INTERSPEECH 2017
August 20–24, 2017, Stockholm, Sweden

Residual Memory Networks in Language Modeling: Improving the Reputation of Feed-Forward Networks

Karel Beneš, Murali Karthick Baskar, Lukáš Burget

Brno University of Technology, Speech@FIT and IT4I Center of Excellence, Czechia
{ibenes,baskar,burget}@fit.vutbr.cz

Abstract

We introduce the Residual Memory Network (RMN) architecture to language modeling. RMN is an architecture of feed-forward neural networks that incorporates residual connections and time-delay connections that allow us to naturally incorporate information from a substantial time context. As this is the first time RMNs are applied for language modeling, we thoroughly investigate their behaviour on the well studied Penn Treebank corpus. We change the model slightly for the needs of language modeling, reducing both its time and memory consumption. Our results show that RMN is a suitable choice for small-sized neural language models: With test perplexity 112.7 and as few as 2.3M parameters, they out-perform both a much larger vanilla RNN (PPL 124, 8M parameters) and a similarly sized LSTM (PPL 115, 2.08M parameters), while being only by less than 3 perplexity points worse than twice as big LSTM.

Index Terms: residual memory networks, feed-forward networks, language modeling

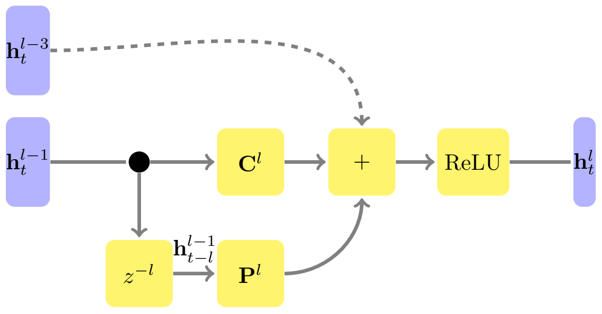

Figure 1: *Computation of a single hidden vector $\boldsymbol{h}_t^l$ in Residual Memory Network (RMN), at time t and layer l. The z^{-l} block delays its input by l timesteps. Therefore, to compute $\boldsymbol{h}_t^l$, values of the previous hidden layer from the current time t and the history time $t - l$ are used. The dashed line represents residual connection; it is present in every third layer only.*

1. Introduction

When first introduced to modern language modeling by Bengio et al. [1], neural language models brought a principal improvement over n-gram counting techniques: Instead of treating individual words as unrelated events, these neural models learn to represent words by continuous-valued vectors of fixed length. This way, the rest of the network is independent of the exact identity of input words and neural language models are thus able to partially overcome the curse of dimensionality.

The next fundamental step was taken by Mikolov et al. [2], who have compressed whole histories of words into a fixed length representation in their recurrent neural networks (RNN). Since then, research activity was focused on recurrent networks:

The "vanilla RNNs", using sigmoid activation functions, were soon replaced by more sophisticated LSTMs [3], that achieved yet better results, arguably due to their ability to effectively learn long term dependencies. Up to this point, all experiments indicated that increasing the model size inevitably leads to overfitting.

This obstacle was overcome by the work of Zaremba et al. [4], who successfully applied dropout to LSTMs. Since then, further improvements were achieved by adding cache-like mechanisms [5, 6].

Feed-forward networks were left aside of these improvements. Only recently, a recurrence-free Gated Convolutional Network model [7] was proposed. This sophisticated model consists of many wide layers of multiplicative "gating" units. Its authors have shown it to perform on par with big LSTM networks.

We approach the problem from a different angle, looking for inspiration into acoustic modeling: Here, RNNs have also replaced deep neural networks as the state of the art tool [8, 9, 10]. However, Baskar et al. have recently proposed Residual Memory Network (RMN) [11], a feed-forward architecture capable of reaching better WER than several stacked LSTM layers. This architecture is actually a DNN with layers cleverly spread over time, thus able to model temporal dependencies. Also, residual connections [12] are employed to allow training deep models.

We were therefore curious: is Residual Memory Network a competitive language model as well? We will show that RMNs perform more like recurrent networks than the shallow feed-forward networks.

The work was supported by Technology Agency of the Czech Republic project No. TA04011311 "MINT", European Union's Horizon 2020 project No. 645523 BISON, and by Czech Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science – LQ1602".

2. Residual Memory Network Model

The Residual Memory Network[1] architecture was proposed as an improvement for deep neural networks, that will allow them to model long temporal contexts. The output of l-th hidden layer at time t is given as:

$$\boldsymbol{h}_t^l = \mathrm{ReLU}(\boldsymbol{C}^l \boldsymbol{h}_t^{l-1} + \boldsymbol{P}\boldsymbol{C}^l \boldsymbol{h}_{t-l}^{l-1} + \boldsymbol{b}^l + \boldsymbol{h}_t^{l-3}) \qquad (1)$$

Here $\boldsymbol{h}_t^l$ stands for the hidden vector at time t and l-th layer; weight matrices $\boldsymbol{C}^l$ and $\boldsymbol{P}$ stand respectively for transformation

[1]This model has no direct relation to "Convolutional Residual Memory Networks" proposed by Moniz and Pal for computer vision [13], who use an explicit memory component represented by an LSTM running along the depth of the network.

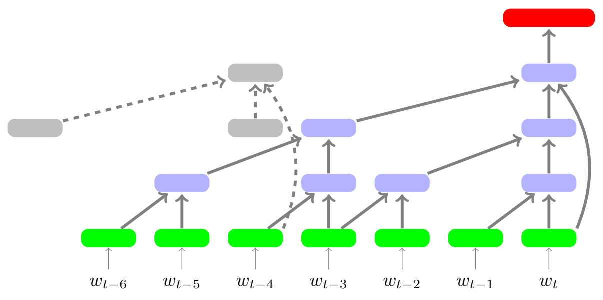

Figure 2: *Three-layer Residual Memory Network predicting a single word w_{t+1}, unrolled in time. Green nodes represent replacing word index by a word vector. Every blue node represents a single hidden vector $\boldsymbol{h}_t^l$; the red node is the output. Relating this illustration to Figure 1, weight matrices are associated with arrows, while summation and nonlinearity are within the blue nodes. The bent line in the right represents residual connection; if the network was deeper, as suggested by grey $\boldsymbol{h}_{t-4}^3$ node, residual connections would appear at other times as well.*

of current and past representation at the previous layer. The residual connection (grey) is present only in every third layer.

The temporal context is captured by delay connections (second term in Eq. (1)), which fast-forward values of hidden vectors through time. Lag of these delay connections changes over the depth of the network, helping to increase the level of abstraction in successive layers.

We use RMN in a slightly simpler definition (illustrated in Figure 1):

$$\boldsymbol{h}_t^l = \mathrm{ReLU}(\boldsymbol{C}^l \boldsymbol{h}_t^{l-1} + \boldsymbol{P}^l \boldsymbol{h}_{t-l}^{l-1} + \boldsymbol{b}^l + h_t^{l-3}) \qquad (2)$$

The formulation (2) directly contrasts RMN to RNN: Instead of combining the input of a given layer with its previous output, the input is combined with a historical input. The effect of accessing historical representation from previous layer stacks — the historical input depends on yet older outputs of the layer before. Thus the length of the input considered by the network, although finite, is significant (Section 2.2).

Being a feed-forward network dealing with sequences, RMN under formulation (2) can also be viewed as a special variant of Time Delay Neural Network (TDNN) [14]. Differences from TDNNs, as used recently for acoustic modeling [15], include adding residual connections, orienting the model at depth rather than width and considering only previous inputs.

2.1. Residual Memory Networks Unrolled in Time

Similar to RNNs, we can view the RMN architecture unrolled in time, as illustrated in Figure 2. This perspective offers two additional insights:

Every (nonresidual) path from an input word to the output is of the same length. Thus all words are nearly equal in terms of how difficult it is to backpropagate gradients to them. This is likely to be part of explanation why are the RMNs able to outperform RNNs.

Since the projections $\boldsymbol{C}^l$ and $\boldsymbol{P}^l$ are shared across different timesteps, every hidden representation has to be usable not only in the current timestep but also in those following. This helps RMNs to avoid the curse of dimensionality, much like RNNs.

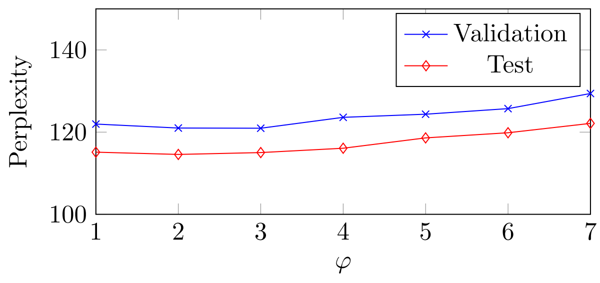

Figure 3: *Dependence of the model performance on the frequency of look-back increase φ. With shorter models (higher φ), the performance degrades, the effect is consistent since* $\varphi = 5$.

2.2. Reducing the Temporal Span

As the length of history look-back is increased with every layer, the input temporal span on RMN is proportional to the square of the number of layers, exactly as $\frac{1}{2}L(L+1)+1$, where L is depth of the network. For example, for a 15-layered RMN, sequence of 121 words is considered as the input. However, all previous results of neural language modeling indicate, that such context can not be effectively exploited without some form of cache mechanism. Also, so wide input requires considerable memory and computational effort. Therefore we aim to reduce it.

Formally, we replace the original definition (2) of a RMN layer by:

$$\boldsymbol{h}_t^l = \mathrm{ReLU}(\boldsymbol{C}^l \boldsymbol{h}_t^{l-1} + \boldsymbol{P}^l \boldsymbol{h}_{t-D(l)}^{l-1} + \boldsymbol{b}^l + h_t^{l-3}) \qquad (3)$$

Here $D : \mathbb{N}^+ \to \mathbb{N}^+$ is an arbitrary function. For $D(l) = l$, the original model is recovered; but we want to set it so that it grows slower. We achieve this in a parametrized way by introducing a *frequency of look-back increase* (φ), which is basically a scaling factor. $D(l)$ is then defined as: $D(l) = 1 + \left\lfloor \frac{l-1}{\varphi} \right\rfloor$.

For instance by setting $\varphi = 4$: Layers 1 to 4 take input from their respective timestep ($\boldsymbol{h}_t^{l-1}$) and the previous one ($\boldsymbol{h}_{t-1}^{l-1}$); layers 5 to 8 combine the $\boldsymbol{h}_t^{l-1}$ and $\boldsymbol{h}_{t-2}^{l-1}$ and so on.

Using the φ equal to $\hat{\varphi}$ results in reducing the context at most $\hat{\varphi}$-times. For example, with 15 layers and $\varphi = 4$, the length of the context drops from 121 to 37 words.

In practice, values of φ up to 4 typically grant nice memory savings without hindering the performance.

3. Testing on the Penn Treebank

We explore abilities of the model on the well studied Penn Treebank [16], as preprocessed by Mikolov [17]. Specifically, the training set consists of 930k tokens, validation set of 73k tokens, and test set of 82k tokens. The vocabulary is reduced to 10k most common words, the rest is replaced by an `<unk>` token.

We have implemented the model in Keras [18], using Theano [19] as backend. We found the model easier to optimize using Adam [20] than with plain stochastic gradient descent (momentum included).

Inspired by the practice of [21], we take advantage of the feed-forward nature of the model and create minibatches from randomly shuffled data.

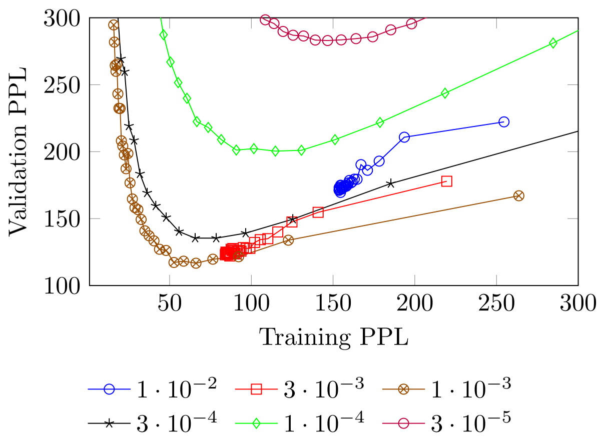

Figure 4: *Influence of the learning rate α; training progresses from right to left. Learning rates from $1 \cdot 10^{-3}$ down result in over-training, higher ones end up under-fitting the training data. The best result is achieved right on the edge of overfitting by the model optimized with learning rate $1 \cdot 10^{-3}$.*

In preliminary experiments, we made three general findings:

1. Using batch normalization (BN) [22] helps the model to learn significantly faster. Adding BN also typically leads to slight decrease in final perplexity. We add BN right before every nonlinearity in the model, except for the output softmax.
2. We have observed that increasing the size of batches improves not only the speed of computation, but also the best achieved validation perplexity. Thus we have settled on using minibatches of 256 samples.
3. The best results were achieved with residuals skipping 2 nonlinearities; with skipping 1 nonlinearity being closely behind.

3.1. Size of the model

The first set of our experiments focuses on the size of the model, evaluating 3 hyperparameters:

1. **Network depth:** The performance of RMNs is not very sensitive to network depth. When increasing the depth from small values (e.g. 3), perplexity drops are achieved. Then, networks in range 12–18 give very similar results. Networks deeper than 18 layers do not show performance problems, but are difficult to fit into memory when using big minibatches.
2. **Hidden layer width:** Again, we found RMNs not very sensitive to this parameter. Networks with 100 to 300 hidden units per layer were found reasonable. However, as hidden layers widen, it becomes more difficult to train the model, eventually hindering both training and validation performance.
3. **Frequency of look-back increase φ:** Performance of networks was quite invariant to small values of φ (see Figure 3). Therefore we took an aggressive approach with $\varphi = 4$, which led to networks considering shortest input history while maintaining good performance.

We selected following configuration for successive experiments: 15 hidden layers, 256 units each, $\varphi = 4$.

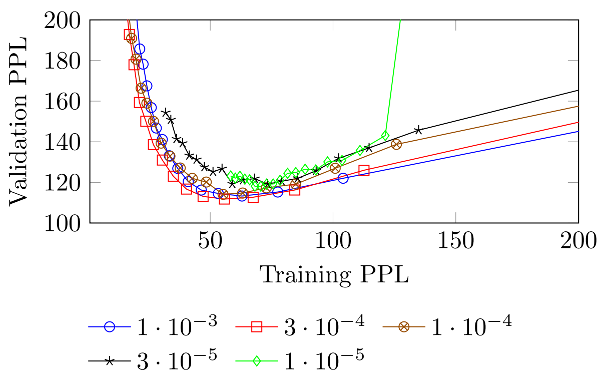

Figure 5: *Progress during training (right to left), as dependent on the decay rate k; the initial learning rate was set to $\alpha = 1 \cdot 10^{-3}$. Learning was stopped after 20 epochs in all cases. Training with small decay allows near-perfect fit to the training data, but consistently misses the high-quality validation valley. The best result was obtained with decay $3 \cdot 10^{-4}$. Using stronger decay prevents the model from learning training data so well and model trained this way also misses the valley of best validation performance.*

3.2. Regularizing the model

Although batch normalization was reported to act as regularizer [22], we found some of our reasonably small models to nearly learn the training corpus (reaching training PPL under 5), while losing any generalization power.

Thus we experimented with additional regularization: dropout [23], word-dropout [24] and MaxNorm [23, 25]. None of these schemes prevented overfitting while achieving good performance at the same time.

Therefore, we followed the regularization practice of [11] and applied L2 regularization. This regularization was found to be a crucial element in the training procedure, with reasonable values of β being in range from $3 \cdot 10^{-5}$ to $3 \cdot 10^{-4}$.

Finally, we only regularized the L2 norm of our model, with regularization weight $\beta = 10^{-4}$.

3.3. The learning procedure

Having set the model architecture and regularization, we move our attention to the learning procedure. Using the Adam optimizer, we left the momentum controlling parameters β_1, β_2 to their suggested values (0.9, 0.999) and focused on the learning rate α.

In an experiment with fixed learning rate (Figure 4), we found out there is a critical point between values $\alpha = 1 \cdot 10^{-3}$ and $\alpha = 3 \cdot 10^{-4}$. Using bigger learning rates hinders learning, keeping both training and validation perplexity high. Setting the learning rate to smaller values leads to severe overfitting, but reasonable solution can be obtained by early stopping. An interesting fact is that although most small learning rates eventually converge to a similar performance on the training set (around PPL 6), only those close to $\alpha = 3 \cdot 10^{-4}$ pass through a region of low validation perplexity.

Next, we experimented with learning rate scheduling, by

Table 1: *Comparison of RMNs to previously published results*

Model	# parameters	test PPL
KN-smoothed 5-grams [17]		141.2
shallow feed-forward NN [17]	around[1] 2 M	140.2
vanilla RNN[2]	8.2 M	123.3
small LSTM [26]	2.0 M	115
SCRN [26]	2.8 M	115
small RMN	2.3 M	112.7
medium LSTM [27]	5 M	109.7
medium RMN	7.1 M	107.0
big LSTM [27]	20 M	79.8

[1] Exact size of the model unknown; estimate based on experiments in [1].
[2] Trained in-house with Mikolov's original tool `rnnlm`.

using a continuous learning rate decay, parametrized as:

$$\alpha_t = \frac{\alpha_0}{1 + k \cdot u} \tag{4}$$

where α_0 is the initial learning rate, k is the decay constant and u is the learning time, measured in number of parameter updates.

Results of this experiment are summarized in Figure 5. Pattern similar to Figure 4 emerges, with a valley of good validation performance around training perplexity 60. The best results here are better than in the experiment with the fixed learning rate; we obtain our best overall result here, as the model that achieved validation PPL 111.8 with $k = 3 \cdot 10^{-4}$, has reached test perplexity 107.0.

3.4. Comparison with models in literature

Finally, Table 1 compares the performance of RMNs with other language models found in the literature. RMNs are represented by (1) a small model with 100 hidden units per layer and (2) a medium sized one with layer width 256. Parameters of both these models were obtained with decay $k = 3 \cdot 10^{-4}$.

It immediately follows from Table 1 that the additional computational strength of RMNs puts them well above the shallow feed-forward networks.

Comparing RMNs to recurrent networks, RMNs do their job rather well, outperforming not only much larger vanilla RNN, but also an similarly sized LSTM and Structurally Constrained Recurrent Network (SCRN) [26].

Considering the shallow feed-forward network as baseline for both RNN and RMN, we see the hierarchical structure of RMN brings much greater gain, while the fixed input scope is not limiting its capacity. Taking a message from these results, it seems that having multiple nonlinear transformations has greater direct effect than just compressing all history into a single vector and increasing its length.

Increasing the model size, we see that RMNs scale similarly to LSTMs. However as discussed in 3.1, there is a limit to scaling RMNs, leaving gap between best RMN and large, dropout-regularized LSTMs [27].

4. Conclusions

We have successfully applied Residual Memory Networks (RMN) to language modeling. Despite the fact that RMNs are purely feed-forward and free of any gating-like multiplication, we have found the model competitive to similar sized RNNs, including LSTMs.

Specifically, we have obtained test perplexity 112.7 on the Penn Treebank, using a small model using 100 units per hidden layer. Enlarging the model to 256 hidden units per layer, RMN reaches perplexity 107.0, thus outperforming appropriately scaled LSTM.

We were not able to train large models yet, but given the experience in acoustic modeling, we believe this is still possible. Next we aim to (1) explore the effects of adding multiplicative connections into the architecture and (2) to compare our architecture with the original TDNN in detail, seeking some conclusion on limits of similar feed-forward architectures.

5. References

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A Neural Probabilistic Language Model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Mar. 2003.

[2] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, 2010, pp. 1045–1048.

[3] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM Neural Networks for Language Modeling," in *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*. Portland, OR, USA: ISCA, Sep. 2012, pp. 194–197.

[4] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[5] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer Sentinel Mixture Models," *CoRR*, vol. abs/1609.07843, 2016. [Online]. Available: http://arxiv.org/abs/1609.07843

[6] E. Grave, A. Joulin, and N. Usunier, "Improving Neural Language Models with a Continuous Cache," *CoRR*, vol. abs/1612.04426, 2016. [Online]. Available: http://arxiv.org/abs/1612.04426

[7] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language Modeling with Gated Convolutional Networks," *CoRR*, vol. abs/1612.08083, 2016.

[8] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014, Florence, Italy, May 4-9, 2014*. IEEE, 2013.

[9] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, 2014, pp. 338–342.

[10] Y. Miao, M. Gowayyed, X. Na, T. Ko, F. Metze, and A. Waibel, "An empirical exploration of CTC acoustic models," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2016, pp. 2623–2627.

[11] M. K. Baskar, M. Karafiát, L. Burget, K. Veselý, F. Grézl, and J. Černocký, "Residual Memory Networks: Feed-forward approach to learn long-term temporal dependencies," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, USA, March 5-9, 2017*, 2017.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 770–778.

[13] J. Moniz and C. J. Pal, "Convolutional residual memory networks," *CoRR*, vol. abs/1606.05262, 2016. [Online]. Available: http://arxiv.org/abs/1606.05262

[14] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Readings in speech recognition," A. Waibel and K.-F. Lee, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, ch. Phoneme Recognition Using Time-delay Neural Networks, pp. 393–404.

[15] V. Peddinti, D. Povey, and S. Khudanpur, "A time delay neural network architecture for efficient modeling of long temporal contexts," in *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, 2015, pp. 3214–3218.

[16] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a Large Annotated Corpus of English: The Penn Treebank," *Comput. Linguist.*, vol. 19, no. 2, pp. 313–330, Jun. 1993.

[17] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernocký, "Empirical Evaluation and Combination of Advanced Language Modeling Techniques." in *INTERSPEECH 2011, 12th Annual Conference of the International Speech Communication Association, Florence, Italy, August 27-31, 2011*. ISCA, 2011, pp. 605–608.

[18] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[19] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[20] D. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in *Proceedings of ICLR*, 2015.

[21] G. Saon, H. Soltau, A. Emami, and M. Picheny, "Unfolded recurrent neural networks for speech recognition," in *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*. ISCA, 2014, pp. 343–347.

[22] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *ICML*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 448–456.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.

[24] M. Iyyer, V. Manjunatha, J. L. Boyd-Graber, and H. D. III, "Deep Unordered Composition Rivals Syntactic Methods for Text Classification." in *ACL (1)*. The Association for Computer Linguistics, 2015, pp. 1681–1691.

[25] N. Srebro, J. D. M. Rennie, and T. S. Jaakola, "Maximum-Margin Matrix Factorization," in *Advances in Neural Information Processing Systems 17*. MIT Press, 2005, pp. 1329–1336.

[26] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato, "Learning Longer Memory in Recurrent Neural Networks," *CoRR*, vol. abs/1412.7753, 2014. [Online]. Available: http://arxiv.org/abs/1412.7753

[27] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An Empirical Exploration of Recurrent Network Architectures," *Journal of Machine Learning Research*, 2015.