Restricted Recurrent Neural Tensor Networks: Exploiting Word Frequency and Compositionality

Alexandre Salle[1]   **Aline Villavicencio**[1,2]

[1]Institute of Informatics, Federal University of Rio Grande do Sul (Brazil)
[2]School of Computer Science and Electronic Engineering, University of Essex (UK)
alex@alexsalle.com   avillavicencio@inf.ufrgs.br

Abstract

Increasing the capacity of recurrent neural networks (RNN) usually involves augmenting the size of the hidden layer, with significant increase of computational cost. Recurrent neural tensor networks (RNTN) increase capacity using distinct hidden layer weights for each word, but with greater costs in memory usage. In this paper, we introduce restricted recurrent neural tensor networks (r-RNTN) which reserve distinct hidden layer weights for frequent vocabulary words while sharing a single set of weights for infrequent words. Perplexity evaluations show that for fixed hidden layer sizes, r-RNTNs improve language model performance over RNNs using only a small fraction of the parameters of unrestricted RNTNs. These results hold for r-RNTNs using Gated Recurrent Units and Long Short-Term Memory.

1   Introduction

Recurrent neural networks (RNN), which compute their next output conditioned on a previously stored hidden state, are a natural solution to sequence modeling. Mikolov et al. (2010) applied RNNs to word-level language modeling (we refer to this model as s-RNN), outperforming traditional n-gram methods. However, increasing capacity (number of tunable parameters) by augmenting the size H of the hidden (or recurrent) layer — to model more complex distributions — results in a significant increase in computational cost, which is $O(H^2)$.

Sutskever et al. (2011) increased the performance of a character-level language model with a multiplicative RNN (m-RNN), the factored approximation of a recurrent neural tensor network (RNTN), which maps each symbol to separate hidden layer weights (referred to as recurrence matrices from hereon). Besides increasing model capacity while keeping computation constant, this approach has another motivation: viewing the RNN's hidden state as being transformed by each new symbol in the sequence, it is intuitive that different symbols will transform the network's hidden state in different ways (Sutskever et al., 2011). Various studies on compositionality similarly argue that some words are better modeled by matrices than by vectors (Baroni and Zamparelli, 2010; Socher et al., 2012). Unfortunately, having separate recurrence matrices for each symbol requires memory that is linear in the symbol vocabulary size ($|V|$). This is not an issue for character-level models, which have small vocabularies, but is prohibitive for word-level models which can have vocabulary size in the millions if we consider surface forms.

In this paper, we propose the Restricted RNTN (r-RNTN) which uses only $K < |V|$ recurrence matrices. Given that $|V|$ words must be assigned K matrices, we map the most frequent $K - 1$ words to the first $K - 1$ matrices, and share the K-th matrix among the remaining words. This mapping is driven by the statistical intuition that frequent words are more likely to appear in diverse contexts and so require richer modeling, and by the greater presence of predicates and function words among the most frequent words in standard corpora like COCA (Davies, 2009). As a result, adding K matrices to the s-RNN both increases model capacity and satisfies the idea that some words are better represented by matrices. Results show that r-RNTNs improve language model performance over s-RNNs even for small K with no computational overhead, and even for small K approximate the performance of RNTNs using a fraction of the parameters. We also exper-

iment with r-RNTNs using Gated Recurrent Units (GRU) (Cho et al., 2014) and Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), obtaining lower perplexity for fixed hidden layer sizes. This paper discusses related work (§2), and presents r-RNTNs (§3) along with the evaluation method (§4). We conclude with results (§5), and suggestions for future work.

2 Related Work

We focus on related work that addresses language modeling via RNNs, word representation, and conditional computation.

Given a sequence of words $(x_1, ..., x_T)$, a language model gives the probability $P(x_t|x_{1...t-1})$ for $t \in [1, T]$. Using a RNN, Mikolov et al. (2010) created the s-RNN language model given by:

$$h_t = \sigma(W_h x_t + U_h h_{t-1} + b_h) \quad (1)$$

$$P(x_t|x_{1...t-1}) = x_t^T Softmax(W_o h_t + b_o) \quad (2)$$

where h_t is the hidden state represented by a vector of dimension H, $\sigma(z)$ is the pointwise logistic function, W_h is the $H \times V$ embedding matrix, U_h is the $H \times H$ recurrence matrix, W_o is the $V \times H$ output matrix, and b_h and b_o are bias terms. Computation is $O(H^2)$, so increasing model capacity by increasing H quickly becomes intractable.

The RNTN proposed by Sutskever et al. (2011) is nearly identical to the s-RNN, but the recurrence matrix in eq. (1) is replaced by a tensor as follows:

$$h_t = \sigma(W_h x_t + U_h^{i(x_t)} h_{t-1} + b_h) \quad (3)$$

where $i(z)$ maps a hot-one encoded vector to its integer representation. Thus the U_h tensor is composed of $|V|$ recurrence matrices, and at each step of sequence processing the matrix corresponding to the current input is used to transform the hidden state. The authors also proposed m-RNN, a factorization of the $U_h^{i(x_t)}$ matrix into $U_{lh} diag(v_{x_t}) U_{rh}$ to reduce the number of parameters, where v_{x_t} is a *factor* vector of the current input x_t, but like the RNTN, memory still grows linearly with $|V|$. The RNTN has the property that input symbols have both a vector representation given by the embedding and a matrix representation given by the recurrence matrix, unlike the s-RNN where symbols are limited to vector representations.

The integration of both vector and matrix representations has been discussed but with a focus on representation learning and not sequence modeling (Baroni and Zamparelli, 2010; Socher et al., 2012). For instance, Baroni and Zamparelli (2010) argue for nouns to be represented as vectors and adjectives as matrices.

Irsoy and Cardie (2014) used m-RNNs for the task of sentiment classification and obtained equal or better performance than s-RNNs. Methods that use conditional computation (Cho and Bengio, 2014; Bengio et al., 2015; Shazeer et al., 2017) are similar to RNTNs and r-RNTNs, but rather than use a static mapping, these methods train gating functions which do the mapping. Although these methods can potentially learn better policies than our method, they are significantly more complex, requiring the use of reinforcement learning (Cho and Bengio, 2014; Bengio et al., 2015) or additional loss functions (Shazeer et al., 2017), and more linguistically opaque (one must learn to interpret the mapping performed by the gating functions).

Whereas our work is concerned with updating the network's hidden state, Chen et al. (2015) introduce a technique that better approximates the output layer's Softmax function by allocating more parameters to frequent words.

3 Restricted Recurrent Neural Tensor Networks

To balance expressiveness and computational cost, we propose restricting the size of the recurrence tensor in the RNTN such that memory does not grow linearly with vocabulary size, while still keeping dedicated matrix representations for a subset of words in the vocabulary. We call these Restricted Recurrent Neural Tensor Networks (r-RNTN), which modify eq. (3) as follows:

$$h_t = \sigma(W_h x_t + U_h^{f(i(x_t))} h_{t-1} + b_h^{f(i(x_t))}) \quad (4)$$

where U_h is a tensor of $K < |V|$ matrices of size $H \times H$, b_h is a $H \times K$ bias matrix with columns indexed by f. The function $f(w)$ maps each vocabulary word to an integer between 1 and K.

We use the following definition for f:

$$f(w) = min(rank(w), K) \quad (5)$$

where $rank(w)$ is the rank of word w when the vocabulary is sorted by decreasing order of unigram frequency.

This is an intuitive choice because words which appear more often in the corpus tend to have more
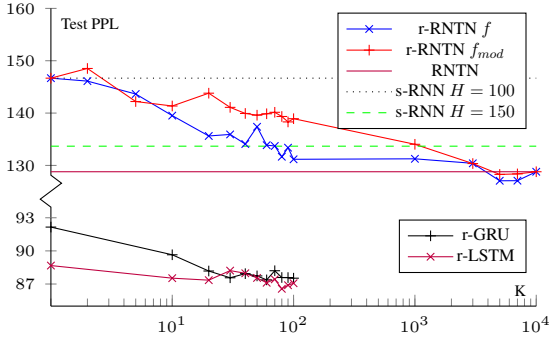
Figure 1: PTB test PPL as K varies from 1 to 10000 (100 for gated networks). At $K = 100$, the r-RNTN with f mapping already closely approximates the much bigger RNTN, with little gain for bigger K, showing that dedicated matrices should be reserved for frequent words as hypothesized.

variable contexts, so it makes sense to dedicate a large part of model capacity to them. A second argument is that frequent words tend to be predicates and function words. We can imagine that predicates and function words transform the meaning of the current hidden state of the RNN through matrix multiplication, whereas nouns, for example, add meaning through vector addition, following Baroni and Zamparelli (2010).

We also perform initial experiments with r-RNTNs using LSTM and GRUs. A GRU is described by

$$r_t = \sigma(W_h^r x_t + U_h^r h_{t-1} + b_h^r) \qquad (6)$$

$$z_t = \sigma(W_h^z x_t + U_h^z h_{t-1} + b_h^z) \qquad (7)$$

$$\tilde{h}_t = \tanh(W_h^h x_t + U_h^h (r_t \odot h_{t-1}) + b_h^h) \quad (8)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \qquad (9)$$

and an LSTM by

$$f_t = \sigma(W_h^f x_t + U_h^f h_{t-1} + b_h^f) \qquad (10)$$

$$i_t = \sigma(W_h^i x_t + U_h^i h_{t-1} + b_h^i) \qquad (11)$$

$$o_t = \sigma(W_h^o x_t + U_h^o h_{t-1} + b_h^o) \qquad (12)$$

$$\tilde{c}_t = \tanh(W_h^c x_t + U_h^c h_{t-1} + b_h^c) \qquad (13)$$

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1} \qquad (14)$$

$$h_t = o_t \odot tanh(c_t) \qquad (15)$$

We create r-RNTN GRUs (r-GRU) by making U_h^h and b_h^h input-specific (as done in eq. (4)). For r-RNTN LSTMs (r-LSTM), we do the same for U_h^c and b_h^c.

4 Materials

We evaluate s-RNNs, RNTNs, and r-RNTNs by training and measuring model perplexity (PPL) on the Penn Treebank (PTB) corpus (Marcus et al., 1994) using the same preprocessing as Mikolov et al. (2011). Vocabulary size is 10000.

For an r-RNTN with $H = 100$, we vary the tensor size K from 1, which corresponds to the s-RNN, all the way up to 10000, which corresponds to the unrestricted RNTN. As a simple way to evaluate our choice of rank-based mapping function f, we compare it to a pseudo-random variant:

$$f_{mod}(w) = rank(w) \bmod K \qquad (16)$$

We also compare results to 1) an s-RNN with $H = 150$, which has the same number of parameters as an r-RNTN with $H = 100$ and $K = 100$. 2) An m-RNN with $H = 100$ with the size of factor vectors set to 100 to match this same number of parameters. 3) An additional r-RNTN with $H = 150$ is trained to show that performance scales with H as well.

We split each sentence into 20 word subsequences and run stochastic gradient descent via backpropagation through time for 20 steps without mini-batching, only resetting the RNN's hidden state between sentences. Initial learning rate (LR) is 0.1 and halved when the ratio of the validation perplexity between successive epochs is less than 1.003, stopping training when validation improvement drops below this ratio for 5 consecutive epochs. We use Dropout (Srivastava et al., 2014) with $p = .5$ on the softmax input to reduce overfitting. Weights are drawn from $\mathcal{N}(0, .001)$; gradients are not clipped. To validate our proposed method, we also evaluate r-RNTNs using the much larger text8[1] corpus with a 90MB-5MB-5MB train-validation-test split, mapping words which appear less than 10 times to $\langle unk \rangle$ for a total vocabulary size of 37751.

Finally, we train GRUs, LSTMs, and their r-RNTN variants using the PTB corpus and parameters similar to those used by Zaremba et al. (2014). All networks use embeddings of size 650 and a single hidden layer. Targeting $K = 100$, we set $H = 244$ for the r-GRU and compare with a GRU with $H = 650$ which has the same number of parameters. The r-LSTM has $H = 254$ to match the number of parameters of an LSTM

---

[1]http://mattmahoney.net/dc/textdata.html

10

Method	H	PTB		text8		Method	H	PTB	
		# Params	Test PPL	# Params	Test PPL			# Params	Test PPL
s-RNN	100	2M	146.7	7.6M	236.4	GRU	244	9.6M	92.2
r-RNTN f	100	3M	131.2	11.4M	190.1	GRU	650	15.5M	90.3
RNTN	100	103M	128.8	388M	-	r-GRU f	244	15.5M	**87.5**
m-RNN	100	3M	164.2	11.4M	895.0	LSTM	254	10M	88.8
s-RNN	150	3M	133.7	11.4M	207.9	LSTM	650	16.4M	**84.6**
r-RNTN f	150	5.3M	**126.4**	19.8M	**171.7**	r-LSTM f	254	16.4M	87.1

Table 1: Comparison of validation and test set perplexity for r-RNTNs with f mapping ($K = 100$ for PTB, $K = 376$ for text8) versus s-RNNs and m-RNN. r-RNTNs with the same H as corresponding s-RNNs significantly increase model capacity and performance with no computational cost. The RNTN was not run on text8 due to the number of parameters required.

with $H = 650$. The training procedure is the same as above but with mini-batches of size 20, 35 timestep sequences without state resets, initial LR of 1, Dropout on all non-recurrent connections, weights drawn from $\mathcal{U}(-.05, .05)$, and gradients norm-clipped to 5.

5 Results

Results are shown in fig. 1 and table 1.

Comparing the r-RNTN to the baseline s-RNN with $H = 100$ (fig. 1), as model capacity grows with K, test set perplexity drops, showing that r-RNTN is an effective way to increase model capacity with no additional computational cost. As expected, the f mapping outperforms the baseline f_{mod} mapping at smaller K. As K increases, we see a convergence of both mappings. This may be due to matrix sharing at large K between frequent and infrequent words because of the modulus operation in eq. (16). As infrequent words are rarely observed, frequent words dominate the matrix updates and approximate having distinct matrices, as they would have with the f mapping.

It is remarkable that even with K as small as 100, the r-RNTN approaches the performance of the RNTN with a small fraction of the parameters. This reinforces our hypothesis that complex transformation modeling afforded by distinct matrices is needed for frequent words, but not so much for infrequent words which can be well represented by a shared matrix and a distinct vector embedding. As shown in table 1, with an equal number of parameters, the r-RNTN with f mapping outperforms the s-RNN with a bigger hidden layer. It appears that heuristically allocating increased model capacity as done by the f based r-RNTN is a better way to increase performance than sim-

ply increasing hidden layer size, which also incurs a computational penalty.

Although m-RNNs have been successfully employed in character-level language models with small vocabularies, they are seldom used in word-level models. The poor results shown in table 1 could explain why.[2]

For fixed hidden layer sizes, r-RNTNs yield significant improvements to s-RNNs, GRUs, and LSTMs, confirming the advantages of distinct representations.

6 Conclusion and Future Work

In this paper, we proposed restricted recurrent neural tensor networks, a model that restricts the size of recurrent neural tensor networks by mapping frequent words to distinct matrices and infrequent words to shared matrices. r-RNTNs were motivated by the need to increase RNN model capacity without increasing computational costs, while also satisfying the ideas that some words are better modeled by matrices rather than vectors (Baroni and Zamparelli, 2010; Socher et al., 2012). We achieved both goals by pruning the size of the recurrent neural tensor network described by Sutskever et al. (2011) via sensible word-to-matrix mapping. Results validated our hypothesis that frequent words benefit from richer, dedicated modeling as reflected in large perplexity improvements for low values of K.

Interestingly, results for s-RNNs and r-GRUs suggest that given the *same number of parameters*, it is possible to obtain higher performance by increasing K and reducing H. This is not the

---

[2]It should be noted that Sutskever et al. (2011) suggest m-RNNs would be better optimized using second-order gradient descent methods, whereas we employed only first-order gradients in all models we trained to make a fair comparison.

case with r-LSTMs, perhaps to due to our choice of which of the recurrence matrices to make input-specific. We will further investigate both of these phenomena in future work, experimenting with different combinations of word-specific matrices for r-GRUs and r-LSTMs (rather than only U_h^h and U_h^c), and combining our method with recent improvements to gated networks in language modeling (Jozefowicz et al., 2016; Merity et al., 2018; Melis et al., 2018) which we believe are orthogonal and hopefully complementary to our own.

Finally, we plan to compare frequency-based and additional, linguistically motivated f mappings (for example different inflections of a verb sharing a single matrix) with mappings learned via conditional computing to measure how external linguistic knowledge contrasts with knowledge automatically inferred from training data.

Acknowledgments

References

Marco Baroni and Roberto Zamparelli. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1183–1193.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. 2015. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297* .

Welin Chen, David Grangier, and Michael Auli. 2015. Strategies for training large vocabulary neural language models. *arXiv preprint arXiv:1512.04906* .

Kyunghyun Cho and Yoshua Bengio. 2014. Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning. *arXiv preprint arXiv:1406.7362* .

Kyunghyun Cho, Bart van Merrienboer, aglar Gülehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*.

Mark Davies. 2009. The 385+ million word corpus of contemporary american english (1990–2008+): Design, architecture, and linguistic insights. *International journal of corpus linguistics* 14(2):159–190.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9 8:1735–80.

Ozan Irsoy and Claire Cardie. 2014. Modeling compositionality with multiplicative recurrent neural networks. *arXiv preprint arXiv:1412.6577* .

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410* .

Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19(2):313–330.

Gbor Melis, Chris Dyer, and Phil Blunsom. 2018. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*.

T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernocký. 2011. Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH*. pages 605–608.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. 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*. pages 1045–1048.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* .

Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, pages 1201–1211.

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

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. pages 1017–1024.

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