Structured Sparsification of Gated Recurrent Neural Networks

Ekaterina Lobacheva,[1*] **Nadezhda Chirkova,**[1*] **Alexander Markovich,**[2] **Dmitry Vetrov**[1,3]
[1]Samsung-HSE Laboratory, National Research University Higher School of Economics
[2]National Research University Higher School of Economics
[3]Samsung AI Center Moscow Moscow, Russia
{elobacheva, nchirkova, dvetrov}@hse.ru, amarkovich@edu.hse.ru

Abstract

One of the most popular approaches for neural network compression is sparsification — learning sparse weight matrices. In structured sparsification, weights are set to zero by groups corresponding to structure units, e. g. neurons. We further develop the structured sparsification approach for the gated recurrent neural networks, e. g. Long Short-Term Memory (LSTM). Specifically, in addition to the sparsification of individual weights and neurons, we propose sparsifying the preactivations of gates. This makes some gates constant and simplifies an LSTM structure. We test our approach on the text classification and language modeling tasks. Our method improves the neuron-wise compression of the model in most of the tasks. We also observe that the resulting structure of gate sparsity depends on the task and connect the learned structures to the specifics of the particular tasks.

Introduction

Recurrent neural networks (RNNs) yield high-quality results in many applications (Chan et al. 2016; Ha, Dai, and Le 2017; Ren, Kiros, and Zemel 2015; Wu et al. 2016) but often are memory- and time-consuming due to a large number of parameters. There are a lot of memory-limited applications where RNN compression is desired, for example, programs running on smartphones. In many practical problems, RNNs can be compressed orders of times with only a slight quality drop or even with quality improvement due to the regularization effect (Chirkova, Lobacheva, and Vetrov 2018; Narang et al. 2017; Wen et al. 2018).

Most of the methods for RNN compression can be divided into three groups: based on matrix factorization (Jose, Cissé, and Fleuret 2018; Tjandra, Sakti, and Nakamura 2017; Prabhavalkar et al. 2016), quantization (Liu, Cao, and Yu 2018; Wang et al. 2018), or sparsification (Narang et al. 2017; Chirkova, Lobacheva, and Vetrov 2018). In this paper, we focus on sparsification, i. e. setting a lot of weights to zero. In addition to the individual weight sparsification, modern sparsification approaches perform group sparsification, which is also called structured sparsification (Wen et al. 2018). Group sparsification implies removing weights by groups corresponding to structure units (neurons in RNNs).

[*]Equal contribution

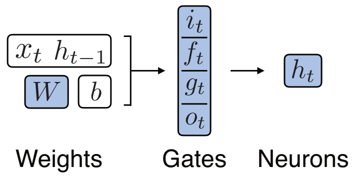

Figure 1: Proposed sparsification scheme for LSTM with three levels of sparsity: weights, gates, and neurons (shown in blue).

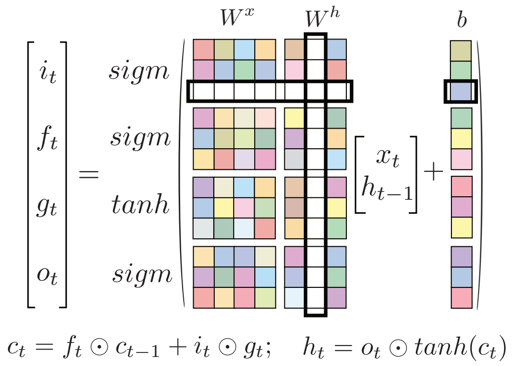
W^x
W^h
b
i_t
f_t
g_t
o_t
$sigm$
$sigm$
$tanh$
$sigm$
x_t
h_{t-1}
$c_t = f_t \odot c_{t-1} + i_t \odot g_t; \quad h_t = o_t \odot tanh(c_t)$

Figure 2: Proposed sparsification scheme for LSTM: neuron weight group and gate weight group. White cells represent zero weights. Zero row along with non-zero bias leads to the constant corresponding gate. Zero column results in removing the corresponding neuron.

In addition to compression, group sparsification also accelerates the testing stage.

Altogether, existing approaches perform RNN sparsification, either at the level of individual weights (Chirkova, Lobacheva, and Vetrov 2018; Narang et al. 2017; See, Luong, and Manning 2016), or at the level of hidden neurons (Wen et al. 2018). However, most of the modern recurrent architectures (e. g. LSTM (Hochreiter and Schmidhuber 1997), or Gated Recurrent Units (GRU) (Cho et al. 2014)), have a gated structure: to compute the next hidden state, one firstly computes gate values and then updates the hidden state based on the gates. In this paper, we propose to add an intermediate level of sparsification between weights and neurons — gates (see fig. 1). Removing weights by groups corresponding to gates makes the values of some gates constant, independent of the inputs, and equal to the activation function of the bias (see fig. 2). As a result, the LSTM/GRU structure is simplified. With this intermediate level introduced, we obtain a three-level hierarchy of sparsification: weights — gates — neurons. The sparsification of individual weights helps to sparsify the preactivations of gates (make their values constant), and the latter helps to sparsify hidden neurons (remove them from the model).

The proposed idea can be implemented for any gated architecture and in any sparsification framework. We implement the idea for the most common architecture LSTM in two sparsification frameworks: pruning (Wen et al. 2018) and Bayesian sparsification (Chirkova, Lobacheva, and Vetrov 2018). The introduced gate sparsification improves the neuron-wise compression of the gated RNNs in most cases. Moreover, we observe that the resulting gate structures, which gates are constant and which are not, vary for different natural language processing tasks. We analyze typical resulting gate structures and connect them to the specifics of the particular tasks.

Related work

For neural network sparsification, two main groups of approaches are pruning and Bayesian sparsification. In pruning (Han, Mao, and Dally 2016; Baoyuan Liu et al. 2015), ℓ_1-regularized (Tibshirani 1996) weights with absolute values less than a predefined threshold are set to zero. To achieve group sparsification (Scardapane et al. 2017; Wen et al. 2016), for each neuron, a group of all weights associated with this neuron is defined, and group Lasso (Yuan and Lin 2006) is applied to the resulting groups. If all the weights associated with a neuron are set to zero, this neuron is removed from the model. Approaches for pruning differ in the pruning schedule (when to start pruning, when to change threshold, etc.), threshold choice (constant, based on the portion of weights/groups we want to set to zero, etc.), and regularization. For RNNs, See, Luong, and Manning; Narang et al. (2016; 2017) propose unstructured sparsification method, and Wen et al. (2018) propose removing weights by groups corresponding to the hidden neurons in LSTM. Narang, Undersander, and Diamos (2018) remove blocks of the RNN weight matrices.

Bayesian sparsification techniques (Molchanov, Ashukha, and Vetrov 2017; Louizos, Welling, and Kingma 2018) treat the weights of the network as random variables and approximate the posterior distribution over the weights given a sparsity-inducing prior distribution. Louizos, Ullrich, and Welling; Neklyudov et al. (2017; 2017) perform structured sparsification for fully-connected and convolutional networks. To do it, they modify the computational graph, and multiply the output of each neuron by a learnable (and sparsifiable) group weight. If a group weight is set to zero, the corresponding neuron is removed from the model. Chirkova, Lobacheva, and Vetrov (2018) adapt Bayesian sparsification to RNNs taking into account the recurrent specifics.

Our work focuses on a specific definition of weight groups for the gated RNNs and can be implemented in any structured sparsification framework.

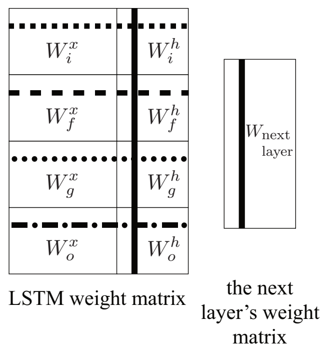

Figure 3: Proposed groups of weights in the LSTM layer. Different groups are shown using different line types: four groups corresponding to the gates (dotted horizontal lines), and one group corresponding to a neuron (solid vertical lines).

Proposed method

Main idea

We describe our approach for LSTM (Hochreiter and Schmidhuber 1997), however, it can be straightforwardly applied to other gated architectures. The LSTM cell is composed of input, forget and output gates (i, f, o), and information flow g (which we also call gate for brevity). In LSTM, the computation of the next hidden state h_t is performed as follows:

$$\begin{aligned} i_t &= sigm(W_i^x x_t + W_i^h h_{t-1} + b_i) \\ f_t &= sigm(W_f^x x_t + W_f^h h_{t-1} + b_f) \\ g_t &= tanh(W_g^x x_t + W_g^h h_{t-1} + b_g) \\ o_t &= sigm(W_o^x x_t + W_o^h h_{t-1} + b_o) \\ c_t = f_t \odot c_{t-1} + i_t \odot g_t & \qquad h_t = o_t \odot tanh(c_t) \end{aligned} \tag{1}$$

The computation of the gates can be seen as a fully-connected layer with input $[x_t, h_{t-1}]$, weight matrix, composed of all W in (1), and biases $b = [b_i, b_f, b_g, b_o]$ (see fig. 2).

Our idea is to take into account the gated structure when performing the sparsification of the gated RNNs. To do it, in addition to the sparsification of individual weights and hidden neurons, we add an intermediate level of sparsification — gates (see fig. 1). We do not sparsify biases because they do not take up much memory, compared to the weight matrices.

The introduction of the gate level means removing weights by groups corresponding to gates. These groups are represented by the rows of the LSTM weight matrix W (see dotted horizontal lines in fig. 3). For example, if we zero out the k-th row of matrices W_f^x and W_f^h, there are no ingoing connections to the k-th forget gate, so the gate becomes constant, independent of x_t and h_{t-1} and equal to $sigm(b_{f,k})$. As a result, we do not need to compute the k-th forget gate on the forward pass and can use a precomputed value. We can construct a mask (whether the gate is constant or not) and use this mask to insert constant values into the gate vectors i, f, g, o. This reduces the amount of computations on the forward pass.

At the neuron level, we define weight groups corresponding to the hidden neurons in LSTM. If we zero out the m-th column of the LSTM weight matrix W and of the matrix from the next layer (see solid vertical lines in fig. 3), there are no outgoing connections from the m-th hidden neuron,[1] and the neuron does not affect the network's output. As a result, we can remove the m-th hidden neuron and all its weights (corresponding rows and columns of W, and the column of the next layer's weight matrix).

To sum up, our three-level hierarchy of the gated RNN sparsification works as follows. We begin with a wide dense network and induce sparsity during training. After training, we firstly remove neurons that do not affect the network's output. For the remaining neurons, some gates are constant, so the neurons' structures are simplified. For the non-constant gates, some weights are sparsified, so we can store weight matrices using a sparse matrix format.

Below we describe how we implement the proposed idea in two sparsification frameworks: pruning (Wen et al. 2018), and Bayesian sparsification (Chirkova, Lobacheva, and Vetrov 2018). We chose (Wen et al. 2018), among other pruning techniques for RNNs, because it is the only one that performs a neuron-wise sparsification. To the best of our knowledge, Bayesian approaches for RNNs with neuron-wise sparsification are not investigated in the literature, therefore, we use a modified version of the method from (Chirkova, Lobacheva, and Vetrov 2018) as a base approach.

Implementation of the idea in pruning

Consider a dataset of N sequences (x^n, y^n) and a model $p(y|x, w, b)$, defined by an RNN, with weights w, and biases b.

To implement our idea about three levels of sparsification, for each neuron η, we define five (intersecting) sets of weights $w_{\eta,i}$, $w_{\eta,f}$, $w_{\eta,g}$, $w_{\eta,o}$, $w_{\eta,h}$. The first four sets of weights correspond to four gates (dotted horizontal lines in fig. 3), and the last set corresponds to the hidden neuron (solid vertical lines in fig. 3). We apply group Lasso regularization (Yuan and Lin 2006) to these groups. We also apply Lasso regularization to the individual weights, to achieve the three-level sparsification hierarchy. The resulting objective is as follows (H denotes all hidden neurons in the LSTM layer):

$$-\sum_{n=1}^{N} \log p(y^n|x^n, w, b) + \lambda_1 \sum_{\eta \in H} (\|w_{\eta,i}\|_2 + \|w_{\eta,f}\|_2 + \\ + \|w_{\eta,g}\|_2 + \|w_{\eta,o}\|_2 + \|w_{\eta,h}\|_2) + \lambda_2 \|w\|_1 \to \min_{w,b}$$

[1]Weights in the m-th column of the LSTM weight matrix W correspond to the outgoing connections from $h_{t-1,m}$ to h_t, and weights in the m-th column of the next layer's weight matrix, correspond to the outgoing connections from $h_{t-1,m}$ to the next layer (e. g., fully-connected output layer).

We use a similar pruning algorithm as in Intrinsic Sparse Structures (ISS) (Wen et al. 2018) but with other weight groups. We prune the weights from scratch. During training, on the forward pass, we set to zero all the individual weights with absolute values less than the threshold. We do not mask gradients for the pruned weights, so the weights can again become non-zero. The value of the threshold is the same for all epochs. After training, if for some η, all the weights in $w_{\eta,h}$ are set to zero, we remove the corresponding hidden neuron as it does not affect the network's output. If for some gate (for example, f), all the weights in $w_{\eta,f}$ are set to zero, we mark this gate as constant.

In contrast to our approach, in (Wen et al. 2018), group Lasso is applied to larger groups in order to eliminate hidden neurons from the model:

$$w_\eta = w_{\eta,i} \cup w_{\eta,f} \cup w_{\eta,g} \cup w_{\eta,o} \cup w_{\eta,h}.$$

This approach does not lead to the sparse gate structures.

Implementation of the idea in Bayesian framework

Preliminaries. Our approach relies on a Sparse variational dropout (SparseVD) (Molchanov, Ashukha, and Vetrov 2017). This model comprises a log-uniform prior over the weights: $p(|w_{ij}|) \propto \frac{1}{|w_{ij}|}$, and a fully factorized normal approximate posterior: $q(w_{ij}|\mu_{ij}, \sigma_{ij}) = \mathcal{N}(w_{ij}|\mu_{ij}, \sigma^2_{ij})$. The biases are treated as deterministic parameters. To find the parameters of the approximate posterior distribution, and the biases, the evidence lower bound (ELBO) is optimized:

$$\sum_{n=1}^{N} \mathbb{E}_{q(w|\mu,\sigma)} \log p(y^n|x^n, w, b) \\ - KL(q(w|\mu,\sigma)||p(w)) \to \max_{\mu,\sigma,b} \quad (2)$$

Because of the log-uniform prior, for the majority of weights, the signal-to-noise ratio μ^2_{ij}/σ^2_{ij} tends to zero. These weights become too noisy and can be removed from the model without affecting the performance. In (Chirkova, Lobacheva, and Vetrov 2018), SparseVD is adapted to RNNs.

Our model. To sparsify the individual weights, we apply SparseVD (Molchanov, Ashukha, and Vetrov 2017) to all the weights w of the RNN, taking into account the recurrent specifics underlined in (Chirkova, Lobacheva, and Vetrov 2018). To compress the LSTM layer and remove hidden neurons, we follow (Louizos, Ullrich, and Welling 2017) and multiply the activations of the hidden neurons by group weights z^h.

The key component of our model is introducing group weights z^i, z^f, z^g, z^o that are multiplied by the preactivations of the gates. The resulting forward pass, through the LSTM layer, looks as follows:

$$f_t = sigm\Big(\big(W^x_f x_t + W^h_f h_{t-1}\big) \odot z^f + b_f\Big)$$

$$\{\text{similarly for } i_t, o_t \text{ and } g_t\}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \qquad h_t = o_t \odot tanh(c_t) \odot z^h$$

Algorithm 1 Forward pass through Bayesian LSTM for one sequence.

```
Require: $[x_1, \dots x_T], c_0, h_0$
Require: Parameters $\mu$, $\sigma$, $b$
1: Sample $\epsilon^x_i, \epsilon^h_i, \dots, \epsilon^i, \epsilon^f, \dots \sim \mathcal{N}(0, I)$
2: $W^x_i = \mu^x_i + \epsilon^x_i \odot \sigma^x_i, \dots;\ z^i = \mu^i + \epsilon^i \odot \sigma^i, \dots$
3:                              // sampling reparametrized weights
4: for $t = 1, \dots, T$: do
5:     $f_t = sigm\Big(\big(W^x_f x_t + W^h_f h_{t-1}\big) \odot z^f + b_f\Big)$,
       similarly for $i_t, o_t, g_t$
6:     $c_t = f_t \odot c_{t-1} + i_t \odot g_t, h_t = o_t \odot tanh(c_t) \odot z^h$
   return $[c_1, \dots, c_T]$, $[h_1, \dots, h_T]$
```

The model is equivalent to multiplying the rows and the columns of the weight matrices by group weights:

$$\hat{w}^h_{f,ij} = w^h_{f,ij} \cdot z^h_i \cdot z^f_j \quad \{\text{similarly for } i, o \text{ and } g\}$$

If some component of z^i, z^f, z^o, z^g is set to zero, the corresponding gate is marked as constant. If some component of z^h is set to zero, we remove the corresponding neuron from the model, as its output is always multiplied by zero.

Training our model. We work with the group weights z in the same way as with the weights w: we approximate the posterior with the fully factorized normal distribution, given the fully factorized log-uniform prior distribution. To find the approximate posterior distribution, we maximize ELBO (2). To estimate an expectation in (2), we sample weights from the approximate posterior distribution, using the same weight parametrization as in SparseVD for RNNs (for the details, see (Chirkova, Lobacheva, and Vetrov 2018)). The forward pass is presented in algorithm 1.

With the integral estimated with one Monte-Carlo sample, the first term in (2) becomes a usual loss function (e. g., cross-entropy in language modeling). The second term is a regularizer depending on the parameters μ and σ (for the exact formula, see (Molchanov, Ashukha, and Vetrov 2017)).

After learning, we zero out all the weights and group weights with the signal-to-noise ratio less than the threshold. At the testing stage, we use the mean values of all the weights and group weights.

Experiments

We perform experiments with LSTM architecture in both sparsification frameworks. For each framework, we use a dense model as a baseline for quality, and compare the two-level sparsification approach (weights+neurons, W+N) and the proposed, three-level, sparsification approach (weights+gates+neurons, W+G+N) in terms of quality and sparsity. We follow the experimental setups of (Wen et al. 2018) and (Chirkova, Lobacheva, and Vetrov 2018) for pruning, and Bayesian sparsification respectively, and use them as baselines for sparsity. We also analyze gate structures, obtained using our method, on different tasks. Please note that we do not compare two frameworks between each other. Our goal is to show that the proposed idea leads to an improvement in both frameworks.

Experimental setup – Pruning

For the pruning framework, we follow Wen et al. (2018) and perform experiments on a word-level language modeling task on the Penn Treebank (PTB) dataset (Marcus, Marcinkiewicz, and Santorini 1993). We train a standard model of Zaremba, Sutskever, and Vinyals (2014) of two sizes (small and large) with an embedding layer, two LSTM layers, and a fully-connected output layer. The embedding and LSTM layers, in the small and large models, contain 200 and 1500 neurons, respectively.

In these experiments, we apply regularization only to the LSTM layers for comparability with ISS results (Wen et al. 2018). The strengths of the individual and group Lasso regularizations are selected using grid search, so that the validation perplexities of ISS, and our model, are approximately equal. Below, we provide more training details for the reproducibility of the results.

Details. All the small models, including the baseline model, are trained without dropout, as in standard TensorFlow implementation. We train them from scratch, for 20 epochs, with Stochastic Gradient Descent (SGD), with a decaying learning rate schedule: the initial learning rate is equal to 1, the learning rate starts to decay after the 4-th epoch, the learning rate decay is equal to 0.6. For the two-level sparsification (W+N), we use group Lasso regularization with $\lambda_1 = 0.002$, and Lasso regularization with $\lambda_2 = 1e-5$. For the three-level sparsification (W+G+N), we use group Lasso regularization with $\lambda_1 = 0.0017$, and Lasso regularization with $\lambda_2 = 1e-5$. We use the threshold $1e-4$ to prune the weights in both models during training.

All the large models, including the baseline model, are trained in the same setting as in (Wen et al. 2018), except for the group Lasso regularization, because we change the weight groups. We use the code provided by the authors. Particularly, we use a binary dropout (Zaremba, Sutskever, and Vinyals 2014), with the same dropout rates. We train the models from scratch, for 55 epochs, with SGD, with a decaying learning rate: the initial learning rate is equal to 1, the learning rate decreases two times during training (after epochs 18 and 36), the learning rate decay is equal to 0.2 and 0.1 for the two- and three-level sparsification, respectively. For the two-level sparsification (W+N), we use group Lasso regularization with $\lambda_1 = 0.0015$, and Lasso regularization with $\lambda_2 = 1e-5$. For the three-level sparsification (W+G+N), we use group Lasso regularization with $\lambda_1 = 0.00125$, and Lasso regularization with $\lambda_2 = 1.5e-5$. We use the same threshold $1e-4$ as in small models.

Experimental setup – Bayesian sparsification

For the Bayesian framework, we follow Chirkova, Lobacheva, and Vetrov (2018) and perform an evaluation on text classification and language modeling tasks. For the text classification experiments, we use the Internet Movie Database (IMDb) dataset (Maas et al. 2011) for binary classification, and AG's Corpus of News Articles (AGNews)

Task	Method	Quality	Compression	Neurons	Gates
Word PTB	Original	120.28 – 114.41	1x	200 − 200	800 – 800
(small)	Pruning W+N (ISS)	110.34 – 106.25	1.44x	72 – 123	288 – 492
Perplexity	Pruning W+G+N	**110.04 – 105.64**	**1.49x**	**64 – 115**	**193 – 442**
Word PTB	Original	82.57 – 78.57	1x	1500 − 1500	6000 − 6000
(large)	Pruning W+N (ISS)	**81.25 – 77.62**	2.97x	324 – 394	1296 – 1576
Perplexity	Pruning W+G+N	**81.24 – 77.82**	**3.22x**	**252 – 394**	**881 – 1418**

Table 1: Quantitative results for pruning. Pruning W+N corresponds to the ISS method of Wen et al. (2018). For language modeling, we evaluate quality on the validation and test sets. Compression is equal to $|w|/|w \neq 0|$. In the last columns, the numbers of remaining hidden neurons and non-constant gates, in LSTM layers, are reported.

Task	Method	Quality	Compression	Neurons	Gates
	Original	**84.1**	1x	128	512
IMDb	Bayes W (SparseVD)	83.62	18567	8	17
Accuracy %	Bayes W+N	83.98	17874x	5	12
	Bayes W+G+N	83.98	**19747x**	**4**	**6**
	Original	**90.6**	1x	512	2048
AGNews	Bayes W (SparseVD)	89.14	561x	34	76
Accuracy %	Bayes W+N	88.55	645x	17	62
	Bayes W+G+N	88.41	**647x**	**14**	**39**
	Original	1.499 − 1.454	1x	1000	4000
Char PTB	Bayes W (SparseVD)	1.472 − 1.429	7.9x	431	1718
Bits-per-char	Bayes W+N	1.478 − 1.430	**10.2x**	**390**	**1560**
	Bayes W+G+N	**1.467 − 1.425**	9.8x	404	1563
	Original	120.28 – 114.41	1x	200 − 200	800 – 800
Word PTB	Bayes W (SparseVD)	114.80 – 109.85	10.52x	55 – 124	218 – 415
(small)	Bayes W+N	110.25 – 104.81	**11.65x**	68 – 110	272 – 392
Perplexity	Bayes W+G+N	**109.98 – 104.45**	11.44x	**52 – 108**	**197 – 349**

Table 2: Quantitative results for Bayesian sparsification. Bayes W corresponds to the SparseVD method of Chirkova, Lobacheva, and Vetrov (2018). For text classification tasks, the vocabulary is sparsified in all sparsification approaches. For language modeling, we evaluate quality on validation and test sets. Compression is equal to $|w|/|w \neq 0|$. In the last columns, the numbers of remaining hidden neurons and non-constant gates, in LSTM layers, are reported.

dataset (Zhang, Zhao, and LeCun 2015) for four-class classification, and an architecture which consists of an embedding layer, one LSTM layer and a fully-connected output layer on the last step. We conduct language modeling experiments on the PTB dataset on both character- and word-level tasks. For character-level language modeling, we use a model with one LSTM layer and a fully-connected output layer, while for word-level language modeling, we use the same small architecture as in pruning. Embedding layers for classification tasks have 300 neurons. The sizes of LSTM layers for all tasks may be found in tab. 2.

In these experiments, we apply regularization to all layers, including the embedding and output layers, for comparability with SparseVD results (Chirkova, Lobacheva, and Vetrov 2018). Since in the text classification tasks, usually only a small number of input words are important, we use additional multiplicative weights to sparsify the input vocabulary following Chirkova, Lobacheva, and Vetrov (2018). For the networks with the embedding layer, in configurations W+N and W+G+N, we also sparsify the embedding components (by introducing group weights z^x multiplied by x_t). To compute the number of remaining neurons or non-constant gates, we use the corresponding rows/columns of W, and the corresponding group weights, z, if applicable. Below, we provide more training details for the reproducibility of the results.

Details. We train our networks using Adam (Kingma and Ba 2015). Baseline networks overfit for all our tasks, therefore, we present the results for them with early stopping. Models for the text classification and character-level language modeling are trained in the same setting as in (Chirkova, Lobacheva, and Vetrov 2018) (we used the code provided by the authors). For the text classification tasks, we use the learning rate equal to 0.0005 and train the Bayesian models for 800 / 150 epochs on IMDb / AGNews. The embedding layer, for IMDb / AGNews, is initialized with word2vec (Mikolov et al. 2013) / GloVe (Pennington, Socher, and Manning 2014). For the language modeling tasks, we train the Bayesian models for 250 / 50 epochs on character-level / word-level tasks using the learning rate of 0.002.

For all the weights that we sparsify, we initialize $\log \sigma$ with -3. We eliminate the weights with the signal-to-noise ratio less than $\tau = 0.05$.

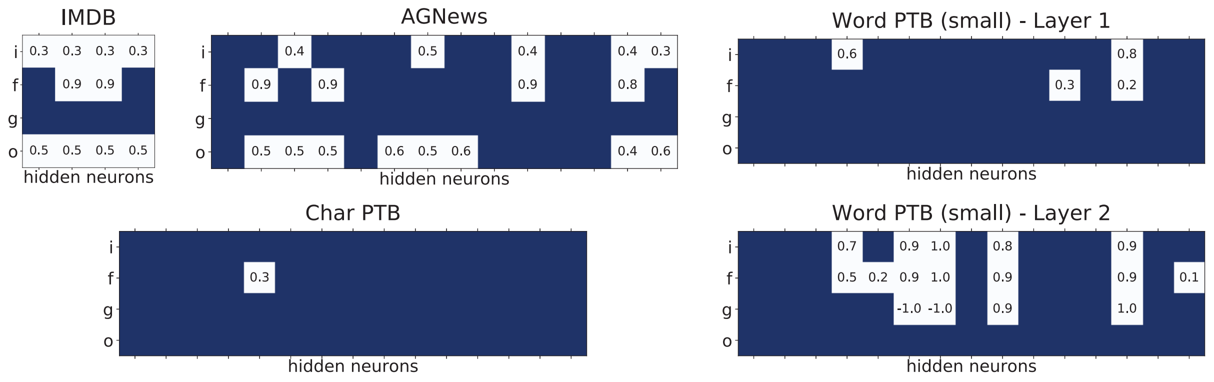

Figure 4: Structure of gate sparsity for text classification and language modeling obtained with the proposed Bayesian approach (Bayes W+G+N). Constant gates are shown in white with corresponding activation values. For language modeling, only 15 randomly chosen active neurons are presented.

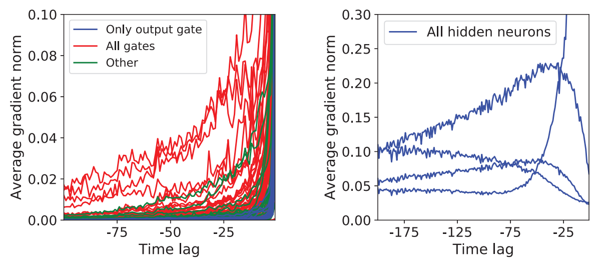

Figure 5: Averaged over validation sequences, the norm of the gradients of hidden neurons, w. r. t. LSTM input, for different time-lag. Left: the second LSTM layer of the word-level language model (neurons with different gate structure are shown in different colors). Right: IMDb classification.

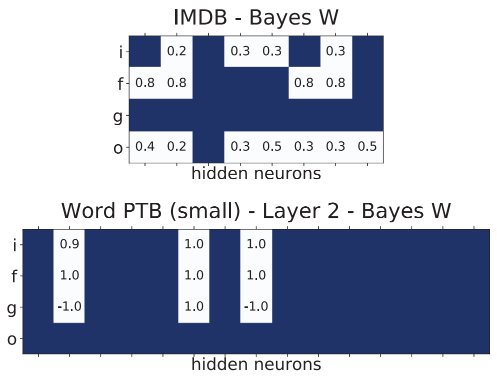

Figure 6: Structure of gate sparsity for text classification and word-level language modeling obtained with Bayes W (without group weights). Constant gates are shown in white, with corresponding activation values. For language modeling, only 15 randomly chosen active neurons are presented.

Quantitative results

The quantitative results for Bayesian sparsification and pruning are shown in tables 1, 2. Pruning W+N, in table 1, corresponds to the ISS method of Wen et al. (2018), Bayes W, in table 2, corresponds to the SparseVD method of Chirkova, Lobacheva, and Vetrov (2018).

In most experiments, the proposed three-level sparsification approach improves the gate-wise and the neuron-wise compression of the model without a significant quality drop. The only exception is the character-level language modeling task, which we discuss later in the qualitative results section. The overall compression is not always higher for the proposed method, because a high neuron-wise compression does not always lead to a high weight-wise compression: if a model uses fewer neurons, it may need more complex dependencies between these neurons, and, as a result, more non-zero weights. The numbers for compression are not comparable between two frameworks because, in pruning, only LSTM layers are sparsified, while in the Bayesian framework, all the weights in the network are sparsified.

Qualitative results

In this section, we analyze the resulting gate structure for different tasks, models, and sparsification approaches.

Gate structure depends on the tasks. Figure 4 shows the typical examples of the gate structure of the remaining hidden neurons, obtained using the Bayesian approach. We observe that the gate structures vary for different tasks. For language modeling tasks, output gates are very important, because models need to, both, store all the information about the input in the memory, and output only the current prediction at each timestep. On the contrary, for the text classification tasks, models need to output the answer only once, at the end of the sequence, hence they rarely use output gates. The character-level language modeling task is more challenging than the word-level one: the model uses the whole gate mechanism to solve it. We think this is the main reason why gate sparsification does not help here.

As can be seen in fig. 4, in the second LSTM layer of the small word-level language model, a lot of neurons have only one non-constant gate — output gate. We investigate

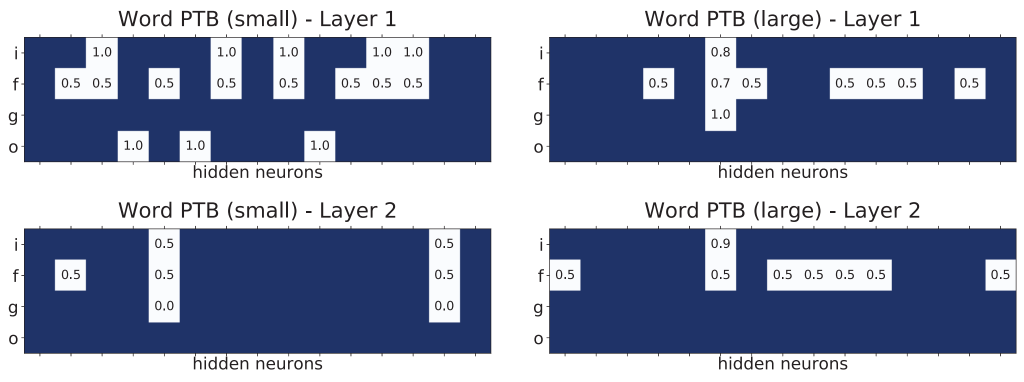

Figure 7: Structure of gate sparsity for language modeling obtained with the proposed pruning approach (Pruning W+G+N). Constant gates are shown in white, with corresponding activation values. In each image, only 15 randomly chosen active neurons are presented.

the described effect and find that the neurons with only non-constant output gate learn short-term dependencies, while neurons with all non-constant gates usually learn long-term dependencies. To show that, we compute the gradients of each hidden neuron of the second LSTM layer w. r. t. the input of this layer, at different lag t, and average the norm of this gradient, over the validation set (see fig. 5, left). The neurons with only non-constant output gates are “short”: the gradient is large only for the latest timesteps and small for the old timesteps. On the contrary, neurons with all non-constant gates, are mostly “long”: the gradient is non-zero even for the old timesteps. In other words, changing input 20–100 steps ago does not affect “short” neurons too much, which is not true for the “long’ neurons. The presence of such “short” neurons is expectable for the language model: neurons without memory quickly adapt to the latest changes in the input sequence and produce actual output.

In fact, for the neurons with only non-constant output gate, the memory cell c_t is either monotonically increasing, or monotonically decreasing, depending on the sign of the constant information flow g, so $tanh(c_t)$ always equals either to -1 or +1,[2] and $h_t = o_t$ or $-o_t$. Hence, these neurons are simplified to vanilla recurrent units.

For the classification tasks, memorizing information about the whole input sequence, until the last timestep, is important, therefore information flow g is not constant and saves information from the input to the memory. In other words, long dependencies are highly important for classification. Gradient plots (fig. 5, right) confirm this claim: the values of the neurons are strongly influenced by both the old and latest inputs. Gradients are bigger for the short lag only for one neuron, because this neuron focuses, not only on the previous hidden states, but also on reading the current inputs.

Gate structure intrinsically exists in LSTM. In the previous experiments, the most visible gate structures were obtained for IMDb classification and for the second LSTM layer of the word-level language modeling task. For these tasks, the same gate structures were detected, even with unstructured sparsification. If we compare the results of the proposed three-level sparsification method in fig. 4, and the results of the unstructured Bayesian approach (Bayes W), in fig. 6, we can see that gate structures for the same task have a similar form. In the case of IMDb classification, the model has a lot of constant output gates and non-constant information flow, in the case of language modeling, the model has neurons with only non-constant output gates. The described effect shows that gate structure intrinsically exists in LSTM and depends on the task. The proposed method utilizes this structure to achieve better compression.

We observe the similar effect when we compare gate structures for the small word-level language models, obtained using Bayes W+G+N (fig. 4) and Pruning W+G+N (fig. 7). We can see that the same gates become constant in these models: constant input and/or forget gate in the first layer and only non-constant output gate in the second layer. For the large model (fig. 7), the structure is slightly different than for the small model. It is expected because there is a big quality gap between these two models, so their intrinsic structure may be different.

Conclusion

In this paper, we propose a sparsification approach for the gated RNNs, that takes into account the intrinsic gated structure and simplifies it. We experiment on several natural language processing tasks and show that the sparsification of the gate preactivations improves neuron-wise compression in two common sparsification frameworks: Bayesian sparsification and pruning. We also perform the analysis of the resulting gate structures and connect the observed gate structures to the specifics of the particular tasks.

Acknowledgments

This research is (in part) based on the work supported by Samsung Research, Samsung Electronics.

[2]Except for the first few epochs because c_t is initialized with 0 value

References

Baoyuan Liu; Min Wang; Foroosh, H.; Tappen, M.; and Penksy, M. 2015. Sparse convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition*.

Chan, W.; Jaitly, N.; Le, Q. V.; and Vinyals, O. 2016. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *International Conference on Acoustics, Speech and Signal Processing*.

Chirkova, N.; Lobacheva, E.; and Vetrov, D. 2018. Bayesian compression for natural language processing. In *Conference on Empirical Methods in Natural Language Processing*.

Cho, K.; van Merriënboer, B.; Gülçehre, Ç.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*.

Ha, D.; Dai, A.; and Le, Q. V. 2017. Hypernetworks. In *International Conference on Learning Representations*.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8).

Jose, C.; Cissé, M.; and Fleuret, F. 2018. Kronecker recurrent units. In *International Conference on Machine Learning*.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Liu, X.; Cao, D.; and Yu, K. 2018. Binarized LSTM language model. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Louizos, C.; Ullrich, K.; and Welling, M. 2017. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*.

Louizos, C.; Welling, M.; and Kingma, D. P. 2018. Learning sparse neural networks through L0 regularization. In *International Conference on Learning Representations*.

Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C. 2011. Learning word vectors for sentiment analysis. In *Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

Marcus, M. P.; Marcinkiewicz, M. A.; and Santorini, B. 1993. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics* 19(2).

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*.

Molchanov, D.; Ashukha, A.; and Vetrov, D. 2017. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*.

Narang, S.; Diamos, G. F.; Sengupta, S.; and Elsen, E. 2017. Exploring sparsity in recurrent neural networks. In *International Conference for Learning Representations*.

Narang, S.; Undersander, E.; and Diamos, G. F. 2018. Block-sparse recurrent neural networks. In *arXiv preprint arXiv:1711.02782*.

Neklyudov, K.; Molchanov, D.; Ashukha, A.; and Vetrov, D. P. 2017. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing*.

Prabhavalkar, R.; Alsharif, O.; Bruguier, A.; and McGraw, I. 2016. On the compression of recurrent neural networks with an application to LVCSR acoustic modeling for embedded speech recognition. In *International Conference on Acoustics, Speech and Signal Processing*.

Ren, M.; Kiros, R.; and Zemel, R. S. 2015. Exploring models and data for image question answering. In *Advances in Neural Information Processing Systems*.

Scardapane, S.; Comminiello, D.; Hussain, A.; and Uncini, A. 2017. Group sparse regularization for deep neural networks. *Neurocomputing* 241(C).

See, A.; Luong, M.-T.; and Manning, C. D. 2016. Compression of neural machine translation models via pruning. In *SIGNLL Conference on Computational Natural Language Learning*.

Tibshirani, R. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)* 58.

Tjandra, A.; Sakti, S.; and Nakamura, S. 2017. Compressing recurrent neural network with tensor train. In *arXiv preprint arXiv:1705.08052*.

Wang, P.; Xie, X.; Deng, L.; Li, G.; Wang, D.; and Xie, Y. 2018. HitNet: Hybrid ternary recurrent neural network. In *Advances in Neural Information Processing Systems*.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*.

Wen, W.; He, Y.; Rajbhandari, S.; Zhang, M.; Wang, W.; Liu, F.; Hu, B.; Chen, Y.; and Li, H. 2018. Learning intrinsic sparse structures within long short-term memory. In *International Conference on Learning Representations*.

Wu, Y.; Schuster, M.; Chen, Z.; Le, Q. V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; Klingner, J.; Shah, A.; Johnson, M.; Liu, X.; Kaiser, L.; Gouws, S.; Kato, Y.; Kudo, T.; Kazawa, H.; Stevens, K.; Kurian, G.; Patil, N.; Wang, W.; Young, C.; Smith, J.; Riesa, J.; Rudnick, A.; Vinyals, O.; Corrado, G.; Hughes, M.; and Dean, J. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. In *arXiv preprint arXiv:1409.2329*.

Yuan, M., and Lin, Y. 2006. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B* 68.

Zaremba, W.; Sutskever, I.; and Vinyals, O. 2014. Recurrent neural network regularization. In *arXiv preprint arXiv:1409.2329*.

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*.