Revisiting Activation Regularization for Language RNNs

Stephen Merity [1] **Bryan McCann** [1] **Richard Socher** [1]

Abstract

Recurrent neural networks (RNNs) serve as a fundamental building block for many sequence tasks across natural language processing. Recent research has focused on recurrent dropout techniques or custom RNN cells in order to improve performance. Both of these can require substantial modifications to the machine learning model or to the underlying RNN configurations. We revisit traditional regularization techniques, specifically L_2 regularization on RNN activations and slowness regularization over successive hidden states, to improve the performance of RNNs on the task of language modeling. Both of these techniques require minimal modification to existing RNN architectures and result in performance improvements comparable or superior to more complicated regularization techniques or custom cell architectures. These regularization techniques can be used without any modification on optimized LSTM implementations such as the NVIDIA cuDNN LSTM.

1. Introduction

The need for effective regularization methods for RNNs has seen extensive focus in recent years. While application of dropout (Srivastava et al., 2014) to the input and output of an RNN has been shown to be effective (Zaremba et al., 2014), dropout is destructive when naively applied to the recurrent connections of an RNN. When naive dropout is applied to the recurrent connections, it is almost impossible to retain information over long periods of time.

Given this fundamental issue, substantial work has gone into understanding and improving dropout when applied to recurrent connections. Of these techniques, which we shall broadly refer to as recurrent dropout, some specific variations have gained popular usage.

Variational RNNs (Gal & Ghahramani, 2016) drop the same network units at each timestep, as opposed to dropping different network units at each timestep. By performing dropout on the same units at each timestep, destructive loss of the RNN hidden state is avoided and the same information is masked at each timestep.

Rather than dropping units, another tactic is to drop updates to given network units. Semeniuta et al. (2016) perform dropout on the input gate of the LSTM (Hochreiter & Schmidhuber, 1997) but allow the forget gate to discard portions of the existing hidden state. Zoneout (Krueger et al., 2016) prevents hidden state updates from occurring by setting a randomly selected subset of network unit activations in h_{t+1} to be equal to the previous activations from h_t. Both of these act to prevent updates to the hidden state while preserving existing content.

On an extreme end, work has also been done to restrict the recurrent matrices in an RNN in order to limit their computational capacity. Some RNN architectures only allow element-wise interactions (Balduzzi & Ghifary, 2016; Bradbury et al., 2016; Seo et al., 2016), removing the recurrent matrix entirely, while others act to restrict the capacity by parameterizing the recurrent matrix (Arjovsky et al., 2016; Wisdom et al., 2016; Jing et al., 2016).

Other forms of regularization explicitly act upon activations such as such as batch normalization (Ioffe & Szegedy, 2015), recurrent batch normalization (Cooijmans et al., 2016), and layer normalization (Ba et al., 2016). These all introduce additional training parameters and can complicate the training process while increasing the sensitivity of the model. Norm stabilization (Krueger & Memisevic, 2015) penalizes the model when the norm of an RNN's hidden state changes substantially between timesteps, achieving strong results in character language modeling on and phoneme recognition.

In this work, we revisit L_2 regularization in the form of activation regularization (AR) and temporal activation regularization (TAR). When applied to modern baselines that do not contain recurrent dropout or normalization techniques, AR and TAR achieve comparable or superior results.

Compared to other invasive regularization techniques

[1]Salesforce Research, Palo Alto, USA. Correspondence to: Stephen Merity <smerity@salesforce.com>.

Part of 34th International Conference on Machine Learning's Workshop on Learning to Generate Natural Language, Sydney, Australia, 2017.

which may require modifications to the RNN cell itself or complex model changes, both AR and TAR require no substantial modifications to the RNN or model. This enables AR and TAR to be applied to optimized RNN implementations such as the cuDNN LSTM which can be many times faster than naïve but flexible LSTM implementations.

2. Activation Regularization

2.1. L_2 activation regularization (AR)

While L_2 regularization is traditionally used on the weights of machine learning models (L_2 weight decay), it could also be used on the activations. We define AR as

$$\alpha\, L_2(m \odot h_t)$$

where m is the dropout mask used by later parts of the model, $L_2(\cdot) = \|\cdot\|_2$ (L_2 norm), h_t is the output of the RNN at timestep t, and α is a scaling coefficient.

When applied to the output of a dense layer, AR penalizes activations that are substantially away from 0, encouraging the activations to remain small. While acting implicitly rather than explicitly, this has similarities to the various batch or layer normalization techniques.

The L_2 penalty on the RNN activations can be applied to h_t or to $m \odot h_t$ (the dropped output used in the rest of the model). In our experiments, we found that applying AR to $m \odot h_t$ was more effective than applying it to neurons not updated during the current optimization step.

2.2. Temporal activation regularization (TAR)

Adding a prior that minimizes differences between states has been explored in the past. This broad concept falls under the broad concept of slowness regularization (Hinton, 1989; Földiák, 1991; Luciw & Schmidhuber, 2012; Jonschkowski & Brock, 2015; Wen et al., 2015) which attempts to minimize $L(f(x_t), f(x_{t+1}))$ where L is a loss function describing the distance between $f(x_t)$ and $f(x_{t+1})$ and f is an arbitrary mapping function.

Temporal activation regularization (TAR) is a direct descendant of this slowness regularization, minimizing

$$\beta\, L_2(h_t - h_{t+1})$$

where $L_2(\cdot) = \|\cdot\|_2$ (L_2 norm), h_t is the output of the RNN at timestep t, and β is a scaling coefficient.

TAR penalizes any large changes in hidden state between timesteps, encouraging the model to keep the output as consistent as possible. For the LSTM, the hidden state which is regularized is only h_t, not the long term memory c_t, though this could optionally be regularized in a similar manner.

Model	Parameters	Validation
$\alpha = 0$	13M	78.4
$\alpha = 1$	13M	76.2
$\alpha = 3$	13M	73.9
$\alpha = 5$	13M	73.7
$\alpha = 7$	13M	73.0
$\alpha = 9$	13M	74.0

Table 1. Results over the Penn Treebank for testing α coefficients for AR with base model $h = 650, \beta = 0, \text{dp} = 0.5, \text{dp}_h = 0.5$.

Model	Parameters	Validation
$\beta = 0$	13M	78.4
$\beta = 1$	13M	77.2
$\beta = 3$	13M	75.2
$\beta = 5$	13M	74.4
$\beta = 7$	13M	74.1
$\beta = 9$	13M	74.7

Table 2. Results over the Penn Treebank for testing β coefficients for TAR with base model $h = 650, \alpha = 0, \text{dp} = 0.5, \text{dp}_h = 0.5$.

3. Experiments

3.1. Language Modeling

We benchmark activation regularization (AR) and temporal activation regularization (TAR) applied to a strong non-variational LSTM baseline[1]. The experiment uses a preprocessed version of the Penn Treebank (PTB) (Mikolov et al., 2010) and WikiText-2 (WT2) (Merity et al., 2016). All hyperparameters, including α for AR and β for TAR, are optimized over the validation dataset. The best found hyperparameters as determined by the validation results are then run on the test set.

PTB: As the Penn Treebank is a small dataset, preventing overfitting is of considerable importance and a major focus of research. Almost all competitive models rely upon a form of recurrent dropout to ensure the RNN does not overfit through drastic changes in the hidden state. Other aggressive dropout techniques, such as performing dropout on the embedding layer such that entire words are dropped from a sequence, are also frequently used.

WT2: WikiText-2 is a dataset approximately twice as large as PTB but with a vocabulary three times larger. The text is also tokenized and processed in a manner similar to datasets used for machine translation using the Moses tokenizer (Koehn et al., 2007).

[1]PyTorch Word Level Language Modeling example: https://github.com/pytorch/examples/tree/master/word_language_model

Model	Parameters	Validation	Test
PTB, LSTM (tied) $h = 650, \text{dp} = 0.5, \text{dp}_h = 0.4$	13M	78.2	74.8
PTB, LSTM (tied) $h = 950, \text{dp} = 0.6, \text{dp}_h = 0.5$	24M	75.3	72.2
PTB, LSTM (tied) $h = 1500, \text{dp} = 0.75, \text{dp}_h = 0.5$	51M	71.3	68.3
PTB, LSTM (tied) $h = 650, \alpha = 5, \beta = 2, \text{dp} = 0.5, \text{dp}_h = 0.4$	13M	72.0	68.9
PTB, LSTM (tied) $h = 950, \alpha = 6, \beta = 4, \text{dp} = 0.6, \text{dp}_h = 0.5$	24M	70.2	66.9
PTB, LSTM (tied) $h = 1500, \alpha = 4, \beta = 4, \text{dp} = 0.75, \text{dp}_h = 0.5$	51M	68.2	65.4

Table 3. Single model perplexity results over the Penn Treebank. Models noting *tied* use weight tying on the embedding and softmax weights. The top section contain models without AR or TAR with the bottom section containing equivalent models using them.

Model	Parameters	Validation	Test
Inan et al. (2016) - Variational LSTM (tied) ($h = 650$)	28M	92.3	87.7
Inan et al. (2016) - Variational LSTM (tied) ($h = 650$) + augmented loss	28M	91.5	87.0
WT2, LSTM (tied) $h = 650, \text{dp} = 0.5, \text{dp}_h = 0.4$	28M	88.8	84.9
WT2, LSTM (tied) $h = 650, \alpha = 5, \beta = 2, \text{dp} = 0.5, \text{dp}_h = 0.4$	28M	85.8	81.8

Table 4. Results over WikiText-2. The increases in parameters compared to the models on PTB are due to the larger vocabulary. Models noting *tied* use weight tying on the embedding and softmax weights.

Experiment details: All experiments use a model containing a two layer RNN. The AR and TAR loss are only applied to the output of the final RNN layer, not to all layers. For the majority of experiments, we follow the medium model size of Zaremba et al. (2014): a two layer RNN with 650 hidden units in each layer.

For training the model, stochastic gradient descent (SGD) without momentum was used for up to 80 epochs. The learning rate began at 20 and was divided by four each time validation perplexity failed to improve. L_2 weight regularization of 10^{-7} was used over all weights in the model and gradients with norm over 10 were rescaled. Batches consist of 20 examples with each example containing 35 timesteps. The loss was averaged over all examples and timesteps. All embedding weights were uniformly initialized in the interval $[-0.1, 0.1]$ and all other weights were initialized between $[-\frac{1}{\sqrt{H}}, \frac{1}{\sqrt{H}}]$, where H is the hidden size.

For dropout, we have two different parameters, dp and dp_h. dp is the dropout rate used on the word vectors and the final RNN output. dp_h is the dropout rate used on the connection between RNN layers. All models use weight tying between the embedding and softmax layer (Inan et al., 2016; Press & Wolf, 2016).

Evaluating AR and TAR independently on PTB: To understand the potential of AR and TAR, we investigate their impact on language model perplexity when used independently in Table 1 (AR) and Table 2 (TAR). While both result in a substantial reduction in perplexity, AR results in the strongest improvement of 5.3, while TAR only achieves 4.3. The drops achieved by this are equivalent to using an LSTM model with twice as many parameters - a substantial improvement given the simplicity of AR and TAR.

Evaluating AR and TAR jointly on PTB: When both AR and TAR are used together, we found the best result was achieved by decreasing α and β, likely as the model was over-regularized otherwise. In Table 3 we present PTB results for three different model sizes comparing models without AR/TAR to those which use both. The model sizes $h \in [650, 950, 1500]$ were chosen to be comparable in size to other published results. With both AR and TAR, the smallest model has an improvement of 6.2 over the baseline model. The improvements continue for the two larger size models, $h = 950$ and $h = 1500$, though the gains fall off as the model size is increased.

Comparing to state-of-the-art PTB: In Table 5 we summarize the current state of the art models in language modeling over the Penn Treebank.

The largest LSTM we train ($h = 1500$) achieves comparable results to the Recurrent Highway Network (RHN) (Zilly et al., 2016), a human developed custom RNN architecture, but with approximately double the number of parameters. Although the LSTM uses twice as many parameters, the RHN runs a cell 10 times per timestep (referred to as recurrence depth), resulting in far more computation. This would likely result in the RHN being slower than the larger LSTM model during both training and prediction, especially when factoring in optimized LSTM implementations such as NVIDIA's cuDNN LSTM.

Model	Parameters	Validation	Test
Zaremba et al. (2014) - LSTM (medium)	20M	86.2	82.7
Zaremba et al. (2014) - LSTM (large)	66M	82.2	78.4
Gal & Ghahramani (2016) - Variational LSTM (medium)	20M	81.9 ± 0.2	79.7 ± 0.1
Gal & Ghahramani (2016) - Variational LSTM (medium, MC)	20M	–	78.6 ± 0.1
Gal & Ghahramani (2016) - Variational LSTM (large)	66M	77.9 ± 0.3	75.2 ± 0.2
Gal & Ghahramani (2016) - Variational LSTM (large, MC)	66M	–	73.4 ± 0.0
Kim et al. (2016) - CharCNN	19M	–	78.9
Merity et al. (2016) - Pointer Sentinel-LSTM	21M	72.4	70.9
Inan et al. (2016) - Variational LSTM (tied) + augmented loss	24M	75.7	73.2
Inan et al. (2016) - Variational LSTM (tied) + augmented loss	51M	71.1	68.5
Zilly et al. (2016) - Variational RHN (tied)	23M	67.9	65.4
Zoph & Le (2016) - NAS Cell (tied)	25M	–	64.0
Zoph & Le (2016) - NAS Cell (tied)	54M	–	62.4
PTB, LSTM (tied) $h = 650, \alpha = 5, \beta = 2, \text{dp} = 0.5, \text{dp}_h = 0.4$	13M	72.0	68.9
PTB, LSTM (tied) $h = 950, \alpha = 6, \beta = 4, \text{dp} = 0.6, \text{dp}_h = 0.5$	24M	70.2	66.9
PTB, LSTM (tied) $h = 1500, \alpha = 4, \beta = 4, \text{dp} = 0.75, \text{dp}_h = 0.5$	51M	68.2	65.4

Table 5. Single model perplexity on validation and test sets for the Penn Treebank language modeling task. Models noting *tied* use weight tying on the embedding and softmax weights.

We also compare to the Neural Architecture Search (NAS) cell (Zoph & Le, 2016). While Zoph & Le (2016) do not report any of the hyperparameters or what type of dropout they used for their Penn Treebank result, they do note that they performed an extensive hyperparameter search over learning rate, weight initialization, dropout rates, and decay epoch in order to produce their best performing model. It is possible that a large contributor to their improved result was in these tuned hyperparameters as they did not compare their NAS cell results to a standard or variational LSTM cell that was subjected to the same extensive hyperparameter search. Our largest LSTM results are 3 perplexity higher in comparison but have not undergone extensive hyperparameter search, do not use additional regularization techniques such as recurrent or embedding dropout, and do not use a custom RNN cell.

WikiText-2 Results: We compare our WikiText-2 results to Inan et al. (2016) who introduced weight tying between the embedding and softmax weights. While we did not perform any hyperparamter search over the coefficient values of α and β for AR and TAR, instead using the best results from PTB, we find them to still be quite effective. The baseline LSTM already achieves a 2.1 perplexity improvement over the variational LSTM models from Inan et al. (2016), including one which uses an augmented loss that modifies standard cross entropy with temperature and a KL divergence based loss. When the AR and TAR parameters optimized over PTB are used, perplexity falls an additional 3.1 perplexity. This is not as strong an improvement as seen on the PTB dataset and may be due to the increased complexity of the dataset (larger vocabulary meaning a longer tail of usage, different genre, and so on) or may just be due to the lack of hyperparamter tuning.

AR and TAR for GRU and $\tanh$ RNN: While neither the GRU (Cho et al., 2014) or $\tanh$ RNN are traditionally used in language modeling, we wanted to see the generality of AR and TAR to other types of RNN cells. We applied the best values of α and β for an LSTM cell to the GRU and $\tanh$ RNN on PTB without any further search in Table 6. These values are likely quite suboptimal but are sufficient for illustrative purposes. For the GRU, perplexity improved by 2.2 from the baseline. This is a positive sign given the impact of these regularization techniques on a GRU are quite different to that of an LSTM. The LSTM only has h_t subjected to AR and TAR, leaving the long term memory c_t unregularized, but the GRU uses h_t both as output at that timestep and as the hidden state input for the next timestep. For the $\tanh$ RNN, the model did not train to acceptable levels at all without the application of AR and TAR. For the $\tanh$ RNN, TAR likely forced the recurrent matrix to learn an identity function in order to ensure h_t could produce h_{t+1}. This would be important given the weights in this model were randomly initialized and suggests TAR acts as an implicit identity initialization constraint (Le et al., 2015).

4. Conclusion

In this work, we revisit L_2 regularization in the form of activation regularization (AR) and temporal activation regularization (TAR). While simple to implement, activity regularization and temporal activity regularization are com-

Model	Parameters	Validation	Test
PTB, RNN (tied) $h = 650, \text{dp} = 0.5, \text{dp}_h = 0.4$	13M	712.3	667.5
PTB, RNN (tied) $h = 650, \alpha = 5, \beta = 2, \text{dp} = 0.5, \text{dp}_h = 0.4$	13M	232.1	227.8
PTB, GRU (tied) $h = 650, \text{dp} = 0.5, \text{dp}_h = 0.4$	13M	86.1	83.3
PTB, GRU (tied) $h = 650, \alpha = 5, \beta = 2, \text{dp} = 0.5, \text{dp}_h = 0.4$	13M	83.9	81.1

Table 6. Single model perplexity results over the Penn Treebank for $\tanh$ RNN and GRU. Neither cell are traditionally used for language modeling but this demonstrates the generality for AR (α) and TAR (β). Values for α, β taken from best LSTM model with no search. Models noting *tied* use weight tying on the embedding and softmax weights.

petitive with other far more complex regularization techniques and offer equivalent or better results. The improvements that these techniques provide can likely be combined with other regularization techniques, such as the variational LSTM, and may lead to further improvements in performance as well, especially if subjected to an extensive hyperparameter search.

Sample generated text

For generating text samples, words were sampled using the standard generation script contained in the PyTorch word level language modeling example. WikiText-2 was used given the larger vocabulary and more realistic looking text. Neither the $\langle eos \rangle$ token nor the $\langle unk \rangle$ were allowed to be selected. Each paragraph is a separate sample of text with the tokens following Moses (Koehn et al., 2007), joining words with @-@ and dot-decimal split to a @.@ token.

" Something Borrowed " is the second episode of the fourth season of the American comedy television series The X @-@ Files . The episode was written by David McCarthy and directed by Mark Sacks . It aired in the United States on November 30 , 2011 , as a two @-@ episode episode, watched by 4 @.@ 9 million viewers and was the highest rated show on the Fox network .

The work of Olivier 's , a large 1950s table with the center of a vinyl beam , was used for bony motifs from the upper @-@ production model via the Club van X . The modified works were released in the museum , which gave its namesake to the visual designers in Hong Kong .

The first prototype was released for the PlayStation 4 , containing the 2 @.@ 5 part series , with 3 @.@ 5 million copies sold . In October 2010 , Activision announced that both the game and the main gameplay was " downloadable " . The first game , titled Snow : The Game of the Battlefield 2 : The Ultimate Warrior , was the third anime game , and was released in August 2016 .

The German Land Forces had been reversed in the early 1990s , although the Soviet Union continued to deter NDH forces in the nation . The area was moved to Sarajevo , and the troops were despatched to the National Register of Historic Places in the summer of 1918 for the establishment of full political and social parties . The Polish language was protected by the Soviet Union , which was the first Polish continental conflict of the newly formed Union in North America , and the Polish Front with the last of the Polish Communist Party .

References

Arjovsky, Martin, Shah, Amar, and Bengio, Yoshua. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.

Ba, Jimmy, Kiros, Jamie Ryan, and Hinton, Geoffrey E. Layer normalization. *CoRR*, abs/1607.06450, 2016.

Balduzzi, David and Ghifary, Muhammad. Strongly-typed recurrent neural networks. *arXiv preprint arXiv:1602.02218*, 2016.

Bradbury, James, Merity, Stephen, Xiong, Caiming, and Socher, Richard. Quasi-Recurrent Neural Networks. *arXiv preprint arXiv:1611.01576*, 2016.

Cho, Kyunghyun, Van Merriënboer, Bart, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Cooijmans, Tim, Ballas, Nicolas, Laurent, César, and Courville, Aaron C. Recurrent batch normalization. *CoRR*, abs/1603.09025, 2016.

Földiák, Peter. Learning invariance from transformation sequences. *Neural Computation*, 3(2):194–200, 1991.

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

Hinton, Geoffrey E. Connectionist learning procedures. *Artificial intelligence*, 40(1-3):185–234, 1989.

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

Inan, Hakan, Khosravi, Khashayar, and Socher, Richard. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. *arXiv preprint arXiv:1611.01462*, 2016.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Jing, Li, Shen, Yichen, Dubček, Tena, Peurifoy, John, Skirlo, Scott, Tegmark, Max, and Soljačić, Marin. Tunable Efficient Unitary Neural Networks (EUNN) and their application to RNN. *arXiv preprint arXiv:1612.05231*, 2016.

Jonschkowski, Rico and Brock, Oliver. Learning state representations with robotic priors. *Auton. Robots*, 39:407–428, 2015.

Kim, Yoon, Jernite, Yacine, Sontag, David, and Rush, Alexander M. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Koehn, Philipp, Hoang, Hieu, Birch, Alexandra, Callison-Burch, Chris, Federico, Marcello, Bertoldi, Nicola, Cowan, Brooke, Shen, Wade, Moran, Christine, Zens, Richard, Dyer, Chris, Bojar, Ondej, Constantin, Alexandra, and Herbst, Evan. Moses: Open source toolkit for statistical machine translation. In *ACL*, 2007.

Krueger, David and Memisevic, Roland. Regularizing rnns by stabilizing activations. *CoRR*, abs/1511.08400, 2015.

Krueger, David, Maharaj, Tegan, Kramár, János, Pezeshki, Mohammad, Ballas, Nicolas, Ke, Nan Rosemary, Goyal, Anirudh, Bengio, Yoshua, Larochelle, Hugo, Courville, Aaron, et al. Zoneout: Regularizing RNNss by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

Le, Quoc V, Jaitly, Navdeep, and Hinton, Geoffrey E. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Luciw, Matthew and Schmidhuber, Juergen. Low complexity proto-value function learning from sensory observations with incremental slow feature analysis. *Artificial Neural Networks and Machine Learning–ICANN 2012*, pp. 279–287, 2012.

Merity, Stephen, Xiong, Caiming, Bradbury, James, and Socher, Richard. Pointer Sentinel Mixture Models. *arXiv preprint arXiv:1609.07843*, 2016.

Mikolov, Tomas, Karafiát, Martin, Burget, Lukás, Cernocký, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

Press, Ofir and Wolf, Lior. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Semeniuta, Stanislau, Severyn, Aliaksei, and Barth, Erhardt. Recurrent dropout without memory loss. In *COLING*, 2016.

Seo, Minjoon, Min, Sewon, Farhadi, Ali, and Hajishirzi, Hannaneh. Query-Reduction Networks for Question Answering. *arXiv preprint arXiv:1606.04582*, 2016.

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

Wen, Tsung-Hsien, Gasic, Milica, Mrksic, Nikola, Su, Pei-Hao, Vandyke, David, and Young, Steve. Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems. *arXiv preprint arXiv:1508.01745*, 2015.

Wisdom, Scott, Powers, Thomas, Hershey, John, Le Roux, Jonathan, and Atlas, Les. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 4880–4888, 2016.

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

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

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