Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 4749–4753
Marseille, 11–16 May 2020

Language Modeling with a General Second-Order RNN

Diego Maupomé and Marie-Jean Meurs
Université du Québec à Montréal - UQAM
Montreal, QC, Canada
maupome.diego@courrier.uqam.ca, meurs.marie-jean@uqam.ca

Abstract

Different Recurrent Neural Network (RNN) architectures update their state in different manners as the input sequence is processed. RNNs including a multiplicative interaction between their current state and the current input, second-order ones, show promising performance in language modeling. In this paper, we introduce a second-order RNNs that generalizes existing ones. Evaluating on the Penn Treebank dataset, we analyze how its different components affect its performance in character-lever recurrent language modeling. We perform our experiments controlling the parameter counts of models. We find that removing the first-order terms does not hinder performance. We perform further experiments comparing the effects of the relative size of the state space and the multiplicative interaction space on performance. Our expectation was that a larger states would benefit language models built on longer documents, and larger multiplicative interaction states would benefit ones built on larger input spaces. However, our results suggest that this is not the case and the optimal relative size is the same for both document tokenizations used.

Keywords: Recurrent Neural Networks, Language Modeling

1. Introduction

One of the principal challenges in computational linguistics is learning probability distributions over sequences of words or characters. Of course, the number of possible phrases grows exponentially with respect to the length. It is therefore required of natural language models to take efficient approaches to representing natural language observations.

Recurrent Neural Networks (RNNs) are an elegant solution to this problem. Indeed, by consuming sequences of tokens (e.g. words, characters) in order, RNNs dynamically construct an internal representation of the input sentence. This representation can be used for downstream predictions.

As for inference, the backpropagation algorithm can intuitively be extended to run through the time steps of a sequence in reverse order. This is known as Backpropagation Through Time (BPTT). It allows for efficient computation of the gradients needed for gradient descent. It remains, however, that the shape of an RNN will impact its ability to learn language models.

In this work, we take interest in second-order RNN architectures. We begin by introducing the simple first-order RNN as well as several examples of second-order RNNs in Section 2. We then introduce in Section 3. an architecture generalizing these second-order architectures. Section 4. presents our experiments in character-level language modeling, and their results are discussed in Section 5. Finally, Section 6. concludes this paper.

2. First and Second-Order RNNs

RNNs process sequential data by reading one token at a time and updating a real-valued vector called their *hidden state*. Let a document be a sequence $x_1, \ldots, x_T$ indexed by $t = 1, \ldots, T$.

The hidden state of the RNN is first null,

$$h_0 = 0. \tag{1}$$

Thereafter, it is computed as a function of the past hidden state as well as the input at the current time step,

$$h_t = f(h_{t-1}, x_t). \tag{2}$$

The function updating this value, f, is known as the *transition function*.

2.1. First-Order RNNs

In a first-order RNN, the transition function takes the form

$$h_t = \phi(\mathbf{U}x_t + \mathbf{W}h_{t-1} + \mathbf{b}), \tag{3}$$

where $\mathbf{U}$,$\mathbf{W}$ are matrices of parameters and $\mathbf{b}$ is a vector of parameters. The function ϕ is a squashing function such as the hyperbolic tangent.

This structure allows, in theory, for straightforward modeling of sequences of arbitrary length (Hammer, 2000).

However, simple RNNs can be difficult to train by gradient descent on natural language data. Indeed, the gradient of the hidden state over n time steps takes the following form:

$$\frac{\partial h_t}{\partial h_{t-n+1}} = \prod_{i=t-n}^{t} \operatorname{diag}(\phi'_i)\mathbf{W} \tag{4}$$

This heavy dependence on $\mathbf{W}$ results in numerical instability. Bengio et al. (1994) showed that gradients can quickly disappear (or possibly explode) over time, making it impossible to learn dependencies over long ranges, such as those found in language. This can be addressed with gating mechanisms, such as Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), as well as specialized gradient descent variants (Kanai et al., 2017), weight normalization and initialization techniques (Cooijmans et al., 2016; Talathi and Vartak, 2015).

Simple RNNs can also have trouble recovering from erroneous hidden state values. This can be attributed to the small value of $\mathbf{U}x_t$ compared to $\mathbf{W}h_{t-1}$ in certain contexts (Wu et al., 2016). More importantly, an addition between the terms is a disjunction, allowing each to overrule the other in determining the next value of the hidden state, rather than requiring a conjunction of the two (Sutskever et al., 2011). In language applications, this translates into requiring an agreement between the token being read and the history of past tokens in order to update the value of the memory vector.

2.2. Second-Order RNNs

This brings us to the notion of *second-order* RNN. As previously mentioned, a first-order transition function is akin to a disjunction between the inputs and the previous hidden state. In order to have a conjunction, a multiplicative interaction between the two is needed. This can be done by introducing a 3-way tensor, $\mathbf{A}$, such that the i-th component of the hidden state at time step t would be given by:

$$h_t^{(i)} = \phi(h_{t-1}^\top \mathbf{A}^{(i)} x_t + \mathbf{b}^{(i)}). \quad (5)$$

This is the second-order RNN as proposed by Goudreau et al. (1994). As previously mentioned, such an interaction allows both the input and the hidden state to influence the update of the former regardless of scale. Another advantage of this set-up is that there is a formal distinction between input and hidden-state. Indeed, the additive nature of the transition function of first-order RNNs makes it so that inputs and the hidden state can be seen as exchangeable terms in a sum, contrary to the case of multiplicative interaction.
A significant drawback of this first second-order formulation is the high parameter count such a tensor would entail. To remedy this, one can synthesize a rank r 3-way tensor by adding r rank 1 tensors. This is the simplification used in the multiplicative RNN (mRNN) (Sutskever et al., 2011). Its transition function uses two supplemental parameter matrices, $\mathbf{Z}, \mathbf{V}$, to make this approximation:

$$h_t = \phi(\mathbf{Z}(\mathbf{V}x_t * \mathbf{W}h_{t-1}) + \mathbf{U}x_t + \mathbf{b}). \quad (6)$$

The second-order term, $\mathbf{V}x_t * \mathbf{W}h_{t-1}$, is referred to as the *intermediate state* and denoted by m_t. The dimension of m, $|m|$ is the rank of the synthesized tensor. It can be set freely with respect to h, allowing for the multiplicative interaction to take place in a space of smaller or larger dimension than that of h.
A third example example of second-order RNN is the Multiplicative Integration RNN (MI-RNN) (Wu et al., 2016). In its simple formulation, the transition function takes the following form:

$$h_t = \phi(\mathbf{U}x_t * \mathbf{W}h_{t-1} + \mathbf{b}), \quad (7)$$

where $*$ denotes the Hadamard product. The parametrization is the same as in the first-order RNN, but the interaction between the two terms is multiplicative. This makes the RNN second-order.
The authors also offer a more general formulation, which includes two first-order terms using the same matrices of parameters as well as additional multiplicative bias vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$:

$$h_t = \phi(\boldsymbol{\alpha} * \mathbf{U}x_t * \mathbf{W}h_{t-1} + \boldsymbol{\beta}_1 * \mathbf{U}x_t + \boldsymbol{\beta}_2 * \mathbf{W}h_{t-1} + \mathbf{b}). \quad (8)$$

This allows the resulting models more expressiveness, while adding few additional parameters. This architecture, however, does not allow for the multiplicative interaction to take place in a space of arbitrary dimension.

3. A General Model

We propose here a model that generalizes existing second order models. Its parameters are divided into five matrices, $\mathbf{A}$ through $\mathbf{E}$, and a bias vector, $\mathbf{f}$. The transition function has the following form:

$$h_t = \phi(\mathbf{A}(\mathbf{B}x_t * \mathbf{C}h_{t-1}) + \mathbf{D}x_t + \mathbf{E}h_{t-1} + \mathbf{f}) \quad (9)$$

The two first order terms, $\mathbf{D}x_t$ and $\mathbf{E}h_{t-1}$, as well as the null order term, $\mathbf{f}$, have the same dimension as the hidden state, $|h|$. The second order term, $\mathbf{B}x_t * \mathbf{C}h_{t-1}$ is of free dimension, $|m|$, with $\mathbf{A}$ mapping it back to the space of the hidden state.
The mRNN can therefore be seen as the special case where $\mathbf{E}$ is null. As for MI-RNNs, their general formulation corresponds to the case where the following constraints hold:

$$\mathbf{A} = \mathrm{diag}(\boldsymbol{\alpha}) \quad (10)$$

$$\mathbf{D} = \mathrm{diag}(\boldsymbol{\beta}_1)\mathbf{U} \quad (11)$$

$$\mathbf{E} = \mathrm{diag}(\boldsymbol{\beta}_2)\mathbf{W} \quad (12)$$

To explore the properties of our proposed architecture, we conducted several experiments in recurrent language modeling on the Penn Treebank corpus (Marcus et al., 1993).

4. Language modeling experiments

We performed several exploratory experiments in recurrent language modeling on the Wall Street Journal portion of the Penn Treebank corpus. The dataset is comprised of articles in English published by the Wall Street Journal between 1987 and 1989.
We treat each line in the article as a separate document. We applied the same preprocessing for all experiments. We replaced all non-alphabetic characters by white spaces, then replaced all consecutive white spaces by a single one. Finally, we replaced all uppercase letters by their lower case counterparts. This results in a total 49,689 lines of text with an alphabet of 27 characters. We did not trim the lines in training or testing.
Models are evaluated by average cross-entropy over the test documents, in bits per character (BPC). All of the results reported in this Section are issued from 5-fold cross-validation. Inference was performed using the Adam optimizer (Kingma and Ba, 2014) on 90% of the lines in the training set (82% of the total). The remainder of the training

		Activation function	
		$tanh$	Id
$\mathbf{D}x_t$	$\mathbf{E}h_{t-1}$	BPC	BPC
✓	✓	1.14	1.22
✓	-	1.13	1.16
-	✓	1.14	1.23
-	-	1.12	1.17
first-order RNN		1.27	-
general MI-RNN		1.25	-

Table 1: Results of our first-order term ablation experiments in BPC. We include a first-order RNN and a general MI-RNN of the same parameter count for reference.

set was used as a validation set. A maximum of 25 epochs was given to all models. The training examples were put in batches of 64. The results represent the average performance on the 20% test set across the folds.

4.1. First-order term ablation experiments

We began by testing the importance of the two first-order terms in Eq.9 by ablation. Specifically, we build four recurrent language models, one for each possible combination: removing each, both or none. This indirectly tests the importance of the second-order term as well.
We allot all models the same parameter budget of 500k parameters. The size of the hidden and intermediate states are adjusted accordingly with the constraint that they be equal. The activation function (ϕ) used is the hyperbolic tangent ($tanh$).
If we examine the gradient of the hidden state over n time steps, we obtain

$$\frac{\partial h_t}{\partial h_{t-n}} = \prod_{i=t-n+1}^{t} \mathrm{diag}(\phi_i')(\mathbf{E} + \mathbf{A}\mathrm{diag}(\mathbf{B}x_i)\mathbf{C}) \quad (13)$$

The right-hand side, $\mathbf{A}\mathrm{diag}(\mathbf{B}x_t)\mathbf{C}$, depends on the input at each time step. This will allow the input to dynamically adjust $\mathbf{A}$ and $\mathbf{C}$, as argued by Wu et al. (2016). However, the hidden-to-hidden parameter matrix, $\mathbf{E}$ is ”loose”, much like in the first-order RNN, potentially allowing the gradients to explode. Note that the right-hand size is not potentiated by $\mathbf{E}$ but rather adds to it. We hypothesized this would prevent the gradients from vanishing.
To test this, we conducted the experiments again with ϕ set to the Identity (Id), which facilitates gradient explosion (Grosse, 2017). In short, our expectation was that including $\mathbf{E}h_{t-1}$ in a model would hurt performance but more so with identity activation. Results are presented in Table 1 and discussed in the following Section.

4.2. Experiments on the role of *m* and *h*

We continued with experiments comparing the impact of the dimension of $|m|$ and $|h|$ on performance. We hypothesized that, broadly speaking, a larger hidden state amounts to a larger and therefore longer memory capacity. Nonetheless, in the context of first-order RNNs, a larger hidden state is accompanied by a very significant and perhaps undesired

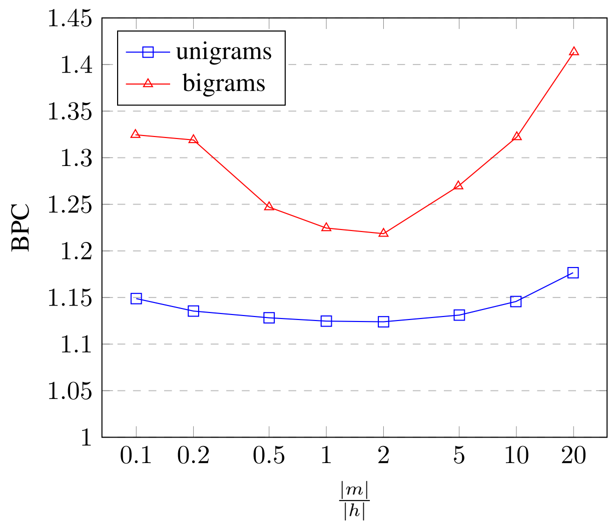

Figure 1: Test cross-entropy in BPC for equally sized models having different values of $\frac{|m|}{|h|}$ in two recurrent language modeling tasks

increase in the parameter count. This tricky trade-off between under and overfitting helps to make first-order RNNs difficult to train.
However, in the case of our second-order architecture, this is mitigated in two ways. Firstly, as our earlier experiments suggest, the first-order terms can be omitted. This significantly alleviates the parameter count increase with respect to $|h|$. Secondly, the dimension of $|m|$ being free, it can be decreased to keep the parameter count low. Furthermore, we contend that a larger $|m|$ allows for more complex transition to take place, as both the input and the hidden state value are mapped onto a larger dimension space separating their different factors of variation.
The settings of these experiments were largely similar to the previous ones except that we also train models operating on sequential pairs of characters. These pairs do not overlap and stay within word boundaries. They are also complemented by single characters. For example the word '`state`' would be tokenized as '`st`', '`at`', '`e`'. The documents are therefore shorter when tokenized in this fashion, but the input space is larger.
Given these two separate tasks – using character unigrams as for the previous experiments, and character bigrams as previously explained – we trained models on them taking with varying values of $|m|$ and $|h|$. We trained 8 models per task, with $\frac{|m|}{|h|}$ taking on values in $\{0.1, 0.2, 0.5, 1, 2, 5, 10, 20\}$ for each. Again, we allot all models a parameter budget of 500k parameters. The absolute sizes of $|m|$ and $|h|$ are adjusted accordingly.
Although the unigram task would presumably obtain better results overall, this was largely immaterial to our experiments. The models being equal in parameter count, as the dimension of h increases, that of m decreases and conversely. In both tasks, having $|h|$ or $|m|$ too small would be detrimental: too little memory capacity would impede the models from remembering the pertinent context; too simple transitions would prevent the models from exploiting

```
to land customers for their well paying stock
th toud tortomers tor the r oaal brr ng torck

but it s just one of those things that happened
tut tt s aust tne of these shangs that tavpensd
```

Figure 2: Prediction example for the best unigram model. Above are the true sequences; below are the model predictions.

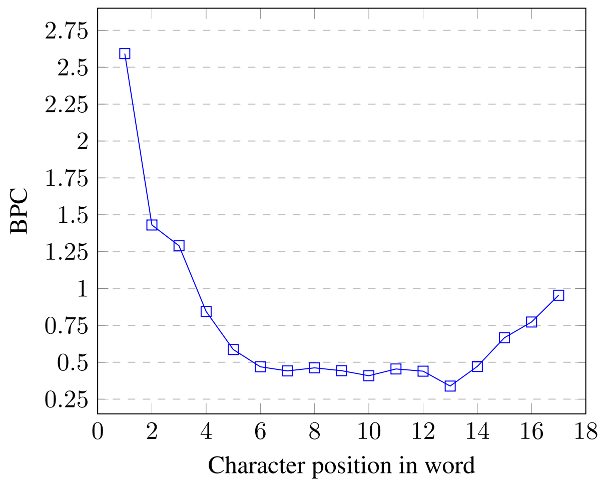

Figure 3: Average test loss of the best unigram model as a function of a character position in a word

its memory.

We therefore expected the optimal value of $\frac{|m|}{|h|}$ to lie inside our selected interval. Under our hypothesis, the unigram task would require more memory because the sequences are longer. The bigram task would require less memory capacity and more complex transitions because the input space is of larger dimension. The best bigram model would therefore have a larger $\frac{|m|}{|h|}$ than the best unigram model.

In summary, we expected the optimal value of $\frac{|m|}{|h|}$ to be larger for the character bigram task than for the character unigram task regardless of what these optimal values yielded in loss. The results we obtained are shown in Figure 1, and discussed in the following Section.

5. Discussion

The results of the first set of experiments are presented in Table 1. As expected, models free of the first-order term $\mathbf{E}h_{t-1}$ outperformed the rest. These differences are very small when using the hyperbolic tangent, whose derivative prevents gradient explosion. They become larger when using no activation.

In addition, our models outperformed our baseline first-order RNN. These results suggest the first-order terms can be entirely omitted, all things being equal.

The second set of experiments yielded surprising yet interesting results. The bigram task was more difficult, as expected. However, while we expected the optima to differ, they appear to be around the same value of 2 for both tasks. The key difference between both tasks is that the bigram task appears to be much more sensitive to different values of $\frac{|m|}{|h|}$.

It is important to note that, while bigram tokenization does reduce the length of the documents, this reduction is much smaller than the increment in the dimension of the input space. Indeed, tokenizing the documents in pairs of characters decreases the median length by about 40%, the dimension of the input space increases by a factor of 27, the size of the alphabet.

Figure 2 shows selected test observations matched to the best unigram model predictions. As one might expect, most prediction errors happen at the beginning of words. The model is however quick to adjust its prediction. These predictions are sensible given the current input character and the containing word.

In the third example of Figure 2, having read ’`happen`’, the model erroneously predicts ’`s`’, which forms a true word. Upon reading the following character ’`e`’, the model adjusts its prediction to the correct word: ’`happened`’.

Figure 3 shows this steep decline in prediction error as the model moves further along the current word. There is, however, a late increase, which is likely due to the rarity of words over 13 characters in the dataset (<2%).

6. Conclusion

We have offered a generalization of second-order RNNs, which allow, through their multiplicative interaction, more expressive transitions of their state. The results of our first set of experiments are consistent with the hypothesis that the first-order terms are largely secondary and should be omitted.

Moreover, our second set of experiments suggest that the length of tokenization, while affecting both length and ”width of input” is ultimately immaterial to the best relative sizes of m and h. This surprising result requires further exploration in determining how different aspects of corpora and RNNs determine the optimal settings for learning probability distributions over language.

Since existing second-order RNNs have been modified into gated variants, such as the multiplicative LSTM (Krause et al., 2016), the multiplicative GRU (Maupomé and Meurs, 2019) and the multiplicative integration LSTM (Wu et al., 2016), further work is required to verify how the results presented in this paper translate to gated variants.

Reproducibility

To ensure reproducibility and comparisons between systems, our source code is publicly released as an open source software in the following repository:
`https://gitlab.ikb.info.uqam.ca/diego/gsornn_lrec`.

Acknowledgment

This work is funded by the New Frontiers in Research Fund (NFRF) RELAI project. We acknowledge the support of the Social Sciences and Humanities Research Council of Canada (SSHRC), the Canadian Institutes of Health Research (CIHR) and the Natural Sciences and Engineering Research Council of Canada (NSERC).

Bibliographical References

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

Cooijmans, T., Ballas, N., Laurent, C., and Courville, A. C. (2016). Recurrent batch normalization. *CoRR*, abs/1603.09025.

Goudreau, M. W., Giles, C. L., Chakradhar, S. T., and Chen, D. (1994). First-order versus second-order single-layer recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(3):511–513, May.

Grosse, R. (2017). Lecture 15: Exploding and Vanishing Gradients, University of Toronto Computer Science.

Hammer, B. (2000). On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1-4):107–123.

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

Kanai, S., Fujiwara, Y., and Iwamura, S. (2017). Preventing gradient explosions in gated recurrent units. In I. Guyon, et al., editors, *Advances in Neural Information Processing Systems 30*, pages 435–444. Curran Associates, Inc.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Krause, B., Lu, L., Murray, I., and Renals, S. (2016). Multiplicative LSTM for Sequence Modelling. *arXiv preprint arXiv:1609.07959*.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330.

Maupomé, D. and Meurs, M. (2019). Multiplicative models for recurrent language modeling. *CoRR*, abs/1907.00455.

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

Talathi, S. S. and Vartak, A. (2015). Improving performance of recurrent neural network with relu nonlinearity. *CoRR*, abs/1511.03771.

Wu, Y., Zhang, S., Zhang, Y., Bengio, Y., and Salakhutdinov, R. R. (2016). On Multiplicative Integration with Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, pages 2856–2864.