Bayesian Sparsification of Recurrent Neural Networks

Ekaterina Lobacheva [* 1 2] **Nadezhda Chirkova** [* 1 3] **Dmitry Vetrov** [1 4]

Abstract

Recurrent neural networks show state-of-the-art results in many text analysis tasks but often require a lot of memory to store their weights. Recently proposed Sparse Variational Dropout (Molchanov et al., 2017) eliminates the majority of the weights in a feed-forward neural network without significant loss of quality. We apply this technique to sparsify recurrent neural networks. To account for recurrent specifics we also rely on Binary Variational Dropout for RNN (Gal & Ghahramani, 2016b). We report 99.5% sparsity level on sentiment analysis task without a quality drop and up to 87% sparsity level on language modeling task with slight loss of accuracy.

1. Introduction

Recurrent neural networks (RNNs) are among the most powerful models for natural language processing, speech recognition, question-answering systems and other problems with sequential data (Chan et al., 2016; Amodei 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) or speech recognition (Amodei et al., 2016) modern RNN architectures incorporate a huge number of parameters. To use these models on portable devices with limited memory, for instance, smartphones, the model compression is desired. High compression level may also lead to an acceleration of RNNs. In addition, compression regularizes RNNs and helps to avoid overfitting.

There are a lot of RNNs compression methods based on specific weight matrix representations (Tjandra et al., 2017; Le et al., 2015) or sparsification via pruning (Narang et al.,

---

[*]Equal contribution [1]National Research University Higher School of Economics, Moscow, Russia [2]Kaspersky Lab, Moscow, Russia [3]Lomonosov Moscow State University, Moscow, Russia [4]Yandex, Moscow, Russia. Correspondence to: Ekaterina Lobacheva <elobacheva@hse.ru>, Nadezhda Chirkova <nchirkova@hse.ru>.

2017). In this paper we focus on RNNs compression via sparsification. Most of the methods from this group are heuristic and require time-consuming hyperparameters tuning.

Recently Molchanov et. al. (2017) proposed a principled method based on variational dropout for sparsification of fully connected and convolutional networks. A probabilistic model was described in which parameters controlling sparsity are tuned automatically during neural network training. This model called Sparse Variational Dropout (Sparse VD) leads to extremely sparse solutions without a significant quality drop. However, this technique was not previously investigated for RNNs.

In this paper we apply Sparse VD to recurrent neural networks. To take into account the specifics of RNNs we rely on some insights underlined in the paper by Gal & Ghahramani (2016b) where they explain the proper way to use binary dropout in RNNs from the Bayesian point of view. In the experiments we show that LSTMs with Sparse VD yield high sparsity level with just a slight drop in quality. We achieved 99.5% sparsity level on sentiment analysis task and up to 87.6% in character level language modeling experiment.

2. Preliminaries

2.1. Bayesian Neural Networks

Consider a neural network with weights ω modeling the dependency of the target variables $y = \{y^1, \ldots, y^\ell\}$ on the corresponding input objects $X = \{x^1, \ldots, x^\ell\}$. In a Bayesian neural network the weights ω are treated as random variables. With the prior distribution $p(\omega)$ we search for the posterior distribution $p(\omega|X, y)$ that will help to find expected target value during inference. In the case of neural networks, true posterior is usually intractable but it can be approximated by some parametric distribution $q_\lambda(\omega)$. The quality of this approximation is measured by the KL-divergence $KL(q_\lambda(\omega)||p(\omega|X, y))$. The optimal parameter λ can be found by maximization of the variational lower bound w.r.t. λ:

$$\mathcal{L} = \sum_{i=1}^{\ell} \mathbb{E}_{q_\lambda(\omega)} \log p(y^i|x^i, \omega) - KL(q_\lambda(\omega)||p(\omega)) \quad (1)$$

The expected log-likelihood term in (1) is usually approximated by Monte-Carlo sampling. To make the MC estimation unbiased, the weights are parametrized by a deterministic function: $\omega = g(\lambda, \xi)$, where ξ is sampled from some non-parametric distribution (the reparameterization trick (Kingma & Welling, 2013)). The KL-divergence term in (1) acts as a regularizer and is usually computed or approximated analytically.

2.2. Sparse Variational Dropout

Dropout (Srivastava, 2013) is a standard technique for regularization of neural networks. It implies that inputs of each layer are multiplied by a randomly generated noise vector. The elements of this vector are usually sampled from Bernoulli or Gaussian distribution with the parameters tuned using cross-validation. Kingma et al. (2015) interpreted Gaussian dropout from a Bayesian perspective that allowed to tune dropout rate automatically during model training. Later this model was extended to sparsify fully connected and convolutional neural networks resulting in a model called Sparse Variational Dropout (Sparse VD) (Molchanov et al., 2017).

Consider one dense layer of a feed-forward neural network with an input of the size n, an output of the size m and a weight matrix W. Following Kingma et al. (2015), in Sparse VD the prior on the weights is a fully factorized log-uniform distribution $p(|w_{ij}|) \propto \frac{1}{|w_{ij}|}$ and the posterior is searched in the form of fully factorized normal distribution:

$$q(w_{ij}|m_{ij}, \alpha_{ij}) = \mathcal{N}(m_{ij}, \alpha_{ij}m_{ij}^2). \quad (2)$$

Employment of such form of the posterior distribution is equivalent to putting multiplicative (Kingma et al., 2015) or additive (Molchanov et al., 2017) normal noise on the weights in the following manner:

$$w_{ij} = m_{ij}\xi_{ij}, \quad \xi_{ij} \sim \mathcal{N}(1, \alpha_{ij}), \quad (3)$$

$$w_{ij} = m_{ij} + \epsilon_{ij}, \quad \epsilon_{ij} \sim \mathcal{N}(0, \sigma_{ij}^2), \quad \alpha_{ij} = \frac{\sigma_{ij}^2}{m_{ij}^2}. \quad (4)$$

The representation (4) is called additive reparameterization (Molchanov et al., 2017). It reduces the variance of the gradients of $\mathcal{L}$ w.r.t. m_{ij}. Moreover, since a sum of normal distributions is a normal distribution with computable parameters, the noise may be applied to the preactivation (input vector times weight matrix W) instead of W. This trick is called the local reparameterization trick (Wang & Manning, 2013; Kingma et al., 2015) and it reduces the variance of the gradients even further and makes training more efficient.

In Sparse VD optimization of the variational lower bound (1) is performed w.r.t. $\{M, \log\sigma\}$. The KL-divergence factorizes over the weights and its terms depend only on α_{ij} because of the specific choice of the prior (Kingma et al., 2015):

$$KL(q(w_{ij}|m_{ij}, \alpha_{ij})||p(w_{ij})) = k(\alpha_{ij}). \quad (5)$$

Each term can be approximated as follows (Molchanov et al., 2017):

$$k(\alpha) \approx 0.64\sigma(1.87 + 1.49\log\alpha) - \\ -0.5\log(1 + \alpha^{-1}) + C. \quad (6)$$

KL-divergence term encourages large values of α_{ij}. If $\alpha_{ij} \to \infty$ for a weight w_{ij}, the posterior over this weight is a high-variance normal distribution and it is beneficial for model to put $m_{ij} = 0$ as well as $\sigma_{ij} = \alpha_{ij}m_{ij}^2 = 0$ to avoid inaccurate predictions. As a result, the posterior over w_{ij} approaches zero-centered δ-function, the weight does not affect the network's output and can be ignored.

2.3. Dropout for Recurrent Neural Networks

Yet another Bayesian model was proposed by Gal & Ghahramani (2016a) to explain the binary dropout. On this base, a recipe how to apply a binary dropout to the RNNs properly was proposed by Gal & Ghahramani (2016b).

The recurrent neural network takes a sequence $x = [x_0, \ldots, x_T]$, $x_t \in \mathbb{R}^n$ as an input and maps it into the sequence of hidden states:

$$h_t = f_h(x_t, h_{t-1}) = g_h(x_t W^x + h_{t-1}W^h + b_1) \\ h_i \in \mathbb{R}^m, \ h_0 = \bar{0} \quad (7)$$

Throughout the paper, we assume that the output of the RNN depends only on the last hidden state:

$$y = f_y(h_T) = g_y(h_T W^y + b_2). \quad (8)$$

Here g_h and g_y are some nonlinear functions. However, all the techniques we discuss further can be easily applied to the more complex setting, e.g. language model with several outputs for one input sequence (one output for each time step).

Gal & Ghahramani (2016b) considered RNNs as Bayesian networks. The prior on the recurrent layer weights $\omega = \{W^x, W^h\}$ is a fully factorized standard normal distribution. The posterior is factorized over the rows of weights, and each factor is searched as a mixture of two normal distributions:

$$q(w_k^x|m_k^x) = p^x\mathcal{N}(0, \sigma^2 I) + (1 - p^x)\mathcal{N}(m_k^x, \sigma^2 I),$$

$$q(w_j^h|m_j^h) = p^h\mathcal{N}(0, \sigma^2 I) + (1 - p^h)\mathcal{N}(m_j^h, \sigma^2 I), \quad (9) \\ k = \overline{1, n}, \quad j = \overline{1, m}.$$

Under assumption $\sigma \approx 0$ sampling the row of weights from such posterior means putting all the weights from this row either to 0 (drop the corresponding input neuron) or to some

learned values. Thus this model is a probabilistic analog of binary dropout with dropout rates p^x and p^h.

After unfolding the recurrence in the network, the maximization of the variational lower bound for such model looks as follows:

$$\sum_{i=1}^{\ell} \int q(\omega|M) \log\Big(y^i \big| f_y\big(f_h(x_T^i, f_h(\ldots f_h(x_1^i, h_0^i)))\big)\Big) d\omega -$$
$$-KL\Big(q(\omega|M)\|p(\omega)\Big) \to \max_M \qquad (10)$$

Each integral in the first part of (10) is estimated with MC integration with a single sample $\hat{\omega}_i \sim q(\omega|M)$. To make this estimation unbiased: (a) the weights sample $\hat{\omega}_i$ should remain the same for all time steps $t = \overline{1, T}$ for a fixed object; (b) dropout rates p^x and p^h should be fixed because the distribution we are sampling from depends on them. The KL-divergence term from (10) is approximately equivalent to L_2 regularization of the variational parameters M.

Finally, this probabilistic model leads to the following dropout application in RNNs: we sample a binary mask for the input and hidden neurons, one mask per object for all moments of time, and optimize the L_2-regularized log-likelihood with the dropout rates and the weight of L_2-regularization chosen using cross-validation. Also, the same dropout technique may be applied to forward connections in RNNs, for example in embedding and dense layers (Gal & Ghahramani, 2016b).

The same technique can be applied to more complex architectures like LSTM in which the information flow between input and hidden units is controlled by the gate elements:

$$i = sigm(h_{t-1}W_i^h + x_t W_i^x) \quad o = sigm(h_{t-1}W_o^h + x_t W_o^x)$$
$$f = sigm(h_{t-1}W_f^h + x_t W_f^x) \quad g = tanh(h_{t-1}W_g^h + x_t W_g^x)$$

$$c_t = f \odot c_{t-1} + i \odot g \quad h_t = o \odot tanh(c_t) \qquad (11)$$

Here binary dropout masks for input and hidden neurons are generated 4 times: individually for each of the gates i, o, f and input modulation g.

3. Variational Dropout for RNN sparsification

Dropout for RNNs proposed by Gal & Ghahramani (2016b) helps to avoid overfitting but is very sensitive to the choice of the dropout rates. On the other hand, Sparse VD allows automatic tuning of the Gaussian dropout parameters individually for each weight which results in the model sparsification. We combine these two techniques to sparsify and regularize RNNs.

Following Molchanov et al. (2017), we use the fully factorized log-uniform prior and approximate the posterior with a fully factorized normal distribution over the weights

$\omega = \{W^x, W^h\}$:

$$q(w_{ki}^x | m_{ki}^x, \sigma_{ki}^x) = \mathcal{N}\big(m_{ki}^x, \sigma_{ki}^{x\,2}\big),$$
$$q(w_{ji}^h | m_{ji}^h, \sigma_{ji}^h) = \mathcal{N}\big(m_{ji}^h, \sigma_{ji}^{h\,2}\big), \qquad (12)$$

where σ_{ki}^x and σ_{ji}^h have the same meaning as in additive reparameterization (4).

To train the model, we maximize the variational lower bound approximation

$$\sum_{i=1}^{\ell} \int q(\omega|M, \sigma) \log\Big(y^i \big| f_y\big(f_h(x_T^i, f_h(\ldots f_h(x_1^i, h_0^i)))\big)\Big) d\omega -$$
$$-\sum_{k,i=1}^{n,m} k\left(\frac{\sigma_{ki}^{x\,2}}{m_{ki}^{x\,2}}\right) - \sum_{j,i=1}^{m,m} k\left(\frac{\sigma_{ji}^{h\,2}}{m_{ji}^{h\,2}}\right) \qquad (13)$$

w. r. t. $\{M, \log\sigma\}$ using stochastic mini-batch methods. Here the recurrence in the expected log-likelihood term is unfolded as in (10) and the KL is approximated using (6). The integral in (13) is estimated with a single sample $\hat{\omega}_i \sim q(\omega|M, \alpha)$ per input sequence. We use the reparameterization trick (for unbiased integral estimation) and additive reparameterization (for gradients variance reduction) to sample both input-to-hidden and hidden-to-hidden weight matrices $\widehat{W}^x, \widehat{W}^h$. To reduce the variance of the gradients and for more computational efficiency we also apply the local reparameterization trick to input-to-hidden matrix $\widehat{W}^x$ moving the noise from the weights to the pre-activations:

$$(x_t \widehat{W}^x)_j = \sum_{k=1}^{n} x_{t,k} m_{kj}^x + \epsilon_j \sqrt{\sum_{k=1}^{n} x_{t,k}^2 \sigma_{kj}^{x\,2}}, \qquad (14)$$
$$\epsilon_j \sim \mathcal{N}(0, 1).$$

As a result, only 2-dimensional noise on input-to-hidden connections is required for each mini-batch: we generate one noise vector of length m for each object in a mini-batch.

The local reparameterization trick cannot be applied to the hidden-to-hidden matrix W^h. We use the same sample $\widehat{W}^h$ for all moments of time, therefore in the multiplication $h_{t-1}\widehat{W}^h$ the vector h_{t-1} depends on $\widehat{W}^h$ and the rule about the sum of normally distributed random variables cannot be applied. Since usage of 3-dimensional noise (2 dimensions of $\widehat{W}^h$ and a mini-batch size) is too resource-consuming we sample one noise matrix for all objects in a mini-batch for efficiency:

$$\hat{w}_{ji}^h = m_{ji}^h + \sigma_{ji}^j \epsilon_{ji}^h, \quad \epsilon_{ji}^h \sim \mathcal{N}(0, 1). \qquad (15)$$

The final framework works as follows: we sample Gaussian additive noise on the input-to-hidden preactivations (one per input sequence) and hidden-to-hidden weight matrix (one per mini-batch), optimize the variational lower

bound (13) w. r. t. $\{M, \log \sigma\}$, and for many weights we obtain the posterior in the form of a zero-centered δ-function because the KL-divergence encourages sparsity. These weights can then be safely removed from the model.

In LSTM the same prior-posterior pair is consisered for all input-to-hidden and hidden-to-hidden matrices and all computations stay the same. The noise matrices for input-to-hidden and hidden-to-hidden connections are generated individually for each of the gates i, o, f and input modulation g.

4. Experiments

We perform experiments with LSTM as the most popular recurrent architecture nowadays. We use Theano (Theano Development Team, 2016) and Lasagne (Dieleman et al., 2015) for implementation. The source code will be available soon at `https://github.com/tipt0p/SparseBayesianRNN`. We demonstrate the effectiveness of our approach on two diverse problems: Character Level Language Modeling and Sentiment Analysis. Our results show that Sparse Variational Dropout leads to a high level of sparsity in recurrent models without a significant quality drop.

We use the dropout technique of Gal & Ghahramani (2016b) as a baseline because it is the most similar dropout technique to our approach and denote it VBD (variational binary dropout).

According to Molchanov et al. (2017), training neural networks with Sparse Variational Dropout from a random initialization is troublesome, as a lot of weights may become pruned away before they could possibly learn something useful from the data. We observe the same effect in our experiments with LSTMs, especially with more complex models. LSTM trained from a random initialization may have high sparsity level, but also have a noticeable quality drop. To overcome this issue we start from pre-trained models that we obtain by training networks without Sparse Variational Dropout for several epochs.

Weights in models with Sparse Variational Dropout cannot converge exactly to zero because of the stochastic nature of the training procedure. To obtain sparse networks we explicitly put weights with high corresponding dropout rates to 0 during testing as in Molchanov et al. (2017). We use the value $\log \alpha = 3$ as a threshold.

For all weights that we sparsify using Sparse Variational Dropout, we initialize $\log \sigma^2$ with -6. We optimize our networks using Adam (Kingma & Ba, 2015).

Networks without any dropout overfit for both our tasks, therefore, we present results for them with early stopping. Throughout experiments we use the mean values of the

Table 1. Results on sentiment regression task. Prediction quality is reported in MSE (the lower the better). Sparsity levels reported for W^x and W^h separately in percents of zero weights. For Sparse VD methods initialization types are reported in brackets.

Method (init)	Test MSE	Sparsity x - h %
No dropout	0.1518	–
VBD	0.1488	–
SparseVD (random)	0.1526	99.91 – 99.90
SparseVD (no dropout)	0.1503	**99.95 – 99.92**
SparseVD (VBD)	**0.1475**	99.63 – 99.49

weights to evaluate the model quality (we do not sample weights from posterior on the evaluating phase). This is a common practice when working with dropout.

4.1. Sentiment Analysis

Data. Following Gal & Ghahramani (2016b) we evaluated our approach on the sentiment analysis regression task. The dataset is constructed based on Cornell film reviews corpus collected by Pang & Lee (2005). It consists of approximately 10 thousands non-overlapping segments of 200 words from the reviews. The task is to predict corresponding film scores from 0 to 1. We use the provided train and test partitions.

Setup. We use networks with one embedding layer of 128 units, one LSTM layer of 128 hidden units, and finally, a fully connected layer applied to the last output of the LSTM (resulting in a scalar output). All weights are initialized in the same way as in Gal & Ghahramani (2016b). We train our networks using batches of size 128 and a learning rate of 0.001 for 1000 epochs. We also clip the gradients with threshold 0.1. For all layers with VBD we use dropout rate 0.3 and weight decay 10^{-3} (these parameters are chosen using cross validation).

Results. As a baseline, we train the network without any dropout and with VBD on all layers.

In this experiment, our goal is to check the applicability of Sparse VD for recurrent networks, therefore we apply it only to LSTM layer. For embedding and dense layers we use VBD.

We try both start training of the network with Sparse VD from random initialization and from two different pre-trained models. The first pre-trained model is obtained after 4 epochs of training of the network without any dropout. The second one is obtained after 200 epochs of training of the network with VBD on all layers. We choose number of pretraining epochs using models quality on cross-validation.

The results are shown in Table 1. In this task our approach achieves extremely high sparsity level both from random initialization and from pre-trained models. Sparse VD networks trained from pre-trained models achieve even better quality than baselines. Note that models already have this sparsity level after approximately 20 epochs.

4.2. Character Level Language Modeling

Data. Following Mikolov et al. (2011) we use the Penn Treebank Corpus to train our Language Model (LM). The dataset contains approximately 6 million characters and a vocabulary of 50 characters. We use the provided train, validation and test partitions.

Setup. We use networks with one LSTM layer of 1000 hidden units to solve the character level LM task. All weight matrices of the networks are initialized orthogonally and all biases are initialized with zeros. Initial values of hidden and cell elements are trainable and also initialized with zeros. We train our networks on non-overlapping sequences of 100 characters in batches of 64 using a learning rate of 0.002 for 50 epochs, and clip gradients with threshold 1. For all layers with VBD we use dropout rate 0.25 and do not use weight decay (these parameters are chosen using quality of VDB model on validation set).

Results. As a baseline, we train the network without any dropout and with VBD only on recurrent weights (hidden-to-hidden). Semeniuta et al. (2016) showed that for this particular task applying dropout for feed-forward connections additionally to VBD on recurrent ones does not improve the network quality. We observe the same effect in our experiments.

In this experiment we try to sparsify both LSTM and dense layers therefore we apply Sparse VD for all layers. We try both start training of the network with Sparse VD from random initialization and from two different pre-trained models. The first pre-trained model is obtained after 11 epochs of training of the network without any dropout. The second one is obtained after 50 epochs of training of the network with VBD on recurrent connections. We choose the number of pretraining epochs using models quality on validation set.

The results are shown in Table 2. Here we do not achieve such extreme sparsity level as in the previous experiment. This effect may be a consequence of the higher complexity of the task. Also in LM problem we have several outputs for one input sequence (one output for each time step) instead of one output in Sentiment regression. As a result the log-likelihood part of the loss function is much stronger for LM task and regularizer can not sparsify the network so effectively. Here we see that the balance between the likelihood and the regularizer varies a lot for different tasks

Table 2. Results on character level Language Modelling task. Prediction quality is reported in bits-per-character (lower is better). Sparsity levels reported for W^x, W^h and W^y separately in percents of zero weights. For Sparse VD methods initialization types are reported in brackets.

Method (init)	Valid	Test	Sparsity x - h - y %
No dropout	1.499	1.453	–
VBD for W^h	**1.394**	**1.358**	–
SparseVD (random)	1.506	1.461	**79.7 – 88.0 – 71.7**
SparseVD (no dropout)	1.454	1.414	61.9 – 60.0 – 43.9
SparseVD (VBD for W^h)	1.409	1.372	52.2 – 49.8 – 37.9

with RNNs and should be explored futher.

Fig. 1 and 2 show the progress of test quality and network sparsity level through the training process. Sparse VD network trained from random initialization underfits and therefore has a slight quality drop in comparison to baseline network without regularization. Sparse VD networks trained from pre-trained models achieve much higher quality but have lower sparsity levels than the one trained from random initialization. Better pretrained models are harder to sparsify. The quality of the model pretrained with VBD drops on the first epochs while the sparsity grows, and the model does not fully recover later.

5. Related Work

5.1. Regularization of RNNs

Deep neural networks often suffer from overfitting, and different regularization techniques are used to improve their generalization ability. Dropout (Srivastava, 2013) is a popular method of neural networks regularization. The first successful implementations of this method for RNNs (Pham et al., 2013; Zaremba et al., 2014) applied dropout only for feed-forward connections and not recurrent ones.

Introducing dropout in recurrent connections may lead to a better regularization technique but its straightforward implementation may results in underfitting and memory loss through time (Semeniuta et al., 2016). Several ways of dropout application for recurrent connections in LSTM were proposed recently (Moon et al., 2015; Gal & Ghahramani, 2016b; Semeniuta et al., 2016). These methods inject binary noise into different parts of LSTM units. Semeniuta et al. (2016) shows that proper implementation of dropout for recurrent connections is important not only for effective regularization but also to avoid vanishing gradients.

Bayer et al. (2013) successfully applied fast dropout (Wang & Manning, 2013), a deterministic approximation of
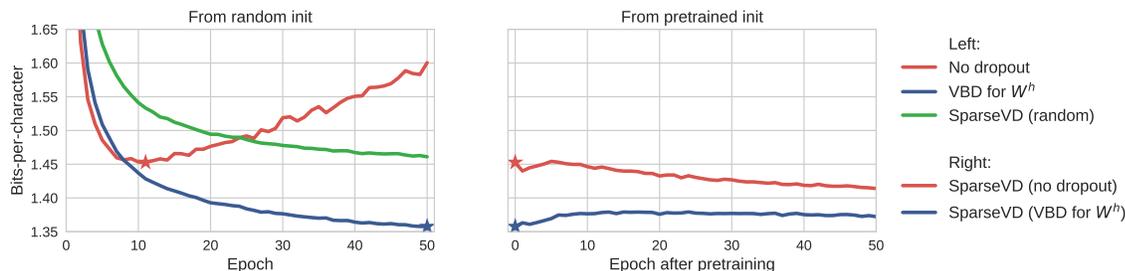
Figure 1. Results on character level Language Modelling task: prediction quality on the test set in bits-per-character (lower is better). Left — training from random initialization, right — training from pretrained initialization. Stars correspond to pretrained models. For Sparse VD methods initialization types are reported in brackets.
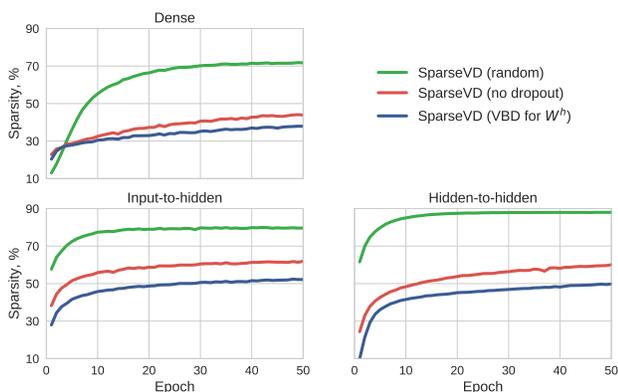

Figure 2. Results on character level Language Modelling task: sparsity levels for W^y, W^x, W^h in LSTM. Initialization types are reported in brackets.

dropout, to RNNs. Krueger et al. (2016) introduced zoneout which forces some hidden units to maintain their previous values, like in feedforward stochastic depth networks (Huang et al., 2016).

5.2. Compression of RNNs

Reducing RNN size is an important and rapidly developing area of research. One possible concept is to represent large RNN weight matrix by some approximation of the smaller size. For example, Tjandra et. al. (2017) use Tensor Train decomposition of the weight matrices and Le et al. (2015) approximate this matrix with Kronecker product. Hubara et. al (2016) limit the weights and activations to binary values proposing a way how to compute gradients w. r. t. them.

Another concept is to start with a large network and to reduce its size during or after training. 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.

6. Discussion and future work

When applying Sparse VD to RNNs we rely on the dropout for RNNs proposed by Gal & Ghahramani (2016b). The reason is that this dropout technique for RNNs is the closest one to Sparse VD approach. However, there are several other dropout methods for recurrent networks that outperform this baseline (Semeniuta et al., 2016; Krueger et al., 2016). Comparison with them is our future work. Combining Sparse VD with these latest dropout recipes is also an interesting research direction. The challenge here is that the noise should be put on the neurons or gates instead of the weights as in our model. However, there are several recent papers (Kirill Neklyudov, 2017; Christos Louizos, 2017) where group sparsity methods are proposed for fully connected and convolutional networks. These methods can be used to solve the underlined problem.

The comparison of our approach with other RNN sparsification techniques is still a work-in-progress. It would be interesting to perform this comparison on larger networks, for example, for speech recognition task.

One more curious direction of the research is to sparsify not only recurrent layer but an embedding layer too. It may have a lot of parameters in the tasks with large dictionary, such as word based language modeling.

References

Amodei, Dario, Ananthanarayanan, Sundaram, Anubhai, Rishita, and et al. Deep speech 2 : End-to-end speech recognition in english and mandarin. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

Bayer, Justin, Osendorfer, Christian, Chen, Nutan, Urban, Sebastian, and van der Smagt, Patrick. On fast dropout and its applicability to recurrent networks. *CoRR*, abs/1311.0701, 2013.

Chan, William, Jaitly, Navdeep, Le, Quoc V., and Vinyals, Oriol. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *ICASSP*, 2016.

Christos Louizos, Karen Ullrich, Max Welling. Bayesian compression for deep learning. In *arXiv preprint arXiv:1705.08665*, 2017.

Dieleman, Sander, Schlter, Jan, Raffel, Colin, and et al. Lasagne: First release., 2015.

Gal, Yarin and Ghahramani, Zoubin. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, 2016a.

Gal, Yarin and Ghahramani, Zoubin. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016b.

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

Huang, Gao, Sun, Yu, Liu, Zhuang, Sedra, Daniel, and Weinberger, Kilian Q. Deep networks with stochastic depth. In *ECCV (4)*, volume 9908 of *Lecture Notes in Computer Science*, pp. 646–661. Springer, 2016.

Hubara, Itay, Courbariaux, Matthieu, Soudry, Daniel, El-Yaniv, Ran, and Bengio, Yoshua. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv e-prints*, abs/1609.07061, September 2016.

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

Kingma, Diederik P. and Welling, Max. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

Kingma, Diederik P., Salimans, Tim, and Welling, Max. Variational dropout and the local reparameterization trick. *CoRR*, abs/1506.02557, 2015.

Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha Dmitry Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *arXiv preprint arXiv:1705.07283*, 2017.

Krueger, David, Maharaj, Tegan, Kramár, János, and et al. Zoneout: Regularizing rnns by randomly preserving hidden activations. *CoRR*, abs/1606.01305, 2016.

Le, Quoc V., Jaitly, Navdeep, and Hinton, Geoffrey E. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941, 2015.

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

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

Moon, Taesup, Choi, Heeyoul, Lee, Hoshik, and Song, Inchul. Rnndrop: A novel dropout for rnns in asr. In *Automatic Speech Recognition and Understanding (ASRU)*, 2015.

Narang, Sharan, Diamos, Gregory F., Sengupta, Shubho, and Elsen, Erich. Exploring sparsity in recurrent neural networks. *CoRR*, abs/1704.05119, 2017.

Pang, Bo and Lee, Lillian. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Association for Computational Linguistics*, 2005.

Pham, Vu, Kermorvant, Christopher, and Louradour, Jrme. Dropout improves recurrent neural networks for handwriting recognition. *CoRR*, abs/1312.4569, 2013.

Ren, Mengye, Kiros, Ryan, and Zemel, Richard S. Exploring models and data for image question answering. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems*, 2015.

Semeniuta, Stanislau, Severyn, Aliaksei, and Barth, Erhardt. Recurrent dropout without memory loss. In *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers*, 2016.

Srivastava, Nitish. *Improving neural networks with dropout*. PhD thesis, University of Toronto, 2013.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, 2016.

Tjandra, Andros, Sakti, Sakriani, and Nakamura, Satoshi. Compressing recurrent neural network with tensor train. *CoRR*, abs/1705.08052, 2017.

Wang, Sida and Manning, Christopher. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, and et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. In *arXiv preprint arXiv:1409.2329*, 2014.