Sparse Bayesian Variational Learning with Matrix Normal Distributions

Viktor Rudnev [1 2] **Dmitry Kropotov** [1 2]

Abstract

The application of variational Bayesian methods to neural networks has been limited by the choice of the posterior approximation family. One could use a simple family like a normal distribution with independent variables, but that results in a low quality of the approximation and optimization issues. In the paper we propose to use Matrix Normal distribution (MN) for variational approximation family. While being more flexible, this family supports efficient reparameterization and Riemannian optimization procedures. We apply this family for Bayesian neural networks sparsification through Automatic Relevance Determination (Kharitonov et al., 2018). We show that MN family here outperforms simpler fully-factorized Gaussians, especially for the case of group sparsification, while remaining as computationally efficient as the latter. We also analyze application of MN distribution for inference in Variational Auto-Encoder model.

1. Introduction

The application of variational Bayesian methods to neural networks has been limited by the choice of the posterior approximation family. One could use a simple family like a multivariate normal distribution with independent variables, but that results in a low quality of the approximation and optimization issues. On the other hand, there are more powerful families such as Normalizing Flows (Rezende & Mohamed, 2015) or implicit distributions (Huszár, 2017). They provide better predictions and uncertainty estimates, but their usage is limited due to their poor scalability and optimization stability issues. So, there is a need for new classes of variational approximation families that would be stable and efficient, but at the same time more flexible than simple independent Gaussian distributions.

[*]Equal contribution [1]National Research University Higher School of Economics, Joint Samsung-HSE Lab [2]Lomonosov Moscow State University, Moscow, Russia. Correspondence to: Viktor Rudnev <viktor.rudnev@gmail.com>, Dmitry Kropotov <dmitry.kropotov@gmail.com>.

Tensor decomposition methods are an efficient tool for storing and manipulating huge data vectors and matrices. Recently these methods were successfully applied for solving inference problems in probabilistic graphical models (Novikov et al., 2014), for compression of fully connected and convolutional neural networks (Novikov et al., 2015; Garipov et al., 2016), for scalable learning in Gaussian Processes models (Izmailov et al., 2017), and others.

One special use case of tensor decomposition methods is the so-called Matrix Normal (MN) distribution (Dutilleul, 1999). It is a multivariate normal distribution with covariance stored as a rank-1 tensor-train (TT) matrix. The structure imposed by this TT format results in storing just two matrices – one defining the distribution of the resulting matrix on the rows, and one defining the distribution on the columns. It can be shown that for one layer fully-connected neural network for regression problem the true distribution for weight matrix is indeed MN distribution (see section 4.1 for details). On the other hand, fully-factorized Gaussian approximation would ignore many correlations in posterior distribution even for this simple model. That is why it seems beneficial to consider MN distributions for variational Bayesian inference in large neural networks.

In this paper we analyze using MN approximation family within variational Bayesian inference. First, we study the effects of using MN in learning Bayesian neural networks with Automatic Relevance Determination (ARD) for sparsification (Kharitonov et al., 2018). We consider both independent and group sparsification. For the latter case we observe significant benefits in using MN. Besides, we analyze using MN in learning variational autoencoder models (VAE). Here latent variables do not have the matrix structure and hence using matrix distributions like MN doesn't leed to significant improvements.

2. Related Work

There are a lot of works considering the expressivity problem for posterior approximations. They include Normalizing Flows (Rezende & Mohamed, 2015), implicit distributions (Huszár, 2017), and semi-implicit distributions (Yin & Zhou, 2018). However, their application to learning neural network weight distributions is limited.

Consider a typical convolutional neural network. Usually it consists of tens of millions of weights, and each layer can contain millions of weights by itself (e.g., a layer with 256 input and 512 output channels, and a 3x3 receptive field).

Normalizing flows do not provide a viable solution for generating such large-dimensional values, since to obtain good quality, flow length should be around several hundreds. This isn't feasible, considering flow stability issues and overall computing time.

Fully implicit models usually use some kind of discriminator network for estimating KL-divergence term. Although there are some attempts for doing this (Pawlowski et al., 2017), training a discriminator with such high-dimensional inputs is hardly possible. First it requires an even bigger network just to discriminate the weights, and then it requires enormous training sample size to result in a meaningful gradient estimate.

So, the only viable application of these posterior families would be in a hierarchical model, where we first generate some low-dimensional latent variable, and then generate neural network's weights from some simple distribution, parameterized by this latent variable.

The other aspect is that we can't analytically compute the KL divergence between the posterior and its approximation when using these families. This would result in less stable learning, since we wouldn't be able to eliminate the noise coming from empirical estimates of KL divergence.

MN distribution doesn't have mentioned deficiencies. It explicitly uses matrix structure for modelling weights of neural networks. MN distributions and the structure of Kronecker product were used in many papers that involved modelling neural network's weight distributions. Examples include K-FAC second-order optimization (Martens & Grosse, 2015), weights sampling procedure (Nado et al., 2018), scalable Laplace approximation for posterior weights' distribution (Ritter et al., 2018) and others.

3. Preliminaries

3.1. Bayesian Sparsification using ARD

Let $\mathcal{D} = (x_i, y_i)_{i=1}^N$ be a dataset of N samples where x_i are observable variables and y_i are target labels. Suppose we have some parametric model $p(\mathcal{D}|\theta) = \prod_{i=1}^N p(y_i|x_i, \theta), \theta \in \mathbb{R}^D$ (e.g., a deep neural network) mapping x to the corresponding y using parameters θ. The parameters θ have a prior distribution $p(\theta|\alpha)$ which is itself parameterized by hyperparameters $\alpha \in \mathbb{R}^H$. Following the Bayesian approach, we wish to find the posterior distribution $p(\theta|\mathcal{D}, \alpha) = p(\mathcal{D}|\theta)p(\theta|\alpha)/p(\mathcal{D}|\alpha)$. We choose hyperparameters α such that the marginal likelihood (evidence) of the dataset is maximized:

$$\alpha^* = \arg\max_\alpha p(\mathcal{D}|\alpha) = \arg\max_\alpha \int p(\mathcal{D}|\theta)p(\theta|\alpha)\,\mathrm{d}\theta.$$

This is *Empirical Bayes* (EB) approach to hyperparameter selection. In particular, when $p(\theta|\alpha) = \prod_{i=1}^D \mathcal{N}(\theta_i|0, \alpha_i^{-1})$, this procedure is called *automatic relevance determination* (ARD).

Since in the case of deep neural networks marginal likelihood $p(\mathcal{D}|\alpha)$ is intractable, we use doubly stochastic variational inference (DSVI) to find an approximate posterior $q(\theta|\phi)$ from some parametric family. This is achieved by maximizing *Evidence Lower Bound* (ELBO) w.r.t. variational parameters ϕ and hyperparameters α:

$$\log p(\mathcal{D}|\alpha) \geq \mathcal{L}(\phi, \alpha) = \mathbb{E}_{q(\theta|\phi)}[\log p(\mathcal{D}|\theta)] - \\ - D_{\mathrm{KL}}(q(\theta|\phi)\|p(\theta|\alpha)) \to \max_{\phi,\alpha}. \quad (1)$$

Now suppose that $p(\theta|\alpha) = \prod_{i=1}^D \mathcal{N}(\theta_i|0, \alpha_i^{-1})$ (ARD prior) and $q(\theta|\mu, \sigma) = \prod_{i=1}^D \mathcal{N}(\theta_i|\mu_i, \sigma_i^2)$. Maximizing ELBO (1) w.r.t. hyperparameters α can be done analytically: $\alpha_i^* = (\mu_i^2 + \sigma_i^2)^{-1}$. Substituting this result back to ELBO (1) leads to the following optimization problem:

$$\mathcal{L}_{\mathrm{ARD}}(\mu, \sigma) = \sum_{i=1}^N \mathbb{E}_{q(\theta|\mu,\sigma)}[\log p(y_i|x_i, \theta)] - \\ -\frac{1}{2}\sum_{j=1}^D \log\left(1 + \frac{\mu_j^2}{\sigma_j^2}\right) = \mathcal{L}_{\mathcal{D}}(\mu, \sigma) + \mathbb{R}_{\mathrm{ARD}}(\mu, \sigma) \to \max_{\mu,\sigma}.$$

In practice, we estimate the gradients of $\mathcal{L}_{ARD}$ w.r.t. variational parameters μ, σ using the local reparametrization trick. It means that we don't sample multiple weight matrices W in each layer of neural network but instead sample directly from the distribution of transformed variables Wx, where x is an input to a layer. During optimization of $\mathcal{L}_{ARD}$ some values of α_i^* go to infinity thus zeroing the corresponding weights.

However, the described approach leads to independent sparsification of components of weight matrices. This could hardly be used for accelerating neural network computations. Here we are more interested in eliminating whole neurons in calculation graph for neural network that corresponds to sparsifying full rows/columns of weight matrices.

3.2. Group Sparsification

Let's consider one layer of neural network with weight matrix W and use the following prior:

$$p(W|\tau, \gamma) = \prod_{i=1}^n \prod_{j=1}^m \mathcal{N}(W_{ij}|0, \tau_i^{-1}\gamma_j^{-1}). \quad (2)$$

This prior is equivalent to ARD prior with a particular structure for hyperparameters, i.e. $\alpha_{im+j} = \tau_i \gamma_j$ (Kropotov et al., 2010). Large values of τ_i or γ_j lead to sparsification of the correspondent row/column of matrix W.

ELBO derivation for the case of group ARD prior (2) does not differ much from the previous section results. The only difference is that we can't find closed form solution for τ and γ, and hence we must optimize ELBO w.r.t. these parameters as well, e.g. with a gradient descent.

Using fully-factorized Gaussian distributions for variational proxy q in this model brings many issues with model training. First, this approximation does not take into account correlations between weights in same rows/columns of weight matrices. Second, since this family is poor, the KL-divergence term in ELBO is outweighing log-likelihood term for the case of large neural networks. This results in low classification accuracy for both training and testing sets. In practice the last issue is solved by adding a coefficient to the term $\mathbb{R}_{\text{ARD}}(\mu, \sigma)$ with proper annealing procedure for this coefficient. However, good annealing schedule is usually hard to find and problem dependent.

3.3. Variational Auto-Encoders

Let $\mathcal{D} = (x_i)_{i=1}^N$ be a dataset of N samples, where x_i are observable variables. Suppose we have some parametric model $p(\mathcal{D}|z, \theta) = \prod_{i=1}^N p(x_i|z_i, \theta), \theta \in \mathbb{R}^D, z_i \in \mathbb{R}^Q$ (e.g., a deep neural network), called the decoder, generating x_i from latent code z_i using parameters θ. The latent code z_i has a standard normal prior distribution $p(z_i) = \mathcal{N}(z_i|0, I)$. Full log-likelihood of the model can be written as follows:

$$\log p(\mathcal{D}|\theta) = \sum_{i=1}^N \log \int p(x_i|z_i, \theta) p(z_i) \, \mathrm{d}z_i.$$

As one may see, this likelihood is intractable, so we use variational inference with a parametric variational distribution $q(z_i|x_i, \phi)$, called the encoder:

$$\log p(\mathcal{D}|\theta) \geq \text{ELBO}(\theta, \phi) = \\ = \sum_{i=1}^N \mathbb{E}_{q(z_i|x_i,\phi)} \log \frac{p(x_i|z_i, \theta) p(z_i)}{q(z_i|x_i, \phi)} \to \max_{\theta, \phi}.$$

Since we need both to sample and estimate point density for the variational distribution, it is often selected to be simple, e.g., a fully-factorized Gaussian distribution with means and variances calculated by a deep neural network with parameters ϕ and input x_i. This results in low values of the model log-likelihood and uneven latent space usage.

3.4. Matrix Normal distributions

Matrix $W \in \mathbb{R}^{n \times m}$ follows Matrix Normal distribution (MN) $\mathcal{MN}(M, U, V)$ if its probability density function has the following form:

$$p(W) = \frac{\exp\left(-\frac{1}{2} \operatorname{Tr}\left[V^{-1}(W - M)^T U^{-1}(W - M)\right]\right)}{(2\pi)^{nm/2} |V|^{n/2} |U|^{m/2}},$$

where $M \in \mathbb{R}^{n \times m}$ is mean matrix, $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{m \times m}$ are covariance matrices, i.e. symmetric and positive definite ones.

Alternatively, one can notice that saying

$$W \sim \mathcal{MN}(M, U, V)$$

is equivalent to the following statement:

$$\operatorname{vec}(W) \sim \mathcal{N}(\operatorname{vec}(M), V \otimes U),$$

where $\operatorname{vec}(W)$ is column-wise vectorization of W, and $\otimes$ denotes the Kronecker product.

Now let's describe basic operations we need for applying MN distributions to variational learning.

Covariance matrices U and V in MN must be positive definite. So it is natural to parameterize these matrices using their Cholesky factors A and B, i.e. $U = AA^T, V = BB^T$ and both matrices A, B are lower-triangular. We also force diagonal elements of A, B to be positive by considering their logarithm values as parameters.

Using the introduced parameterization it is easy to calculate normalizing constant for MN distribution:

$$\log\left((2\pi)^{nm/2} |V|^{n/2} |U|^{m/2}\right) = \\ \frac{mn}{2} \log(2\pi) + \frac{n}{2} \log\det V + \frac{m}{2} \log\det U = \\ \frac{mn}{2} \log(2\pi) + n \sum_{j=1}^m \log B_{jj} + m \sum_{i=1}^n \log A_{ii}. \quad (3)$$

Algorithm 1 Computing the KL-term between MN posterior and diagonal covariance normal prior

Input: $\mathcal{MN}(W|M, AA^T, BB^T)$, where $M \in \mathbb{R}^{n \times m}$, $\mathcal{N}(\alpha_i|\mu_i, \sigma_i)$
Compute logarithm of normalizing constant for posterior using (3);
Compute log-determinant of the prior's covariance diagonal matrix: $\log\det\Sigma_{\text{prior}} = \sum_{i=1}^D 2\log\sigma_i$;
Compute $\operatorname{Tr}\left(\Sigma_{\text{prior}}^{-1}\Sigma_{\text{post}}\right) = \sum_{k,l=1}^{n,m} \sigma_{km+l}^{-1} (AA^T)_{kk} (BB^T)_{ll}$;
Compute $\frac{1}{2} \operatorname{vec}(M)^T \Sigma_{\text{prior}} \operatorname{vec}(M)$;
Finally obtain the KL-term:
$\text{KL} = \frac{1}{2}\big(\log\det\Sigma_{\text{prior}} - \log\det\Sigma_{\text{post}} + \operatorname{Tr}(\Sigma_{\text{prior}}^{-1}\Sigma_{\text{post}}) + \mu^T \Sigma_{\text{prior}} \mu\big)$.

Algorithm 2 Bayesian neural network sparsification with MN posterior

Input: a dataset $\mathcal{D} = (x_i, y_i)_{i=1}^N$, initial values of α_l and ϕ_l
repeat
  Sample a minibatch $(\hat{x}_j, \hat{y}_j)_{j=1}^B \sim \mathcal{D}$;
  Perform the forward pass using local reparametrization: $o_j \sim p(y_j|x_j, \theta)$;
  Calculate original task's log-likelihood estimate using obtained predictions: $L = \frac{N}{B}\sum_{j=1}^B \mathcal{L}(o_j, y_j)$;
  Compute KL-term using algorithm 1: $\mathrm{KL} = D_{\mathrm{KL}}(q(\theta|\phi)\|p(\theta|\alpha))$
  Obtain $\mathrm{ELBO} = L - \mathrm{KL}$;
  Perform backpropagation and optimization step using stochastic gradient descent or one of its modifications, such as Adam(Kingma & Ba, 2014);
until convergence

The exponential term in MN density can be calculated as follows:

$$\mathrm{Tr}\left[V^{-1}(W-M)^T U^{-1}(W-M)\right] = \mathrm{Tr}\, C^T C,$$

where $C = A^{-1}(W-M)B^{-T}$.

Then we need to know how to efficiently perform the reparametrization trick. So, if we have $\mathrm{vec}(X) \sim \mathcal{N}(0, I)$, and define

$$W = M + AXB^T,$$

then

$$W \sim \mathcal{MN}(M, AA^T, BB^T).$$

This way we can quickly draw samples from MN distribution just in two matrix-matrix products.

Also we need to know how to perform the local reparametrization trick. This means that we need a way to quickly obtain samples from the distribution of $W^T x$, where $W \sim \mathcal{MN}(M, AA^T, BB^T)$ and x is some fixed vector. Since MN is a special case of normal distribution, $W^T x$ also follows some normal distribution $\mathcal{N}(W^T x|\mu, ZZ^T)$. So, if $\varepsilon \sim \mathcal{N}(0, I)$, then

$$o = \mu + Z\varepsilon = M^T x + \sqrt{x^T A x}\varepsilon B^T$$

follows the distribution of $W^T x$. So here we don't need to sample a separate W matrix for each sample x, and instead we sample directly from the transformed distribution. That brings down the gradient variance (Kingma et al., 2015), and the cost of computation becomes $O(nm + n^2 + m^2)$ per one x sample, where $W \in \mathbb{R}^{n\times m}$.

4. Proposed Method

4.1. Motivation

The idea to consider matrix normal distribution for variational family comes from simple observation for Bayesian multivariate linear regression model. Suppose we have a dataset $\mathcal{D} = (x_i, y_i)_{i=1}^N$, where $x_i \in \mathbb{R}^{in}$ are feature vectors and $y_i \in \mathbb{R}^{out}$ – multidimensional real target variables. Let's consider the following model:

$$\begin{aligned} &p(y_i \mid x_i, W, \beta) = \mathcal{N}(y_i \mid Wx_i, \beta^{-1}I), \\ &p(W \mid \alpha) = \mathcal{MN}(W \mid 0, I, \mathrm{diag}(\alpha)^{-1}I), \\ &p(Y \mid X, W, \alpha, \beta) = p(W \mid \alpha)\prod_{i=1}^N p(y_i \mid x_i, W, \beta). \end{aligned} \tag{4}$$

Here $W \in \mathbb{R}^{out\times in}$ is weight matrix, $X \in \mathbb{R}^{N\times in}$ are all feature vectors from the dataset $\mathcal{D}$ and $Y \in \mathbb{R}^{N\times out}$ are all target variables from the dataset $\mathcal{D}$. The equation (4) is equivalent to independent ARD prior for all columns of matrix W. Then it can be shown that posterior distribution for W in this model is calculated analytically:

$$\begin{aligned} &p(W \mid Y, X, \alpha, \beta) = \mathcal{MN}(W \mid M, U, V), \\ &U = I,\ V = (\beta X^T X + \mathrm{diag}(\alpha))^{-1}, \\ &M^T = V\beta X^T Y. \end{aligned} \tag{5}$$

The formula (5) means that the true posterior distribution for weight matrix W actually lies in the family of Matrix Normal distributions. It means that using independent Gaussian distribution as variational family would ignore many correlations between components of matrix W. That is why it seems beneficial to choose MN distribution for variational family.

4.2. MN posterior for sparsifying neural networks

To perform bayesian neural network sparsification with MN posterior, one needs to know how to

- perform local reparametrization trick,
- compute KL divergence between MN posterior and the prior.

The first one was described in the previous section. And the latter one is described in algorithm 1.

The overall steps for learning a bayesian sparsification model are summarized in algorithm 2. They are the same both for independent and group sparsification regimes.

The resulting scheme has the same $O(nm)$ time complexity per sample as the original one (Kingma et al., 2015) with diagonal covariance normal posterior.

Algorithm 3 Learning a variational auto-encoder with encoder predicting MN distributions

Input: a dataset $\mathcal{D} = (x_i)_{i=1}^N$, initial values of α_l and ϕ_l
repeat
  Sample a minibatch $(\hat{x}_j)_{j=1}^B \sim \mathcal{D}$;
  Perform encoder's forward pass using the reparametrization trick: $z_j \sim q(z_j|x_j, \phi) = \mathcal{MN}(M_j, U_j, V_j)$;
  Perform decoder's forward pass using obtained latent variables and compute reconstruction log-likelihood: $L_j = \mathcal{L}(x_j|f(z_j, \theta))$;
  Compute prior probabilities $p(z_j)$;
  Compute entropy of the variational distribution;
  Compute ELBO by combining previously obtained values: $\text{ELBO} = \mathbb{E}_x \mathbb{E}_{q(z|x)}[\log p(x|z) + \log p(z) - \log q(z|x)]$;
  Perform backpropagation and optimization step using stochastic gradient descent or one of its modifications, such as Adam (Kingma & Ba, 2014);
until convergence

4.3. MN distribution for VAE

Consider a Variational Auto-Encoder model. The overall steps for learning this model are described in algorithm 3 and roughly follow the original VAE scheme described in (Kingma & Welling, 2013).

5. Experiments

All experiments were conducted using Intel® Xeon® CPU E5-2690 v4, and NVIDIA® Tesla® P40 GPU. All code is written using Python 3.6.7, NumPy 1.15.4, PyTorch 0.4.1.

For sparsification tests we selected three widely-known datasets: MNIST (LeCun et al., 1998), Fashion MNIST (Xiao et al., 2017), and CIFAR-10 (Krizhevsky & Hinton, 2009).

The first one is fairly easy, and one can get very high accuracy values without much effort. So, after learning ELBO consists mostly of the model's KL divergence term, that indicates how good is our posterior approximation. That way we can show, that we actually present a better approximation family, which allows for better prediction quality.

Fashion MNIST is more recent and is much harder. Getting 95% test quality requires heavy augmentation and quite deep neural networks, e.g. a ResNet. We show that we can train very simple networks that can tackle harder problems on small datasets.

And with CIFAR-10 we demonstrate that it is possible to learn large-scale sparse (and group-sparse) bayesian neural networks, that get good test accuracy on hard datasets, from scratch.

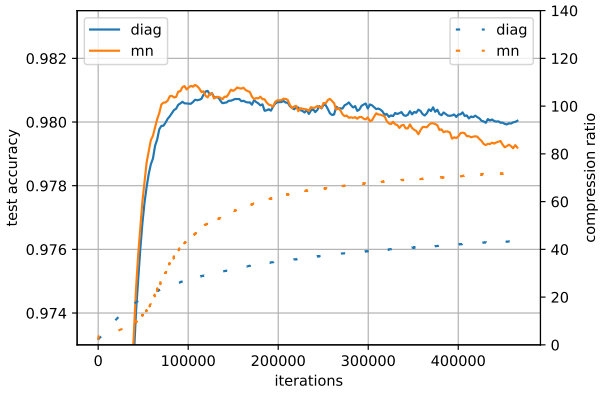

Figure 1. Learning a sparse fully-connected neural network on MNIST.

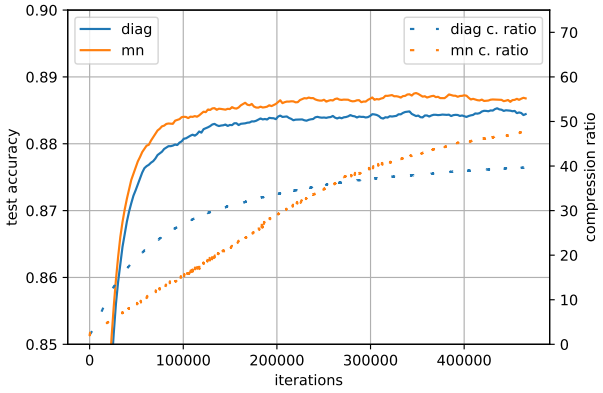

Figure 2. Learning a sparse fully-connected neural network on Fashion MNIST.

5.1. Sparsifying on MNIST and Fashion MNIST

For MNIST we used a fully-connected network, that consists of three layers with 300 hidden units each and ELU (Clevert et al., 2015) activation in between. As one may see on figure 1, matrix normal posterior obtains better test accuracy, as well as compression ratio. We used the same fully-connected network on Fashion MNIST (figure 2) to reach the same conclusions.

We then tested a fairly standard convolutional neural network. It consists of 9 layers. Architecture of this network is 1-10-(mp)-20-(mp)-40-(mp)-80-(mp)-(flatten, resulting in 80 units)-10, where (mp) denotes 2x2 maxpool, numbers before (flatten) denote 3x3 convolution layers with corresponding number of channels and ELU activation, and the

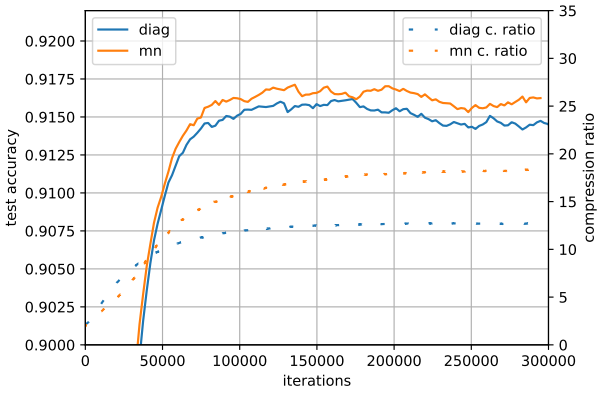

Figure 3. Learning a sparse convolutional neural network on Fashion MNIST.

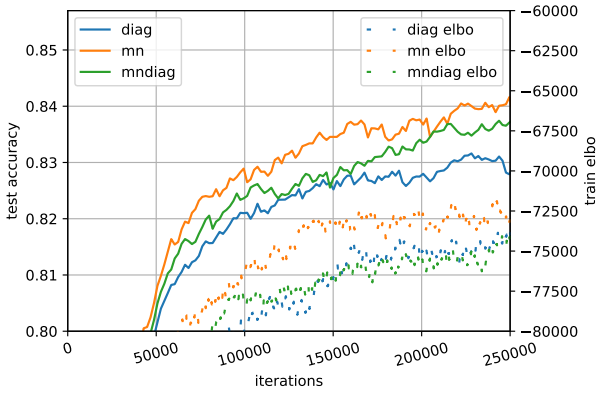

Figure 5. Test accuracy on CIFAR-10 with the VGG-like network and different posterior approximation families in the group-sparse learning task.

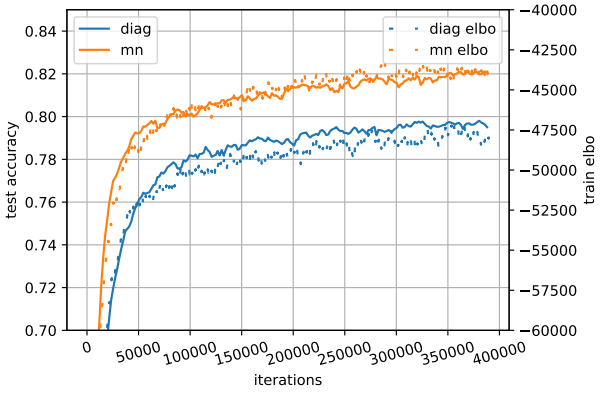

Figure 4. Learning a bayesian VGG-like network on CIFAR-10.

number after (flatten) denote a fully connected layer. On figure 3 one may notice same improvements as with previous tests.

5.2. CIFAR-10 bayesian learning

For CIFAR-10 dataset we test a VGG-like network with 19 layers. It consists of 4 blocks with three 3x3 convolutional layers and ELU activations, followed by a 2x2 max-pool layer. Channel counts are 3-10-10-10-(mp)-20-20-20-(mp)-40-40-40-(mp)-20-20-20-(mp)-(flatten, resulting in 80 units)-10. In this subsection we don't do any sparsification, instead we set our prior variances to a single learnable constant across all weights.

On figure 4 we see that there's a big difference in ELBO values. Also, there's almost a 2% difference in test quality. It means that MN is significantly better than diagonal covariance normal distribution in fairly large CNN bayesian learning task, despite having the same order of magnitude of parameters.

5.3. Structured sparsity on CIFAR-10

In experiments with group sparsification we use additionally horizontal flip augmentation to help learning.

First, we use the same VGG-like network to compare the diagonal covariance normal (DIAG), MN distribution and the MN+Diagonal distribution. The latter one is the same as the MN, but we add a learnable positive-definite diagonal matrix to the covariance matrix. This way we obtain a distribution that is explicitly a superset of both, while MN isn't a superset of DIAG by itself. One deficiency of this combined distribution is that we lose ability to compute exact values of log-determinants of the covariance, since it doesn't possess Kronecker-factored structure we used before. So, we settle on using a lower-bound to ELBO, which doesn't much impact on the final test accuracy.

As one may notice, figure 5 shows that we obtain an around 1% difference in test quality between DIAG and MN-based distributions. MN and MN+Diagonal don't differ much in test accuracy. This means that MN already possesses required capabilities for structured sparse learning, despite not being a superset of DIAG. But instead, MN+Diagonal provides both better sparsification rate of DIAG and better test accuracy of MN. Final count of weights left after sparsifying is shown in the table 1.

Then, we proceed to obtaining better test accuracy by scaling up network by roughly 4 times. That is if one layer possessed 10 units, it would have 20 units in the scaled up version. Also, this time we used the exponential KL term annealing procedure. We weren't able to train the DIAG

	DIAG (18610 WEIGHTS LEFT OF 62270)
A	0.3-0.9-0.9-1.8-3.6-3.6-7.2-14.4-14.4-7.2-3.6-3.6
W	0.2-0.4-0.5-0.9-1.8-2.3-3.1-4.4-2.3-1.4-0.4-0.4
%	30-61-40-52-49-35-57-69-84-80-89-90
	MN (27764 WEIGHTS LEFT OF 62270)
A	0.3-0.9-0.9-1.8-3.6-3.6-7.2-14.4-14.4-7.2-3.6-3.6
W	0.2-0.5-0.6-1.2-1.4-2.4-5.7-6.7-5.2-0.6-0.2-2.5
%	31-41-29-36-62-33-21-54-64-92-94-29
	MN+DIAGONAL (19936 WEIGHTS LEFT OF 62270)
A	0.3-0.9-0.9-1.8-3.6-3.6-7.2-14.4-14.4-7.2-3.6-3.6
W	0.2-0.3-0.6-1.5-1.7-2-3.7-3.8-3.1-1.2-0.7-0.4
%	37-65-32-18-51-44-48-74-79-84-80-88

Table 1. Group-sparse learning of the VGG-like network with different posterior approximation families. A – total number of weights in layers (in thousands), W – number of weights left after deleting zeroed weights (in thousands), % – fraction of weights dropped (in percents).

model, even though there was no KL influence in the ELBO for first several dozens of epochs. It turns out the network becomes so large that DIAG isn't trainable from scratch without changing the model, e.g., without adding Dropout or Batch Normalization.

We obtain 87.1% test accuracy with this network. Per-layer number of weights left after training is shown in the table 2. We can cut around 50% of weights in total, resulting in just around 121000 weights overall, which is an improvement upon previous works (Changpinyo et al., 2017), in which around 1000000 weights were needed to achieve the same test accuracy.

	FREE ZEROING (120884 WEIGHTS LEFT OF 246940)
A	0.5-3.6-3.6-7.2-14.4-14.4-28.8-57.6-57.6-28.8-14.4-14.4
W	0.4-1.8-2.0-4.8-9.7-10.3-21.8-33.3-25.5-7.6-1.3-1.4
%	22-50-44-32-33-29-24-42-56-74-91-90
	ROW-ONLY (3120 ROWS LEFT OF 4527)
A	0.3-1.8-1.8-1.8-3.6-3.6-3.6-7.2-7.2-7.2-3.6-3.6
W	0.3-1.5-1.5-1.8-3.4-3.6-3.6-7.0-4.9-2.9-0.3-0.5
%	0-18-19-1-4-1-1-4-32-59-91-85
	COLUMN-ONLY (413 COLUMNS LEFT OF 540)
A	20-20-20-40-40-40-80-80-80-40-40-40
W	20-16-20-37-37-40-74-61-48-14-6-40
%	0-20-0-7-7-0-7-24-40-65-85-0

Table 2. Group-sparse learning of the 4x version of the VGG-like network with MN posterior. Free zeroing: A – total number of weights in layers (in thousands), W – number of weights left after deleting zeroed weights (in thousands), % – fraction of weights dropped (in percents). Row pruning: same, but A and W is in hundreds of rows. Column pruning: same, but A and W are just numbers of columns.

5.4. VAE on MNIST

To test variational auto-encoders we use a fully-connected encoder and decoder. Both consist of 3 layers with 300 hidden units and ELU activations. We use it to auto-encode Statically Binarized MNIST. This setup follows experiments described in (Tomczak & Welling, 2017).

We test three different families as encoder output: diagonal covariance multivariate normal (DIAG), matrix normal (MN), and fully-parameterized multivariate normal (MV). The last one is implemented as MN distribution with $|z| \times 1$ input-output configuration.

We select two regimes for testing. The first one is $|z| = 40$, i.e., when there is enough dimensions to cover the dataset complexity, therefore setting more dimensions won't get better likelihood values. The second one is $|z| = 9$, i.e., when there's not enough dimensions in the latent space, so tightness of packing the latent code matters more. For the first regime we selected a 20×2 MN input-output configuration, and for the second we selected a 3×3 MN configuration.

We trained all networks with Adam with batch size 100, learning rate $5 \cdot 10^{-4}$, and other parameters set to default for at most 1000 epochs.

To reduce overfitting we used two procedures. First, we used early stopping by looking at validation set likelihood values. Second, we used exponential KL term annealing over first 300 epochs.

As one may see in table 3, there's a natural progression in how well the models perform. DIAG performs the least well of three. MV performs the best, being better than DIAG by around 1 nat in both regimes. And MN is in the between of the first two. That shows that MN is better than DIAG for variational auto-encoding, although they have same order of magnitude of parameter count and computation time.

	$\|z\| = 40$		$\|z\| = 9$	
FAMILY	LL	ELBO	LL	ELBO
DIAG	-89.14	-94.36	-96.80	-101.19
MN	-88.8	-94.03	-96.61	-101.08
MV	**-87.94**	**-93.01**	**-96.20**	**-100.87**

Table 3. Variational Auto-Encoder per-object test set ELBO and log-likelihood with a fully-connected encoder and decoder with varying encoder output distributions. DIAG stands for diagonal covariance normal distribution, MV denotes the fully-parameterized multivariate normal distribution, MN is a 2×20 matrix-normal for $|z| = 40$, and a 3×3 matrix-normal distribution for $|z| = 9$

6. Future Work

MN's covariance can be viewed as a rank-1 tensor-train (TT) tensor. We can use this fact to apply Riemannian optimization technique (Kressner et al., 2014). This would gain in significant improvements in optimization stability, speed, namely in number of iterations, required to converge.

Another possible way to improve upon shown results would be to continue working on MN+Diagonal family for sparsification, described in the previous section. As we've written there, we couldn't obtain exact values of ELBO because of log-determinant computation issues. It is still possible to perform the local reparameterization trick efficiently. So, we could try to get a much tighter bound for the corresponding ELBO, e.g., with techniques described in (Fitzsimons et al., 2017).

7. Results

In bayesian neural network sparsification setting, when using MN distribution instead of independent normal distributions for each weight, we obtain better likelihood values, better test accuracy, and sparsity levels, while maintaining only a drop by a factor of 2 in iteration times. In Variational Auto-Encoder setting, we achieve better likelihood values on the test set and more efficient packing of objects in the latent variable space. However, for VAE case improvements are almost negligible because the latent space doesn't have a matrix structure – a property for which MN distribution is particularly tailored.

References

Changpinyo, S., Sandler, M., and Zhmoginov, A. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Dutilleul, P. The mle algorithm for the matrix normal distribution. *Journal of statistical computation and simulation*, 64(2):105–123, 1999.

Fitzsimons, J., Granziol, D., Cutajar, K., Osborne, M., Filippone, M., and Roberts, S. Entropic trace estimates for log determinants. In Ceci, M., Hollmén, J., Todorovski, L., Vens, C., and Džeroski, S. (eds.), *Machine Learning and Knowledge Discovery in Databases*, pp. 323–338, Cham, 2017. Springer International Publishing. ISBN 978-3-319-71249-9.

Garipov, T., Podoprikhin, D., Novikov, A., and Vetrov, D. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

Huszár, F. Variational inference using implicit distributions. *arXiv preprint arXiv:1702.08235*, 2017.

Izmailov, P., Novikov, A., and Kropotov, D. Scalable gaussian processes with billions of inducing inputs via tensor train decomposition. *arXiv preprint arXiv:1710.07324*, 2017.

Kharitonov, V., Molchanov, D., and Vetrov, D. Variational dropout via empirical bayes. *arXiv preprint arXiv:1811.00596*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

Kressner, D., Steinlechner, M., and Vandereycken, B. Low-rank tensor completion by riemannian optimization. *BIT Numerical Mathematics*, 54(2):447–468, Jun 2014. ISSN 1572-9125. doi: 10.1007/s10543-013-0455-z.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Kropotov, D., Vetrov, D., Wolf, L., and Hassner, T. Variational relevance vector machine for tabular data. In *Proceedings of 2nd Asian Conference on Machine Learning*, pp. 79–94, 2010.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417, 2015.

Nado, Z., Snoek, J., Grosse, R., Duvenaud, D., Xu, B., and Martens, J. Stochastic gradient langevin dynamics that exploit neural network structure. 2018.

Novikov, A., Rodomanov, A., Osokin, A., and Vetrov, D. Putting mrfs on a tensor train. In *International Conference on Machine Learning*, pp. 811–819, 2014.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

Pawlowski, N., Brock, A., Lee, M. C., Rajchl, M., and Glocker, B. Implicit weight uncertainty in neural networks. *arXiv preprint arXiv:1711.01297*, 2017.

Rezende, D. J. and Mohamed, S. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Ritter, H., Botev, A., and Barber, D. A scalable laplace approximation for neural networks. 2018.

Tomczak, J. M. and Welling, M. VAE with a vampprior. *arXiv preprint arXiv:1705.07120*, 2017.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yin, M. and Zhou, M. Semi-implicit variational inference. *arXiv preprint arXiv:1805.11183*, 2018.