Multiplicative Models for Recurrent Language Modeling

Diego Maupomé, Marie-Jean Meurs
maupome.diego@courrier.uqam.ca
meurs.marie-jean@uqam.ca

Université du Québec à Montréal, Montréal, QC, Canada

Abstract. Recently, there has been interest in multiplicative recurrent neural networks for language modeling. Indeed, simple Recurrent Neural Networks (RNNs) encounter difficulties recovering from past mistakes when generating sequences due to high correlation between hidden states. These challenges can be mitigated by integrating second-order terms in the hidden-state update. One such model, multiplicative Long Short-Term Memory (mLSTM) is particularly interesting in its original formulation because of the sharing of its second-order term, referred to as the intermediate state. We explore these architectural improvements by introducing new models and testing them on character-level language modeling tasks. This allows us to establish the relevance of shared parametrization in recurrent language modeling.

1 Introduction

One of the principal challenges in computational linguistics is to account for the word order of the document or utterance being processed [7]. Of course, the numbers of possible phrases grows exponentially with respect to a given phrase length, requiring an approximate approach to summarizing its content. Recurrent Neural Networks (RNNs) are such an approach, and they are used in various tasks in Natural Language Processing (NLP), such as machine translation [17], abstractive summarization [21] and question answering [12]. However, RNNs, as approximations, suffer from numerical troubles that have been identified, such as that of recovering from past errors when generating phrases. We take interest in a model that mitigates this problem, multiplicative RNNs (mRNNs), and how it has been and can be combined for new models. To evaluate these models, we use the task of *recurrent language modeling*, which consists in predicting the next token (character or word) in a document. This paper is organized as follows: RNNs and mRNNs are introduced respectively in Sections 2 and 3. Section 4 presents new and existing multiplicative models. Section 5 describes the datasets and experiments performed, as well as results obtained. Sections 6 discusses and concludes our findings.

2 Recurrent neural networks

RNNs are powerful tools of sequence modeling that can preserve the order of words or characters in a document. A document is therefore a sequence of words, $x_1, \ldots, x_T$. Given the exponential growth of possible histories with respect to the sequence length, the probability of observing a given sequence needs to be approximated. RNNs will make this approximation using the product rule,

$$P(x_1, \ldots, x_T) = P(x_1)P(x_2|x_1) \ldots P(x_T|x_1, \ldots, x_{T-1}),$$

and updating a *hidden state* at every time step. This state is first null,

$$h_0 = \mathbf{0}.$$

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

$$h_t = f(h_{t-1}, x_t),$$

known as the *transition function*. f is a learned function, often taking the form

$$h_t = \tanh(Ux_t + Wh_{t-1}).^1$$

This allows, in theory, for straightforward modeling of sequences of arbitrary length.

In practice, RNNs encounter some difficulties that need some clever engineering to be mitigated. For example, learning long-term dependencies such as those found in language is not without its share of woes arising from numerical considerations, such as the well-known vanishing gradient problem [2]. This can be addressed with gating mechanisms, such as Long Short-Term Memory network (LSTM) [10] and Gated Recurrent Unit (GRU) [3].

A problem that is more specific to generative RNNs is their difficulty recovering from past errors [8], which [16] argue arises from having hidden-state transitions that are highly correlated across possible inputs. One approach to adapting RNNs to have more input-dependent transition functions is to use the multiplicative "trick" [24]. This approximates the idea of having the input at each time synthesize a dedicated kernel of parameters dictating the transition from the previous hidden state to the next. These two approaches can be combined, as in the multiplicative LSTM (mLSTM) [16].

We begin by contending that, in making RNNs multiplicative, sharing what is known as the *intermediate state* does not significantly hinder performance when parameter counts are equal. We verify this with existing as well as new gated models on several well-known language modeling tasks.

[1] Additive biases are omitted throughout the paper for concision

3 Multiplicative RNNs

Most recurrent neural network architectures, including LSTM and GRU share the following building block:

$$\tilde{h}_t = Ux_t + Wh_{t-1}. \tag{1}$$

$\tilde{h}_t$ is the *candidate* hidden state, computed from the previous hidden state, h_{t-1}, and the current input, x_t, weighted by the parameter matrices W and U, respectively. This candidate hidden state may then be passed through gating mechanisms and non-linearities depending on the specific recurrent model.

Let us assume for simplicity that the input is a one-hot vector (one component is 1, the rest are 0 [23] [see p.45]), as it is often the case in NLP. Then, the term Ux_t is reduced to a single column of U and can therefore be thought of as an input-dependent bias in the hidden state transition. As the dependencies we wish to establish between the elements of the sequences under consideration become more distant, the term Wh_t will have to be significantly larger than this input-dependent bias, Ux_t, in order to remain unchanged across time-steps. This will mean that from one time-step to the next, the hidden-to-hidden transition will be highly correlated across possible inputs. This can be addressed by having more input-dependent hidden state transitions, making RNNs more expressive.

In order to remedy the aforementioned problem, each possible input i can be given its own matrix $W^{(i)}$ parameterizing the contribution of h_t to $\tilde{h}_t$.

$$\tilde{h}_t = Ux_t + \underbrace{(\sum_i W^{(i)} x_t^{(i)})}_{\mathbf{W}^{(x_t)}} h_{t-1}. \tag{2}$$

This is known as a tensor RNN (tRNN) [24], because all the matrices can be stacked to form a rank 3 tensor, $\mathbf{W}$. The input x_t selects the relevant slice of the tensor in the one-hot case and a weighted sum over all slices in the dense case. The resulting matrix then acts as the appropriate W.

However, such an approach is impractical because of the high parameter count such a tensor would entail. The tensor can nonetheless be approximated by factorizing it [25] as follows:

$$\mathbf{W}^{(x_t)} = V\mathrm{diag}(W_x x_t)W_h, \tag{3}$$

where W_x and W_h are weight matrices, and diag is the operator turning a vector v into a diagonal matrix where the elements of v form the main diagonal of said matrix. Replacing $\mathbf{W}^{(x_t)}$ in Equation (2) by this tensor factorization, we obtain

$$\tilde{h}_t = Ux_t + Vm_t, \tag{4}$$

where m_t is known as the *intermediate state*, given by

$$m_t = (W_x x_t) * (W_h h_{t-1}). \tag{5}$$

Here, $*$ refers to the Hadamard or element-wise product of vectors. The intermediate state is the result of having the input apply a learned filter via the new parameter kernel W to the factors of the hidden state. It should be noted that the dimensionality of m_t is free and, should it become sufficiently large, the factorization becomes as expressive as the tensor. The ensuing model is known as a mRNN [24].

4 Sharing intermediate states

While mRNNs outperform simple RNNs in character-level language modeling, they have been found wanting with respect to the popular LSTM [10]. This prompted [16] to apply the multiplicative "trick" to LSTM resulting in the mLSTM, which achieved promising results in several language modeling tasks [16].

4.1 mLSTM

Gated RNNs, such as LSTM and GRU, use *gates* to help signals move through the network. The value of these gates is computed in much the same way as the candidate hidden state, albeit with different parameters. For example, LSTM uses two different gates, i and f in updating its memory cell, c_t,

$$c_t = f_t * c_{t-1} + i_t * \tanh(\tilde{h}_t). \tag{6}$$

It uses another gate, o, in mapping c_t to the new hidden state, h_t,

$$h_t = o_t * \sigma(c_t), \tag{7}$$

where σ is the sigmoid function, squashing its input between 0 and 1. f and i are known as forget and input gates, respectively. The forget gates allows the network to ignore components of the value of the memory cell at the past state. The input gate filters out certain components of the new hidden state. Finally, the output gates separates the memory cell from the actual hidden state. The values of these gates are computed at each time step as follows:

$$i_t = \sigma(U_i x_t + W_i h_{t-1}) \tag{8}$$

$$f_t = \sigma(U_f x_t + W_f h_{t-1}) \tag{9}$$

$$o_t = \sigma(U_o x_t + W_o h_{t-1}). \tag{10}$$

Each gate has its own set of parameters to infer. If we were to replace each $W_\star$ by a tensor factorization as in mRNN, we would obtain a mLSTM model. However, in the original formulation of mLSTM, there is no factorization of each would-be $\mathbf{W}_\star$ *individually*. There is no separate intermediate state for each gate, as one would expect. Instead, a single intermediate state, m_t, is computed to replace h_{t-1} in *all* equations in the system, by Eq.5. Furthermore, each gate has its own $V_\star$ weighting m_t. Their values are computed as follows:

$$i_t = \sigma(W_i h_{t-1} + V_i m_t) \tag{11}$$

$$f_t = \sigma(W_f h_{t-1} + V_f m_t) \tag{12}$$

$$o_t = \sigma(W_o h_{t-1} + V_o m_t). \tag{13}$$

The model can therefore no longer be understood as as an approximation of the tRNN. Nonetheless, it has achieved empirical success in NLP. We therefore try to explore the empirical merits of this shared parametrization and apply them to other RNN architectures.

4.2 True mLSTM

We have presented the original mLSTM model with its shared intermediate state. If we wish to remain true to the original multiplicative model, however, we have to factorize every would-be $W_\star$ tensor separately. We have:

$$i_t = \sigma(U_i x_t + V_i m_{i,t}) \tag{14}$$

$$f_t = \sigma(U_f x_t + V_f m_{f,t}) \tag{15}$$

$$o_t = \sigma(U_o x_t + V_o m_{o,t}), \tag{16}$$

with each $m_{\star,t}$ being given by a separate set of parameters:

$$m_{\star,t} = (W_{\star,x} x_t) * (W_{\star,h} h_{t-1}). \tag{17}$$

We henceforth refer to this model as true mLSTM (tmLSTM). We sought to apply the same modifications to the GRU model, as LSTM and GRU are known to perform similarly [9, 4, 13]. That is, we build a true multiplicative GRU (tmGRU) model, as well as a multiplicative GRU (mGRU) with a shared intermediate state.

4.3 GRU

The GRU was first proposed by [3] as a lighter, simpler variant of LSTM. GRU relies on two gates, called, respectively, the *update* and *reset* gates, and no additional memory cell. These gates intervene in the computation of the hidden state as follows:

$$h_t = (1 - z_t) h_{t-1} + z_t \tanh(\tilde{h}_t), \tag{18}$$

where the candidate hidden state, $\tilde{h}_t$, is given by:

$$\tilde{h}_t = U_h x_t + W_h (r_t * h_{t-1}). \tag{19}$$

The update gate deletes specific components of the hidden state and replaces them with those of the candidate hidden state, thus updating its content. On the other hand, the reset gate allows the unit to start anew, as if it were reading the first symbol of the input sequence. They are computed much in the same way as the gates of LSTM:

$$z_t = \sigma(U_z x_t + W_z h_{t-1}), \tag{20}$$

$$r_t = \sigma(U_r x_t + W_r h_{t-1}). \tag{21}$$

4.4 True mGRU

We can now make GRU multiplicative by using the tensor factorization for z and r:

$$z_t = \sigma(U_z x_t + V_z m_{z,t}), \tag{22}$$

$$r_t = \sigma(U_r x_t) + V_r m_{r,t}, \tag{23}$$

with each $m_{\star,t}$ given by Eq. 17. There is a subtlety to computing $\tilde{h}_t$, as we need to apply the reset gate to h_{t-1}. While h_t itself is given by Eq. 4, $m_{h,t}$ is not computed the same way as in mLSTM and mRNN. Instead, it is given by:

$$m_{h,t} = (W_x x_t) * (W_h(r_t * h_{t-1})). \tag{24}$$

4.5 mGRU with shared intermediate state

Sharing an intermediate state is not as immediate for GRU. This is due to the application of r_t, which we need in computing the intermediate state that we want to share. That is, r_t and m_t would both depend on each other. We modify the role of r_t to act as a filter on m_t, rather than a reset on individual components of h_{t-1}. Note that, when all components of r_t go to zero, it amounts to having all components of h_{t-1} at zero. We have

$$z_t = \sigma(U_z x_t + V_z m_t) \tag{25}$$

and

$$r_t = \sigma(U_r x_t + V_r m_t). \tag{26}$$

$\tilde{h}_t$ is given by

$$\tilde{h}_t = U_h x_t + V_h(r_t * m_t), \tag{27}$$

with m_t the same as in mRNN and mLSTM this time, i.e. Eq.5. The final hidden state is computed the same way as in the original GRU (Eq.18).

5 Experiments in character-level language modeling

Character-level language modeling (or character prediction) consists in predicting the next character while reading a document one character at a time. It is a common benchmark for RNNs because of the heightened need for shared parametrization when compared to word-level models. We test mGRU on two well-known datasets, the Penn Treebank and Text8.

5.1 Penn Treebank

The Penn Treebank dataset [18] comes from a series of Wall Street Journal articles written in English. Following [19], sections 0-20 were used for training, 21-22 for validation and 23-24 for testing, respectively, which amounts to 5.1M, 400K and 450K characters, respectively.

Model	Parameter count	Error(BPC)
GRU [1]	3M	1.53
mRNN [19]	-	1.41
LSTM [5]	-	1.38
batch-normalized LSTM [5]	-	1.32
mLSTM [16]	-	1.27
fast-slow LSTM [20]	7.2M	1.19
mLSTM	**292K**	**1.11**
tmLSTM	**292K**	**1.09**
tmGRU	**292K**	**1.08**
mGRU	**292K**	**1.07**
larger mGRU	**2.1M**	**0.98**

Table 1: Test set error on Penn Treebank and parameter counts in character-level language modeling

The vocabulary consists of 10K lowercase words. All punctuation is removed and numbers were substituted for a single capital N. All words out of vocabulary are replaced by the token `<unk>`.

The training sequences were passed to the model in batches of 32 sequences. Following [16], we built an initial mLSTM model of 700 units. However, we set the dimensionality of the intermediate state to that of the input in order to keep the model small. We do the same for our mGRU, tmLSTM and tmGRU, changing only the size of the hidden state so that all four models have roughly the same parameter count. We trained it using the Adam optimizer [14], selecting the best model on validation over 10 epochs. We apply no regularization other than a checkpoint which keeps the best model over all epochs.

The performance of the model is evaluated using the error in bits per character (BPC), which is log_2 of perplexity. A perplexity of p means the model is as good as if it were guessing from p options at each time step, therefore, a lower perplexity (and a lower error) is better.

All models outperform previously reported results for mLSTM [16] despite lower parameter counts. This is likely due to our relatively small batch size. However, they perform fairly similarly. Encouraged by these results, we built an mGRU with both hidden and intermediate state sizes set to that of the original mLSTM (700). This version highly surpasses the previous state of the art while still having fewer parameters than previous work.

For the sake of comparison, results as well as parameter counts (where available) of our models (bold) and related approaches are presented in Table 1. mGRU and larger mGRU, our best models, achieved respectively an error of 1.07 and 0.98 BPC on the test data, setting a new state of the art for this task.

Model	Parameter count	Error (BPC)
GRU [1]	5M	1.53
mRNN [19]	-	1.54
LSTM [5]	-	1.43
mLSTM [16]	20M	1.42
mLSTM	**133K**	**1.37**
batch-normalized LSTM[5]	-	1.36
tmGRU	**133K**	**1.35**
tmLSTM	**133K**	**1.35**
mGRU	**133K**	**1.35**
large mLSTM [16]	46M	1.27
larger mGRU	**877K**	**1.21**
LSTM [15]*	45M	1.19

Table 2: Test set error on Text8 and parameter counts in character-level language modeling

5.2 Text8

The Text8 corpus [11] comprises the first 100M plain text characters in English from Wikipedia in 2006. As such, the alphabet consists of the 26 letters of the English alphabet as well as the space character. No vocabulary restrictions were put in place. As per [19], the first 90M and 5M characters were used for training and validation, respectively, with the last 5M used for testing.

Encouraged by our results on the Penn Treebank dataset, we opted to use similar configurations. However, as the data is one long sequence of characters, we divide it into sequences of 200 characters. We pass these sequences to the model in slightly larger batches of 50 to speed up computation. Again, the dimensionality of the hidden state for mLSTM is set at 450 after the original model, and that of the intermediate state is set to the size of the alphabet. The size of the hidden state is adjusted for the other three models as it was for the PTB experiments. The model is also trained using the Adam optimizer over 10 epochs.

The best model as per validation data over 10 epochs achieves 1.40 BPC on the test data, slightly surpassing an mLSTM of smaller hidden-state dimensionality (450) but larger parameter count. Our results are more modest, as are those of the original mLSTM. Once again, results do not vary greatly between models.

As with the Penn Treebank, we proceed with building an mGRU with both hidden and intermediate state sizes set to 450. This improves performance to 1.21 BPC, setting a new state of the art for this task and surpassing a large mLSTM of 1900 units from [16] despite having far fewer parameters (45M to 5M).

For the sake of comparison, results as well as parameter counts of our models and related approaches are presented in Table 2. It should be noted that some of these models employ *dynamic evaluation* [8], which fits the model further during evaluation. We refer the reader to [15]. These models are indicated by a star.

6 Conclusion

We have found that competitive results can be achieved with mRNNs using small models. We have not found significant differences in the approaches presented, despite added non-intuitive parameter-sharing constraints when controlling for model size. Our results are restricted to character-level language modeling. While the relevance of statistical language modeling and recurrent language modeling to applied NLP tasks is uncertain [6], we believe they are a fair means of comparison among RNNs architectures. Along this line of thought, previous work on mRNNs demonstrated their increased potential when compared to their regular variants [24, 16, 22]. We therefore offer other variants as well as a first investigation into their differences. We hope to have evinced the impact of increased flexibility in hidden-state transitions on RNNs sequence-modeling capabilities. Further work in this area is required to transpose these findings into applied tasks in NLP.

References

1. Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. CoRR **abs/1803.01271** (2018), `http://arxiv.org/abs/1803.01271`
2. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks **5**(2), 157–166 (1994)
3. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv preprint arXiv:1406.1078 (2014)
4. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv preprint arXiv:1412.3555 (2014)
5. Cooijmans, T., Ballas, N., Laurent, C., Courville, A.C.: Recurrent Batch Normalization. CoRR **abs/1603.09025** (2016), `http://arxiv.org/abs/1603.09025`
6. De Mulder, W., Bethard, S., Moens, M.F.: A survey on the application of recurrent neural networks to statistical language modeling. Computer Speech & Language **30**(1), 61–98 (2015)
7. Ghodsi, A., DeNero, J.: An analysis of the ability of statistical language models to capture the structural properties of language. In: Proceedings of the 9th International Natural Language Generation conference. pp. 227–231 (2016)
8. Graves, A.: Generating Sequences With Recurrent Neural Networks. CoRR **abs/1308.0850** (2013), `http://arxiv.org/abs/1308.0850`
9. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: A search space odyssey. IEEE transactions on neural networks and learning systems (2016)

10. Hochreiter, S., Schmidhuber, J.: Long Short-term Memory. Neural computation **9**(8), 1735–1780 (1997)
11. Hutter, M.: Human Knowledge Compression Contest (2006), `http://prize.hutter1.net/`
12. Iyyer, M., Boyd-Graber, J., Claudino, L., Socher, R., Daumé III, H.: A neural network for factoid question answering over paragraphs. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 633–644 (2014)
13. Jozefowicz, R., Zaremba, W., Sutskever, I.: An empirical exploration of recurrent network architectures. In: Proceedings of the 32nd International Conference on Machine Learning (ICML-15). pp. 2342–2350 (2015)
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014), `http://arxiv.org/abs/1412.6980`
15. Krause, B., Kahembwe, E., Murray, I., Renals, S.: Dynamic evaluation of neural sequence models. CoRR **abs/1709.07432** (2017), `http://arxiv.org/abs/1709.07432`
16. Krause, B., Lu, L., Murray, I., Renals, S.: Multiplicative LSTM for Sequence Modelling. arXiv preprint arXiv:1609.07959 (2016)
17. Luong, T., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. pp. 1412–1421 (2015)
18. Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a large annotated corpus of English: The Penn Treebank. Computational linguistics **19**(2), 313–330 (1993)
19. Mikolov, T., Sutskever, I., Deoras, A., Le, H.S., Kombrink, S., Cernocky, J.: Subword language modeling with neural networks. preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf) (2012)
20. Mujika, A., Meier, F., Steger, A.: Fast-slow recurrent neural networks. In: Advances in Neural Information Processing Systems. pp. 5917–5926 (2017)
21. Paulus, R., Xiong, C., Socher, R.: A deep reinforced model for abstractive summarization. CoRR **abs/1705.04304** (2017), `http://arxiv.org/abs/1705.04304`
22. Radford, A., Józefowicz, R., Sutskever, I.: Learning to Generate Reviews and Discovering Sentiment. CoRR **abs/1704.01444** (2017)
23. Socher, R., Bengio, Y., Manning, C.: Deep Learning for NLP. Tutorial at Association of Computational Logistics (ACL), 2012, and North American Chapter of the Association of Computational Linguistics (NAACL) (2013)
24. Sutskever, I., Martens, J., Hinton, G.E.: Generating text with recurrent neural networks. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11). pp. 1017–1024 (2011)
25. Taylor, G.W., Hinton, G.E.: Factored conditional restricted Boltzmann machines for modeling motion style. In: Proceedings of the 26th annual international conference on machine learning. pp. 1025–1032. ACM (2009)