Bayesian Compression for Natural Language Processing

Nadezhda Chirkova[1*], **Ekaterina Lobacheva**[1*], **Dmitry Vetrov**[1,2]
[1]Samsung-HSE Laboratory, National Research University Higher School of Economics
[2]Samsung AI Center
Moscow, Russia
{nchirkova,elobacheva,dvetrov}@hse.ru

Abstract

In natural language processing, a lot of the tasks are successfully solved with recurrent neural networks, but such models have a huge number of parameters. The majority of these parameters are often concentrated in the embedding layer, which size grows proportionally to the vocabulary length. We propose a Bayesian sparsification technique for RNNs which allows compressing the RNN dozens or hundreds of times without time-consuming hyperparameters tuning. We also generalize the model for vocabulary sparsification to filter out unnecessary words and compress the RNN even further. We show that the choice of the kept words is interpretable.

1 Introduction

Recurrent neural networks (RNNs) are among the most powerful models for natural language processing, speech recognition, question-answering systems (Chan et al., 2016; Ha et al., 2017; Wu et al., 2016; Ren et al., 2015). For complex tasks such as machine translation (Wu et al., 2016) modern RNN architectures incorporate a huge number of parameters. To use these models on portable devices with limited memory the model compression is desired.

There are a lot of RNNs compression methods based on specific weight matrix representations (Tjandra et al., 2017; Le et al., 2015) or sparsification (Narang et al., 2017; Wen et al., 2018). In this paper we focus on RNNs compression via sparsification. One way to sparsify RNN is pruning where the weights with a small absolute value are eliminated from the model. Such methods are heuristic and require time-consuming hyperparameters tuning. There is another group of sparsification techniques based on Bayesian approach. Molchanov et al. (2017) describe a model called SparseVD in which parameters controlling sparsity are tuned automatically during neural network training. However, this technique was not previously investigated for RNNs. In this paper, we apply Sparse VD to RNNs taking into account the specifics of recurrent network structure (Section 3.2). More precisely, we use the insight about using the same sample of weights for all timesteps in the sequence (Gal and Ghahramani, 2016; Fortunato et al., 2017). This modification makes local reparametrization trick (Kingma et al., 2015; Molchanov et al., 2017) not applicable and changes SparseVD training procedure.

In natural language processing tasks the majority of weights in RNNs are often concentrated in the first layer that is connected to the vocabulary, for example in embedding layer. However, for some tasks the most of the words are unnecessary for accurate predictions. In our model we introduce multiplicative weights for the words to perform vocabulary sparsification (Section 3.3). These multiplicative weights are zeroing out during training causing filtering corresponding unnecessary words out of the model. It allows to boost RNN sparsification level even further.

To sum up, our contributions are as follows: (i) we adapt SparseVD to RNNs explaining the specifics of the resulting model and (ii) we generalize this model by introducing multiplicative weights for words to purposefully sparsify the vocabulary. Our results show that Sparse Variational Dropout leads to a very high level of sparsity in recurrent models without a significant quality drop. Models with additional vocabulary sparsification boost compression rate on text classification tasks but do not help that much on language modeling tasks. In classification tasks the vocabulary is compressed dozens of times, and the choice of words is interpretable.

[*] Equal contribution.

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 2910–2915
Brussels, Belgium, October 31 - November 4, 2018. ©2018 Association for Computational Linguistics

2 Related work

Reducing RNN size is an important and rapidly developing area of research. There are three research directions: approximation of weight matries (Tjandra et al., 2017; Le et al., 2015), reducing the precision of the weights (Hubara et al., 2016) and sparsification of the weight matrices (Narang et al., 2017; Wen et al., 2018). We focus on the last one. The most popular approach here is pruning: the weights of the RNN are cut off on some threshold. Narang et al. (2017) choose threshold using several hyperparameters that control the frequency, the rate and the duration of the weights eliminating. Wen et al. (2018) propose to prune the weights in LSTM by groups corresponding to each neuron, this allows to accelerate forward pass through the network.

Another group of sparsification methods relies on Bayesian neural networks (Molchanov et al., 2017; Neklyudov et al., 2017; Louizos et al., 2017). In Bayesian NNs the weights are treated as random variables, and our desire about sparse weights is expressed in a prior distribution over them. During training, the prior distribution is transformed into the posterior distribution over the weights, used to make predictions on testing phase. Neklyudov et al. (2017) and Louizos et al. (2017) also introduce group Bayesian sparsification techniques that allow to eliminate neurons from the model.

The main advantage of the Bayesian sparsification techniques is that they have a small number of hyperparameters compared to pruning-based methods. Also, they lead to a higher sparsity level (Molchanov et al., 2017; Neklyudov et al., 2017; Louizos et al., 2017).

There are several works on Bayesian recurrent neural networks (Gal and Ghahramani, 2016; Fortunato et al., 2017), but these methods are hard to extend to achieve sparsification. We apply sparse variational dropout to RNNs taking into account its recurrent specifics, including some insights highlighted by Gal and Ghahramani (2016), Fortunato et al. (2017).

3 Proposed method

3.1 Notations

In the rest of the paper $x = [x_0, \dots, x_T]$ is an input sequence, y is a true output and $\hat{y}$ is an output predicted by the RNN (y and $\hat{y}$ may be single vectors, sequences, etc.), X, Y denotes a training set $\{(x^1, y^1), \dots, (x^N, y^N)\}$. All weights of the RNN except biases are denoted by ω, while a single weight (an element of any weight matrix) is denoted by w_{ij}. Note that we detach biases and denote them by B because we do not sparsify them.

For definiteness, we will illustrate our model on an example architecture for the language modeling task, where $y = [x_1, \dots, x_T]$:

$$\begin{aligned}&\text{embedding}: \tilde{x}_t = w^e_{x_t};\\ &\text{recurrent}: h_{t+1} = \sigma(W^h h_t + W^x \tilde{x}_{t+1} + b^r);\\ &\text{fully-connected}: \hat{y}_t = \text{softmax}(W^d h_t + b^d).\end{aligned}$$

In this example $\omega = \{W^e, W^x, W^h, W^d\}$, $B = \{b^r, b^d\}$. However, the model may be directly applied to any recurrent architecture.

3.2 Sparse variational dropout for RNNs

Following Kingma et al. (2015), Molchanov et al. (2017), we put a fully-factorized log-uniform prior over the weights:

$$p(\omega) = \prod_{w_{ij} \in \omega} p(w_{ij}), \quad p(w_{ij}) \propto \frac{1}{|w_{ij}|}$$

and approximate the posterior with a fully factorized normal distribution:

$$q(w|\theta, \sigma) = \prod_{w_{ij} \in \omega} \mathcal{N}\big(w_{ij}|\theta_{ij}, \sigma^2_{ij}\big).$$

The task of posterior approximation $\min_{\theta,\sigma,B} KL(q(\omega|\theta,\sigma)||p(\omega|X,Y,B))$ is equivalent to variational lower bound optimization (Molchanov et al., 2017):

$$-\sum_{i=1}^{N} \int q(\omega|\theta,\sigma) \log p(y^i|x^i_0, \dots, x^i_T, \omega, B) d\omega + \\ + \sum_{w_{ij} \in \omega} KL(q(w_{ij}|\theta_{ij}, \sigma_{ij})||p(w_{ij})) \to \min_{\theta,\sigma,B}. \tag{1}$$

Here the first term, a task-specific loss, is approximated with one sample from $q(\omega|\theta,\sigma)$. The second term is a regularizer that moves posterior closer to prior and induces sparsity. This regularizer can be very closely approximated analytically (Molchanov et al., 2017):

$$KL(q(w_{ij}|\theta_{ij}, \sigma_{ij})||p(w_{ij})) \approx k\left(\frac{\sigma^2_{ij}}{\theta^2_{ij}}\right), \tag{2}$$

$$k(\alpha) \approx 0.64\sigma(1.87 + 1.49\log\alpha) - \frac{1}{2}\log\Big(1 + \frac{1}{\alpha}\Big).$$

To make integral estimation unbiased, sampling from the posterior is performed with the use of reparametrization trick (Kingma and Welling, 2014):

$$w_{ij} = \theta_{ij} + \sigma_{ij}\epsilon_{ij}, \quad \epsilon_{ij} \sim \mathcal{N}(\epsilon_{ij}|0, 1) \qquad (3)$$

The important difference of RNNs compared to feed-forward networks consists in sharing the same weight variable between different timesteps. Thus, we should use the same sample of weights for each timestep t while computing the likelihood $p(y^i|x_0^i, \dots, x_T^i, \omega, B)$ (Gal and Ghahramani, 2016; Fortunato et al., 2017).

Kingma et al. (2015), Molchanov et al. (2017) also use local reparametrization trick (LRT) that is sampling preactivation instead of individual weights. For example,

$$(W^x x_t)_i = \sum_j \theta_{ij}^x x_{tj} + \epsilon_i \sum_j (\sigma_{ij}^x)^2 x_{tj}^2.$$

Tied weight sampling makes LRT not applicable to weight matrices that are used in more than one timestep in the RNN.

For the hidden-to-hidden matrix W^h the linear combination $(W^h h_t)$ is not normally distributed because h_t depends on W^h from the previous timestep. As a result, the rule about the sum of independent normal distributions with constant coefficients is not applicable. In practice, network with LRT on hidden-to-hidden weights cannot be trained properly.

For the input-to-hidden matrix W^x the linear combination $(W^x x_t)$ is normally distributed. However, sampling the same W^x for all timesteps and sampling the same noise ϵ_i for preactivations for all timesteps are not equivalent. The same sample of W^x corresponds to different samples of noise ϵ_i at different timesteps because of the different x_t. Hence theoretically LRT is not applicable here. In practice, networks with LRT on input-to-hidden weights may give the same results and in some experiments, they even converge a little bit faster.

Since the training procedure is effective only with 2D noise tensor, we propose to sample the noise on the weights per mini-batch, not per individual object.

To sum up, the training procedure is as follows. To perform forward pass for a mini-batch, we firstly sample all weights ω following (3) and then apply RNN as usual. Then the gradients of (1) are computed w.r.t $\theta, \log\sigma, B$.

During the testing stage, we use the mean weights θ (Molchanov et al., 2017). Regularizer (2) causes the majority of θ components approach 0, and the weights are sparsified. More precisely, we eliminate weights with low signal-to-noise ratio $\frac{\theta_{ij}^2}{\sigma_{ij}^2} < \tau$ (Molchanov et al., 2017).

3.3 Multiplicative weights for vocabulary sparsification

One of the advantages of Bayesian sparsification is an easy generalization for the sparsification of any groups of the weights that doesn't complicate the training procedure (Louizos et al., 2017). To do so, one should introduce shared multiplicative weight per each group, and elimination of this multiplicative weight will mean the elimination of the corresponding group. In our work we utilize this approach to achieve vocabulary sparsification.

Precisely, we introduce multiplicative probabilistic weights $z \in \mathbb{R}^V$ for words in the vocabulary (here V is the size of the vocabulary). The forward pass with z looks as follows:

1. sample vector z^i from the current approximation of the posterior for each input sequence x^i from the mini-batch;
2. multiply each one-hot encoded token x_t^i from the sequence x^i by z^i (here both x_t^i and z^i are V-dimensional);
3. continue the forward pass as usual.

We work with z in the same way as with other weights W: we use a log-uniform prior and approximate the posterior with a fully-factorized normal distribution with trainable mean and variance. However, since z is a one-dimensional vector, we can sample it individually for each object in a mini-batch to reduce the variance of the gradients. After training, we prune elements of z with a low signal-to-noise ratio and subsequently, we do not use the corresponding words from the vocabulary and drop columns of weights from the embedding or input-to-hidden weight matrices.

4 Experiments

We perform experiments with LSTM architecture on two types of problems: text classification and language modeling. Three models are compared here: baseline model without any regularization, SparseVD model and SparseVD model with multiplicative weights for vocabulary sparsification (SparseVD-Voc).

To measure the sparsity level of our models we calculate the compression rate of individual weights as follows: $|w|/|w \neq 0|$. The sparsification of weights may lead not only to the compression but also to the acceleration of RNNs through group sparsity. Hence, we report the number of remaining neurons in all layers: input (vocabulary), embedding and recurrent. To compute this number for vocabulary layer in SparseVD-Voc we use introduced variables z_v. For all other layers in SparseVD and SparseVD-Voc, we drop a neuron if all weights connected to this neuron are eliminated.

We optimize our networks using Adam (Kingma and Ba, 2015). Baseline networks overfit for all our tasks, therefore, we present results for them with early stopping. For all weights that we sparsify, we initialize $\log \sigma$ with -3. We eliminate weights with signal-to-noise ratio less then $\tau = 0.05$. More details about experiment setup are presented in Appendix A.

4.1 Text Classification

We evaluated our approach on two standard datasets for text classification: IMDb dataset (Maas et al., 2011) for binary classification and AGNews dataset (Zhang et al., 2015) for four-class classification. We set aside 15% and 5% of training data for validation purposes respectively. For both datasets, we use the vocabulary of 20,000 most frequent words.

We use networks with one embedding layer of 300 units, one LSTM layer of 128 / 512 hidden units for IMDb / AGNews, and finally, a fully connected layer applied to the last output of the LSTM. Embedding layer is initialized with word2vec (Mikolov et al., 2013) / GloVe (Pennington et al., 2014) and SparseVD and SparseVD-Voc models are trained for 800 / 150 epochs on IMDb / AGNews.

The results are shown in Table 1. SparseVD leads to a very high compression rate without a significant quality drop. SparseVD-Voc boosts the compression rate even further while still preserving the accuracy. Such high compression rates are achieved mostly because of the sparsification of the vocabulary: to classify texts we need to read only some important words from them. The remaining words in our models are mostly interpretable for the task (see Appendix B for the list of remaining words for IMBb). Figure 1 shows the only kept embedding component for remaining words on IMDb. This component reflects the sentiment score of the words.

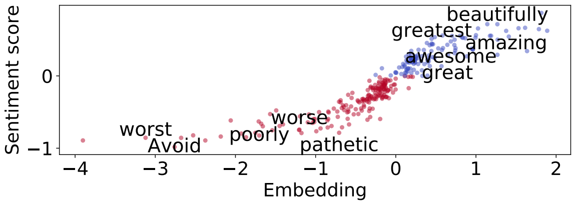

Figure 1: IMDB: remained embedding component vs sentiment score ((#pos. - #neg.) / #all texts with the word).

4.2 Language Modeling

We evaluate our models on the task of character-level and word-level language modeling on the Penn Treebank corpus (Marcus et al., 1993) according to the train/valid/test partition of Mikolov et al. (2011). The dataset has a vocabulary of 50 characters or 10,000 words.

To solve character / word-level tasks we use networks with one LSTM layer of 1000 / 256 hidden units and fully-connected layer with softmax activation to predict next character or word. We train SparseVD and SparseVD-Voc models for 250 / 150 epochs on character-level / word-level tasks.

The results are shown in Table 2. To obtain these results we employ LRT on the last fully-connected layer. In our experiments with language modeling LRT on the last layer accelerate the training without harming the final result. Here we do not get such extreme compression rates as in the previous experiment but still, we are able to compress the models several times while achieving better quality w.r.t. the baseline because of the regularization effect of SparseVD. Vocabulary is not sparsified in the character-level task because there are only 50 characters and all of them matter. In the word-level task more than a half of the words are dropped. However, since in language modeling almost all words are important, the sparsification of the vocabulary makes the task more difficult to the network and leads to the drop in quality and the overall compression (network needs more difficult dynamic in the recurrent layer).

Task	Method	Accuracy %	Compression	Vocabulary	Neurons $\tilde{x}$ - h
IMDb	Original	84.1	1x	20000	$300 - 128$
	SparseVD	**85.1**	1135x	4611	$16 - 17$
	SparseVD-Voc	83.6	**12985x**	**292**	$\mathbf{1 - 8}$
AGNews	Original	**90.6**	1x	20000	$300 - 512$
	SparseVD	88.8	322x	5727	$179 - 56$
	SparseVD-Voc	89.2	**469x**	**2444**	$\mathbf{127 - 32}$

Table 1: Results on text classification tasks. Compression is equal to $|w|/|w \neq 0|$. In last two columns number of remaining neurons in the input, embedding and recurrent layers are reported.

Task	Method	Valid	Test	Compression	Vocabulary	Neurons h
Char PTB Bits-per-char	Original	1.498	1.454	1x	50	1000
	SparseVD	1.472	1.429	**4.2x**	50	**431**
	SparseVD-Voc	**1.4584**	**1.4165**	3.53x	**48**	510
Word PTB Perplexity	Original	135.6	129.5	1x	10000	256
	SparseVD	**115.0**	**109.0**	**14.0x**	9985	**153**
	SparseVD-Voc	126.3	120.6	11.1x	**4353**	207

Table 2: Results on language modeling tasks. Compression is equal to $|w|/|w \neq 0|$. In last two columns number of remaining neurons in input and recurrent layers are reported.

Acknowledgments

Results on SparseVD for RNNs shown in Section 3.2 have been supported by Russian Science Foundation (grant 17-71-20072). Results on multiplicative weights for vocabulary sparsification shown in Section 3.3 have been supported by Samsung Research, Samsung Electronics.

References

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. 2016. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Meire Fortunato, Charles Blundell, and Oriol Vinyals. 2017. Bayesian recurrent neural networks. *Computing Research Repository*, arXiv:1704.02798.

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)*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. Proceedings of Machine Learning Research.

David Ha, Andrew Dai, and Quoc V. Le. 2017. Hypernetworks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Quantized neural networks: Training neural networks with low precision weights and activations. *Computing Research Repository*, arXiv:1609.07061.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference for Learning Representations (ICLR)*.

Diederik P Kingma, Tim Salimans, and Max Welling. 2015. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems 28*, pages 2575–2583.

Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes. In *Proceedings of the International Conference for Learning Representations (ICLR)*.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. 2015. A simple way to initialize recurrent networks of rectified linear units. *Computing Research Repository*, arXiv:1504.00941.

Christos Louizos, Karen Ullrich, and Max Welling. 2017. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems 30*, pages 3288–3298.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of* *the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 142–150, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330.

T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur. 2011. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531.

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*, pages 3111–3119.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. 2017. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*.

Sharan Narang, Gregory F. Diamos, Shubho Sengupta, and Erich Elsen. 2017. Exploring sparsity in recurrent neural networks. In *Proceedings of the International Conference for Learning Representations (ICLR)*.

Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. 2017. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems 30*, pages 6778–6787.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, volume 14, pages 1532–1543.

Mengye Ren, Ryan Kiros, and Richard S. Zemel. 2015. Exploring models and data for image question answering. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2953–2961.

Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. 2017. Compressing recurrent neural network with tensor train. *Computing Research Repository*, arXiv:1705.08052.

Wei Wen, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. 2018. Learning intrinsic sparse structures within long short-term memory. In *International Conference on Learning Representations*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *Computing Research Repository*, arXiv:1609.08144.

X. Zhang, J. Zhao, and Y. LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems (NIPS)*.