Input-to-Output Gate to Improve RNN Language Models

Sho Takase Jun Suzuki Masaaki Nagata
NTT Communication Science Laboratories
{takase.sho, suzuki.jun, nagata.masaaki}@lab.ntt.co.jp

Abstract

This paper proposes a reinforcing method that refines the output layers of existing Recurrent Neural Network (RNN) language models. We refer to our proposed method as Input-to-Output Gate (IOG)[1]. IOG has an extremely simple structure, and thus, can be easily combined with any RNN language models. Our experiments on the Penn Treebank and WikiText-2 datasets demonstrate that IOG consistently boosts the performance of several different types of current topline RNN language models.

1 Introduction

A neural language model is a central technology of recently developed neural architectures in the natural language processing (NLP) field. For example, neural encoder-decoder models, which were successfully applied to various natural language generation tasks including machine translation (Sutskever et al., 2014), summarization (Rush et al., 2015), and dialogue (Wen et al., 2015), can be interpreted as conditional neural language models. Moreover, word embedding methods, such as Skip-gram (Mikolov et al., 2013) and vLBL (Mnih and Kavukcuoglu, 2013), are also originated from neural language models that aim to handle much larger vocabulary and data sizes. Thus, language modeling is a good benchmark task for investigating the general frameworks of neural methods in the NLP field.

In this paper, we address improving the performance on the language modeling task. In particular, we focus on boosting the quality of existing Recurrent Neural Network (RNN) language models. We propose the Input-to-Output Gate (IOG) method, which incorporates an additional gate function in the output layer of the selected RNN language model to refine the output. One notable characteristic of IOG is that it can be easily incorporated in any RNN language models since it is designed to be a simple structure. Our experiments on the Penn Treebank and WikiText-2 datasets demonstrate that IOG consistently boosts the performance of several different types of current topline RNN language models. In addition, IOG achieves comparable scores to the state-of-the-art on the Penn Treebank dataset and outperforms the WikiText-2 dataset.

2 RNN Language Model

This section briefly overviews the RNN language models. Hereafter, we denote a word sequence with length T, namely, $w_1, ..., w_T$ as $w_{1:T}$ for short. Formally, a typical RNN language model computes the joint probability of word sequence $w_{1:T}$ by the product of the conditional probabilities of each timestep t:

$$p(w_{1:T}) = p(w_1) \prod_{t=1}^{T-1} p(w_{t+1}|w_{1:t}). \qquad (1)$$

$p(w_1)$ is generally assumed to be 1 in this literature, that is, $p(w_1) = 1$, and thus, we can ignore the calculation of this term (See the implementation of Zaremba et al. (2014)[2], for example). To estimate the conditional probability $p(w_{t+1}|w_{1:t})$, we apply RNNs. Let V be the vocabulary size, and let $P_t \in \mathbb{R}^V$ be the probability distribution of the vocabulary at timestep t. Moreover, let D_h and D_e respectively be the dimensions of the hidden state and embedding vectors. Then, the RNN language

[1]Our implementation is publicly available at https://github.com/nttcslab-nlp/iog.

[2]https://github.com/wojzaremba/lstm

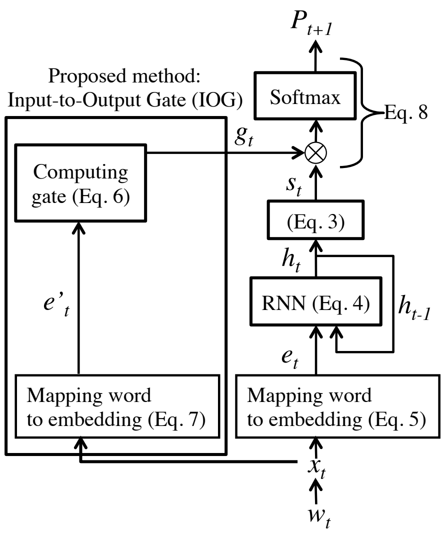

Figure 1: Overview of computing probability distribution.

models predict P_{t+1} by the following equation:

$$P_{t+1} = \text{softmax}(s_t), \tag{2}$$
$$s_t = Wh_t + b, \tag{3}$$
$$h_t = f(e_t, h_{t-1}), \tag{4}$$
$$e_t = Ex_t, \tag{5}$$

where $W \in \mathbb{R}^{V \times D_h}$ is a matrix, $b \in \mathbb{R}^V$ is a bias term, $E \in \mathbb{R}^{D_e \times V}$ is a word embedding matrix, $x_t \in \{0, 1\}^V$ is a one-hot vector representing the word at timestep t, and h_{t-1} is the hidden state at previous timestep $t-1$. h_t at timestep $t = 0$ is defined as a zero vector, that is, $h_0 = \mathbf{0}$. Let $f(\cdot)$ represent an abstract function of an RNN, which might be the Elman network (Elman, 1990), the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), the Recurrent Highway Network (RHN) (Zilly et al., 2017), or any other RNN variants.

3 Input-to-Output Gate

In this section, we describe our proposed method: Input-to-Output Gate (IOG). As illustrated in Figure 1, IOG adjusts the output of an RNN language model by the gate mechanism before computing the probability of the next word. We expect that IOG will boost the probability of the word that may occur. For example, a word followed by a preposition such as 'of' is probably a noun. Therefore, if the word at timestep t is a preposition, IOG refines the output of a language model to raise the probabilities of nouns.

Hyper-parameter	Selected value
Embedding dimension D_g	300
Dropout rate	50%
Optimization method	Adam
Initial learning rate	0.001
Learning rate decay	$1/\sqrt{\text{Epoch}}$
Max epoch	5

Table 1: Hyper-parameters in training IOG.

Formally, let x_t be a one-hot vector representing w_t, IOG calculates the gate g_t by the following equations:

$$g_t = \sigma(W_g e'_t + b_g), \tag{6}$$
$$e'_t = E_g x_t. \tag{7}$$

Here, $W_g \in \mathbb{R}^{V \times D_g}$ is a matrix, $b_g \in \mathbb{R}^V$ is a bias term, and $E_g \in \mathbb{R}^{D_g \times V}$ is a word embedding matrix[3]. Then, we compute the probability distribution of the RNN language model by applying the above gate to the Equation (2) as follows:

$$P_{t+1} = \text{softmax}(g_t \odot s_t), \tag{8}$$

where $\odot$ represents the element-wise multiplication of two vectors.

4 Experiments

4.1 Dataset

We conducted word-level prediction experiments on the Penn Treebank (PTB) (Marcus et al., 1993) and WikiText-2 (Merity et al., 2017b) datasets. The PTB dataset consists of 929k training words, 73k validation words, and 82k test words. The WikiText-2 dataset consists of 2,088k training words, 217k validation words, and 245k test words. Mikolov et al. (2010) and Merity et al. (2017b) respectively published pre-processed PTB[4] and WikiText-2[5] datasets. We used these pre-processed datasets for fair comparisons with previous studies.

4.2 Training Procedure

For the PTB dataset, we prepared a total of 5 RNN language models as our baseline models. First, we replicated LSTM with dropout and LSTM with variational inference based dropout, which we refer to as "LSTM" and "Variational LSTM", respectively. Following Zaremba et al. (2014) and Gal

[3]We prepared different embeddings from those used in an RNN language model.

[4]http://www.fit.vutbr.cz/ imikolov/rnnlm/

[5]https://einstein.ai/research/the-wikitext-long-term-dependency-language-modeling-dataset

Model	Parameters	Validation	Test
LSTM (medium) (Zaremba et al., 2014) †	20M	86.2	82.7
LSTM (medium, replication of Zaremba et al. (2014))	20M	87.1	84.0
+ IOG (proposed)	26M	84.1	81.1
LSTM (large) (Zaremba et al., 2014) †	66M	82.2	78.4
LSTM (large, replication of Zaremba et al. (2014))	66M	82.7	78.6
+ IOG (proposed)	72M	78.5	75.5
Variational LSTM (medium) (Gal and Ghahramani, 2016) †	20M	81.9 ± 0.2	79.7 ± 0.1
Variational LSTM (medium, replication of Gal and Ghahramani (2016))	20M	82.8	79.1
+ IOG (proposed)	26M	81.2	78.1
Variational LSTM (large) (Gal and Ghahramani, 2016) †	66M	77.9 ± 0.3	75.2 ± 0.2
Variational LSTM (large, replication of Gal and Ghahramani (2016))	66M	78.1	74.6
+ IOG (proposed)	72M	76.9	74.1
Variational RHN (depth 8) (Zilly et al., 2017) †	32M	71.2	68.5
Variational RHN (depth 8, replication of Zilly et al. (2017))	32M	72.1	68.9
+ IOG (proposed)	38M	69.2	66.5
Variational RHN (depth 8, replication of Zilly et al. (2017)) + WT	23M	69.2	66.3
+ IOG (proposed)	29M	67.0	64.4
Ensemble of 5 Variational RHNs	160M	66.1	63.1
+ IOG (proposed)	166M	64.7	62.0
Ensemble of 10 Variational RHNs	320M	65.2	62.3
+ IOG (proposed)	326M	64.1	61.4
Neural cache model (Grave et al., 2017) †	21M	-	72.1
Pointer Sentinel LSTM (medium) (Merity et al., 2017b) †	21M	72.4	70.9
Variational LSTM (large) + WT + AL (Inan et al., 2016) †	51M	71.1	68.5
Variational RHN (depth 10) + WT (Press and Wolf, 2017) †	24M	68.1	66.0
Neural Architecture Search with base 8 (Zoph and Le, 2017) †	32M	-	67.9
Neural Architecture Search with base 8 + WT(Zoph and Le, 2017) †	25M	-	64.0
Neural Architecture Search with base 8 + WT (Zoph and Le, 2017) †	54M	-	62.4
AWD LSTM + WT (Merity et al., 2017a) †	24M	60.0	57.3
AWD LSTM + WT (result by code of Merity et al. (2017a)[6])	24M	58.6	56.7
+ IOG (proposed)	30M	58.5	56.7
AWD LSTM + WT + cache (size = 2000) (Merity et al., 2017a) †	24M	53.9	**52.8**
AWD LSTM + WT + cache (size = 500)	24M	53.4	53.0
+ IOG (proposed)	30M	**53.3**	53.0

Table 2: Comparison between baseline models and the proposed method (represented as "+ IOG") on the Penn Treebank (PTB) dataset. † denotes results published in previous studies. The method with WT shared word embeddings (E in the Equation (5)) with the weight matrix of the final layer (W in the Equation (3)). AL denotes that the method used a previously proposed augmented loss function (Inan et al., 2016).

Model	Parameters	Validation	Test
LSTM (medium, replication of Zaremba et al. (2014))	50M	102.2	96.2
+ IOG (proposed)	70M	99.2	93.8
Variational LSTM (medium, replication of Gal and Ghahramani (2016))	50M	97.2	91.8
+ IOG (proposed)	70M	95.9	91.0
Variational LSTM (medium) + cache (size = 2000)	50M	69.6	66.1
+ IOG (proposed)	70M	69.3	65.9
Pointer Sentinel LSTM (Merity et al., 2017b) †	51M[7]	84.8	80.8
Neural cache model (size = 100) (Grave et al., 2017) †	42M	-	81.6
Neural cache model (size = 2000) (Grave et al., 2017) †	42M	-	68.9
AWD LSTM + WT (Merity et al., 2017a) †	33M	68.6	65.8
AWD LSTM + WT (result by code of Merity et al. (2017a))	33M	68.6	65.8
+ IOG (proposed)	53M	68.6	65.9
AWD LSTM + WT + cache (size = 3785) (Merity et al., 2017a) †	33M	53.8	52.0
AWD LSTM + WT + cache (size = 3785)	33M	**53.5**	**51.7**
+ IOG (proposed)	53M	53.6	**51.7**

Table 3: Comparison between baseline models and the proposed method (represented as "+ IOG") on the WikiText-2 dataset. † denotes results published in previous studies.

and Ghahramani (2016), we prepared the medium setting (2-layer LSTM with 650 dimensions for each layer), and the large setting (2-layer LSTM with 1500 dimensions for each layer) for each LSTM. We also replicated "Variational RHN" with a depth of 8 described in Zilly et al. (2017). For the WikiText-2 dataset, we prepared the medium setting standard and variational LSTMs as our baselines, which are identical as those used in Merity et al. (2017b).

After reproducing the baselines, we incorporated IOG with those models. Table 1 summarizes the hyper-parameters used for training the IOG. During training IOG, we fixed the parameters of the RNN language models to avoid over-fitting.

4.3 Results

We show the perplexities of the baselines and those combined with IOG for the PTB in Table 2, and for the WikiText-2 in Table 3. These tables, which contain both the scores reported in the previous studies and those obtained by our reproduced models, indicate that IOG reduced the perplexity. In other words, IOG boosted the performance of the baseline models. We emphasize that IOG is not restricted to a neural architecture of a language model because it improved the RHN and LSTM performances.

In addition to the comparison with the baselines, Table 2 and Table 3 contain the scores published in previous studies. Merity et al. (2017b) and Grave et al. (2017) proposed similar methods. Their methods, which are called "cache mechanism" (or 'pointer'), keep multiple hidden states at past timesteps to select words from previous sequences. Inan et al. (2016) and Press and Wolf (2017) introduced a technique that shares word embeddings with the weight matrix of the final layer (represented as 'WT' in Table 2). Inan et al. (2016) also proposed using word embeddings to augment loss function (represented as 'AL' in Table 2). Zoph and Le (2017) adopted RNNs and reinforcement learning to automatically construct a novel RNN architecture. We expect that IOG will improve these models since it can be combined with any RNN language models. In fact, Table 2 and Table 3 demonstrate that IOG enhanced the performance even when the RNN language model was combined with 'WT' or the cache mechanism.

Table 2 also shows the scores in the ensemble settings. Model ensemble techniques are widely used for further improving the performance of neural networks. In this experiment, we employed a simple ensemble technique: using the average of the output probability distributions from each model as output. We computed the probability distribution P_{t+1} on the ensemble of the M models as follows:

$$P_{t+1} = \frac{1}{M} \sum_{m=1}^{M} {}_{m}P_{t+1}, \tag{9}$$

where ${}_{m}P_{t+1}$ represents the probability distribution predicted by the m-th model. In the ensemble setting, we applied only one IOG to the multiple models. In other words, we used the same IOG for computing the probability distributions of each language model, namely, computing the Equation (8). Table 2 describes that 5 and 10 model ensemble of Variational RHNs outperformed the single model by more than 5 in perplexity. Table 2 shows that IOG reduced the perplexity of the ensemble models. Remarkably, even though the 10 Variational RHN ensemble achieved the state-of-the-art performance on the PTB dataset, IOG improved the performance by about 1 in perplexity[8].

In addition, as additional experiments, we incorporated IOG with the latest method, which was proposed after the submission deadline of IJCNLP 2017. Merity et al. (2017a) introduced various regularization and optimization techniques such as DropConnect (Wan et al., 2013) and averaged stochastic gradient descent (Polyak and Juditsky, 1992) to the LSTM language model. They called their approach AWD LSTM, which is an abbreviation of averaged stochastic gradient descent weight-dropped LSTM. Table 2 and Table 3 indicate the results on the PTB and the WikiText-2 respectively. These tables show that IOG was not effective to AWD LSTM. Perhaps, the reason is that the perplexity of AWD LSTM is close to the best performance of the simple LSTM architecture. We also note that IOG did not have any harmful effect on the language models because it maintained the performances of AWD LSTM with 'WT' and the

[6]In contrast to other comparisons, we used the following implementation by the authors: https://github.com/salesforce/awd-lstm-lm

[7]The number of parameters is different from the one described in Merity et al. (2017b). We guess that they do not consider the increase of the vocabulary size.

[8]This result was the state-of-the-art score at the submission deadline of IJCNLP 2017, i.e., July 7, 2017, but Merity et al. (2017a) surpassed it on Aug 7, 2017. We mention the effect of IOG on their method in the following paragraph.

Model	Diff	Test
Variational RHN (replicate)	-	68.9
Variational RHN + IOG (proposed)	-	**66.5**
Variational RHN + IOG with hidden	+0.8M	75.6
Variational RHN + LSTM gate	+0.7M	68.1

Table 4: Comparison among architectures for computing the output gate on the PTB dataset. The column 'Diff' shows increase of parameters from IOG (proposed).

cache mechanism. Moreover, incorporating IOG is much easier than exploring the best regularization and optimization methods for each RNN language model. Therefore, to improve the performance, we recommend combining IOG before searching for the best practice.

4.4 Discussion

Although IOG consists only of word embeddings and one weight matrix, the experimental results were surprisingly good. One might think that more sophisticated architectures can provide further improvements. To investigate this question, we examined two additional architectures to compute the output gate g_t in the Equation (6).

The first one substituted the calculation of the gate function g_t by the following g'_t:

$$g'_t = \sigma(W'_g[h_t, e'_t] + b_g), \tag{10}$$

where $W'_g \in \mathbb{R}^{V \times (D_h + D_g)}$, and $[h_t, e'_t]$ represents the concatenation of the hidden state h_t of RHN and embeddings e'_t used in IOG. We refer to this architecture as "+ IOG with hidden".

The second one similarly substituted g_t by the following g''_t:

$$g''_t = \sigma(W_g h'_t + b_g), \tag{11}$$

$$h'_t = f'(e'_t, h'_{t-1}), \tag{12}$$

where $f'(\cdot)$ is the 1-layer LSTM in our experiments. We set the dimension of the LSTM hidden state to 300, that is, $D_g = 300$, and the other hyperparameters remained as described in Section 4.2. We refer to the second one as "+ LSTM gate".

Table 4 shows the results of the above two architectures on the PTB dataset. IOG clearly outperformed the other more sophisticated architectures. This fact suggests that (1) incorporating additional architectures does not always improve the performance, and (2) not always become better even if it is a sophisticated architecture. We need to carefully design an architecture that can provide complementary (or orthogonal) information to the baseline RNNs.

Input word	Top 5 weighted words
of	security, columbia, steel, irs, thrift
in	columbia, ford, order, labor, east
go	after, through, back, on, ahead
attention	was, than, ⟨eos⟩, from, to
whether	to, she, estimates, i, ual

Table 5: Top 5 weighted words for each input word on the PTB experiment.

In addition, to investigate the mechanism of IOG, we selected particular words, and listed the top 5 weighted words given each selected word as input in Table 5[9]. IOG gave high weights to nouns when the input word was a preposition: 'of' and 'in'. Moreover, IOG encouraged outputting phrasal verbs such as "go after". These observations generally match human intuition.

5 Conclusion

We proposed Input-to-Output Gate (IOG), which refines the output of an RNN language model by the gate mechanism. IOG can be incorporated in any RNN language models due to its simple structure. In fact, our experimental results demonstrated that IOG improved the performance of several different settings of RNN language models. Furthermore, the experimental results indicate that IOG can be used with other techniques such as ensemble.

Acknowledgments

We thank the anonymous reviewers for their helpful comments.

References

Jeffrey L Elman. 1990. Finding Structure in Time. *Cognitive science* 14(2):179–211.

Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 29 (NIPS 2016).*

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2017. Improving Neural Language Models with a Continuous Cache. In *5th International Conference on Learning Representations (ICLR 2017).*

[9]In this exploration, we excluded words occurring fewer than 100 times in the corpus to remove noise.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9(8):1735–1780.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. In *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017)*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19(2):313–330.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017a. Regularizing and Optimizing LSTM Language Models. *arXiv preprint arXiv:1708.02182* .

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017b. Pointer Sentinel Mixture Models. In *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017)*.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent Neural Network based Language Model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*. pages 1045–1048.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 3111–3119.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning Word Embeddings Efficiently with Noise-Contrastive Estimation. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 2265–2273.

Boris T Polyak and Anatoli B Juditsky. 1992. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization* 30(4):838–855.

Ofir Press and Lior Wolf. 2017. Using the Output Embedding to Improve Language Models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2017)*. pages 157–163.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A Neural Attention Model for Abstractive Sentence Summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*. pages 379–389.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*. pages 3104–3112.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. 2013. Regularization of Neural Networks using DropConnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*. pages 1058–1066.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*. pages 1711–1721.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent Neural Network Regularization. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014)*.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2017. Recurrent Highway Networks. *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)* pages 4189–4198.

Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017)*.