers we release a script[3], which installs necessary versions of the dependencies. We use exactly the same set of hyperparameters which were reported in the original paper (Table 1). We also implement the baseline SCRN model ourselves using `TensorFlow` (Abadi et al., 2016) which, unlike `Torch`, uses static computational graphs, and thus has different style of truncated backpropagation through time[4] (BPTT). The choice of hyperparameters (Table 1) is motivated by the previous work on word-level

Hyper-parameter	Mikolov et al. (2015)	Our implementation
α in (1)	0.95	0.95
batch size	32	20
initial lr	0.05	0.8
lr decay	1/1.5	0.75
BPTT steps	50	70
BPTT freq.	5	70
gradients	renormalized	clipped at 5
weights init.	$[-0.05, 0.05]$	$[-0.05, 0.05]$

Table 1: Hyperparameters of the baseline SCRN model. Abbreviations: lr — learning rate, BPTT — backpropagation through time.

[2]https://github.com/facebookarchive/SCRNNs

[3]http://masked

[4]https://r2rt.com/styles-of-truncated-backpropagation.html

language modeling (Zaremba et al., 2014). Another difference between the two implementations is that Mikolov et al. (2015) decay the learning rate after each training epoch when the validation error does not decrease, whereas we keep it constant during the first 6 epochs and then decay it every epoch regardless of the validation set perplexity.

Regularized and Stacked Models: Regularization and stacking is performed according to Section 4. We optimize hyperparameters under 5M and 20M parameter budgets on the PTB data set. Some of the hyperparameters are tuned using random search according to the marginal distributions:

- $p_i \sim U[0.01, 0.5]$,
- $p_o \sim U[0.01, 0.5]$,
- $d_s \in \{40, 80\}$,
- initial lr $\sim U[0.5, 0.99]$,
- lr decay $\sim U[0.5, 0.89]$,
- momentum $\sim U[0.4, 0.7]$,
- weights initil'n margin $\sim U[0.05, 0.2]$.

where $U[a, b]$ means continuous uniform distribution over the interval $[a, b]$. Other hyperparameters are tuned manually through trial-and-error. When performing random search we first choose ranges mentioned above. After 100 iterations the initial ranges are shrinked to the neighborhoods of the values that give best performances, and the random search is performed again. We repeat this procedure until hyperparameters converge to their (sub)optimal values. To prevent exploding gradients we clip the norm of the gradients (normalized by minibatch size) at 5. For training (4) we use stochastic gradient descent with momentum which is decayed after 10 and 20 epochs for small and medium models respectively.

Hyperparameter	5M budg.	20M budg.
p_i	0.04	0.4
p_o	0.19	0.45
d_s	40	120
d_h	210	750
α	0.95	0.95
batch size	16	16
initial lr	0.51	0.57
lr decay	0.77	0.63
BPTT steps	33	33
BPTT freq.	33	33
gradients norm	clipped at 5	clipped at 5
weights init.	$[-0.3, 0.3]$	$[-0.3, 0.3]$

Table 2: Tuned hyperparameters of the regularized two-layer SCRN models under two budgets.

6 Results

To assure that our implementation of the baseline SCRN is adequate, we evaluate it against the original SCRN code by Mikolov et al. (2015) (Table 3). As one can see, the original SCRN code does *not* fully reproduce the results reported in the corresponding paper. Their hyperparameters (Table 1) work well for the case when $(d_s, d_h) \in \{(40, 10), (90, 10)\}$, but are not optimal for the other two configurations. Our implementation together with our set of hyperparameters (Table 1) brings the validation and test perplexities of the model with $(d_s, d_h) = (300, 40)$ close to that which is reported in the paper, but on the other hand *fails* to perform well for the other configurations. Interestingly, both implementations show practically the same performance for the case when $(d_s, d_h) = (100, 40)$. It seems that hyperparameters should be tuned separately for each case of (d_s, d_h), but we followed the methodology of Mikolov et al. (2015), where the same set of hyperparameters is used across all configurations of (d_s, d_h).

Tuning hyperparameters for the regularized and stacked SCRN model under 5M and 20M parameter budgets gives the values in Table 2. The results of evaluating these two models against regularized and stacked LSTMs on PTB and WikiText-2 are provided in Table 4. Naïve dropout and stacking *do* benefit the simple and intuitive SCRN model and bring its performance close to that of the sophisticated LSTM model under the same parameter budget. This shows that seemingly less complex models are still competitive under appropriate regularization and optimization. It is important to mention that no architectural modifications were applied to the original SCRN model except stacking.

We finally notice that increasing context state size does not always increase the model's performance. The random search showed that the context state size in good SCRN configurations should be smaller than the hidden state size.

6.1 Ablation analysis

We consider removal of some parts or forms of regularization from our best-performing SCRN

Hidden size	Context size	Mikolov et al. (2015) Valid PPL	Mikolov et al. (2015) Test PPL	Our implementation Valid PPL	Our implementation Test PPL
40	10	133.6 (133)	127.5 (127)	145.5	139.9
90	10	125.4 (124)	120.3 (119)	129.9	125.3
100	40	127.6 (120)	122.9 (115)	127.6	122.1
300	40	130.1 (120)	124.4 (115)	120.4	116.2

Table 3: Reproducing the baseline model. For the original implementation (columns 3 and 4), values outside brackets were obtained when running the script from `https://github.com/facebookarchive/SCRNNs`, and values in brackets were reported in the paper of Mikolov et al. (2015).

Word-level model	PTB 5M budg.	PTB 20M budg.	WikiText-2 Small	WikiText-2 Medium
LSTM (Jozefowicz et al., 2015)	—	**79.8**	—	—
LSTM (Kim et al., 2016)	97.6	85.4	116.8[†]	—
LSTM (Zaremba et al., 2014)	—	82.7	—	96.2[‡]
Delta-RNN (Ororbia II et al., 2017)	—	84.0	—	—
SCRN (this paper)	**97.4**	84.7	122.1	99.2

Table 4: Evaluation of the SCRN against LSTMs under naïve dropout regularization. [†]We reproduced the LSTM-Word-Small model from Kim et al. (2016) on PTB and then evaluated it on WikiText-2. [‡]We ran the open-source implementation from `https://github.com/tensorflow/models/tree/master/tutorials/rnn/ptb` at medium config on WikiText-2 data.

model to see whether such removal degrades the performance. It also tells us which parts of the model are most important. The model and dropout variations are listed below.

Removing dropout: To understand how much improvement is gained by the use of regularization we completely remove dropout:

$$p_i = p_o = 0.$$

Removing dropout of the context state: According to (1), context state $\mathbf{h_t}$ changes linearly and thus should not suffer from over-fitting. Thus it seems reasonable to try to not regularize it and apple dropout only to the hidden states, i.e. replacing equations (6) and (7) by

$$\mathbf{y_t}^{(l)} = [\mathbf{s_t}^{(l)}; \mathbf{h_t}^{(l)} \odot \boldsymbol{\xi}_\mathbf{t}^{(l)}(p_o)], \qquad l = 1, 2.$$

Removing the context state from the output: As in the case of the SCRN, the inner state of the LSTM model also consists of two vectors $\mathbf{c_t}$ and $\mathbf{h_t}$, and usually the state $\mathbf{c_t}$ is not used at output. We do the same for the context state $\mathbf{s_t}$ in our model, i.e. the equations (6) and (7) are replaced by

$$\mathbf{y_t}^{(l)} = \mathbf{h_t}^{(l)} \odot \boldsymbol{\xi}_\mathbf{t}^{(l)}(p_o), \qquad l = 1, 2.$$

The meaningfulness of removing the context state from the output is that, in our opinion, it plays the role of a long-term memory and thus should not be crucial for predicting the next word of a sequence. Moreover, such removal implies not using $\mathbf{s_t}$ in the softmax layer, and therefore it reduces the model size by at least $d_s \cdot |\mathcal{W}|$ parameters, which can be significant (see Section 3). Due to architectural changes we use the same approach to hyperparameter search as before (Section 5) to achieve the best result for this ablation.

The results of the ablation analysis are provided in Table 5. As we can see, without dropout our best model fails to generalize well on validation and test sets. Regularizing only the hidden state (and keeping the context state untouched) is less harmful but still degrades the performance of the best model. Finally, not using the context state in the output only slightly worsens the performance but at the same time leads to a significant reduction in model size. We conclude that even though the context layer is important in preventing vanishing gradients, it contains much less information for short-term prediction (unlike the hidden state) and can be omitted from the output.

Model	Small (5M) Valid	Small (5M) Test	Medium (20M) Valid	Medium (20M) Test
SCRN + dropout + two layers	102.9	97.4	89.2	84.5
− dropout	153.2	145.9	175	180
− dropout of context	117.6	111.9	106.3	100.6
− context in output	104.3	99.01	92.8	88.6

Table 5: Model ablations for our best small and medium SCRN models on PTB set.

6.2 Training speeds

Training speeds are provided in the Table 6. Models were implemented in `TensorFlow`, and were run on a single Titan X (Maxwell). As is expected,

	PTB		WikiText-2	
Model	Small	Medium	Small	Medium
LSTM	5.4	3.3	4.1	2.5
SCRN	9.7	5.5	7.8	3.7

Table 6: Training speeds, in thousands of tokens per second.

due to its simplicity, the SCRN model is trained faster than the LSTM.

6.3 Learned Word Representations

We pick several words from the English PTB vocabulary and consider their nearest neighbors under cosine similarity as produced by the medium-sized LSTM and SCRN models at input (Table 7), i.e. in the matrix $\mathbf{E}$. It is not clear whether one model gives better embeddings than the other. Further analysis with specific evaluation metrics is needed and we defer it to our future work.

7 Conclusion

While being conceptually much simpler, SCRN architecture can achieve the performance of the widely used LSTM model on language modeling task under naïve dropout regularization. However, it is interesting to see if the same holds true under the more recently proposed regularization and optimization techniques for recurrent neural network language models, such as variational dropout (Gal and Ghahramani, 2016), weight tying (Press and Wolf, 2017; Inan et al., 2017), averaged stochastic gradient descent and DropConnect (Merity et al., 2018). We defer such study to our future work, which will also include larger-scale language modeling experiments for different languages as well as subword-level modeling.

In addition, we intend to explore how useful the SCRN might be in other tasks that the architectures such as the LSTM currently hold state-of-the-art performance in.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

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.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

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

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Proc. of NIPS*.

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

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *Proc. of ICLR*.

Michael I Jordan. 1990. Artificial neural networks. *Attractor dynamics and parallelism in a connectionist sequential machine*, pages 112–127.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *Proc. of ICML*, pages 2342–2350.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2016. Visualizing and understanding recurrent networks.

	Sample Words				
	woman	me	say	good	study
LSTM (Zaremba et al., 2014)	*man*	*nobody*	*believe*	*bad*	*survey*
	driver	*somebody*	*says*	*bearish*	*report*
	artist	*guys*	*think*	*positive*	*filing*
	senator	*deciding*	*said*	*interesting*	*measure*
SCRN (this paper)	*men*	*him*	*says*	*bad*	*survey*
	someone	*nobody*	*said*	*positive*	*analyst*
	women	*things*	*saying*	*negative*	*authors*
	everybody	*i*	*thinks*	*every*	*director*

Table 7: Sampled words and their nearest vector embeddings based on cosine similarity for two models.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Proc. of AAAI*, pages 2741–2749.

Thomas Laurent and James von Brecht. 2017. A recurrent neural network without chaos. In *Proc. of ICLR*.

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

Tao Lei and Yu Zhang. 2017. Training rnns as fast as cnns. *arXiv preprint arXiv:1709.02755*.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing lstm language models. In *Proc. of ICLR*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proc. of ICLR*.

Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato. 2015. Learning longer memory in recurrent neural networks. In *Proc. of ICLR Workshop Track*.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. of INTERSPEECH*.

Michael C Mozer. 1993. Neural net architectures for temporal sequence processing. In *SANTA FE INSTITUTE STUDIES IN THE SCIENCES OF COMPLEXITY-PROCEEDINGS VOLUME-*, volume 15, pages 243–243. ADDISON-WESLEY PUBLISHING CO.

Alexander G Ororbia II, Tomas Mikolov, and David Reitter. 2017. Learning simpler language models with the differential state framework. *Neural computation*, 29(12):3327–3352.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proc. of EACL*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. of the IEEE*, 78(10):1550–1560.

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

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2016. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*.