arXiv:1705.07283v1 [stat.ML] 20 May 2017

Structured Bayesian Pruning via Log-Normal Multiplicative Noise

Kirill Neklyudov
Yandex, HSE
kirusha@yandex-team.ru

Dmitry Molchanov
Yandex, HSE, Skoltech
dmolch@yandex-team.ru

Arsenii Ashukha
MIPT, HSE
ars.ashuha@phystech.edu

Dmitry Vetrov
Yandex, HSE
dvetrov@hse.ru

Abstract

Dropout-based regularization methods can be regarded as injecting random noise with pre-defined magnitude to different parts of the neural network during training. It was recently shown that Bayesian dropout procedure not only improves generalization but also leads to extremely sparse neural architectures by automatically setting the individual noise magnitude per weight. However, this sparsity can hardly be used for acceleration since it is unstructured. In the paper, we propose a new Bayesian model that takes into account the computational structure of neural networks and provides structured sparsity, e.g. removes neurons and/or convolutional channels in CNNs. To do this we inject noise to the neurons outputs while keeping the weights unregularized. We establish the probabilistic model with a proper truncated log-uniform prior over the noise and truncated log-normal variational approximation that ensures that the KL-term in the evidence lower bound is computed in closed-form. The model leads to structured sparsity by removing elements with a low SNR from the computation graph and provides significant acceleration on a number of deep neural architectures. The model is very easy to implement as it only corresponds to the addition of one dropout-like layer in computation graph.

1 Introduction

Deep neural networks are a flexible family of models which provides state-of-the-art results in many machine learning problems [16, 11]. However, this flexibility often results in overfitting. A common solution for this problem is regularization. One of the most popular ways of regularization is Binary Dropout [15] that prevents co-adaptation of neurons by randomly dropping them during training. An equally effective alternative is Gaussian Dropout [15] that multiplies the outputs of the neurons by Gaussian random noise. In recent years several Bayesian generalizations of these techniques have been developed, e.g. Variational Dropout [6] and Variational Spike-and-Slab Neural Networks [10]. These techniques provide theoretical justification of different kinds of Dropout and also allow for automatic tuning of dropout rates, which is an important practical result.

Besides overfitting, compression and acceleration of neural networks are other critical issues, especially when memory or computational resources are restricted. Further studies of Variational Dropout show that individual dropout rates for each weight allow to shrink the original network architecture and result in a highly sparse model [12]. General sparsity provides a way of neural network compression, while the time of network evaluation may remain the same, as most modern DNN-oriented software can't work with sparse matrices efficiently. At the same time, it is possible

31st Under Review by Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

to achieve acceleration by enforcing *structured* sparsity in convolutional filters or data tensors. In the simplest case it means removing redundant neurons or convolutional filters instead of separate weights; but more complex patterns can also be considered. This way Group-wise Brain Damage [8] employs group-wise sparsity in convolutional filters, Perforated CNNs [2] drop redundant rows from the intermediate dataframe matrices that are used to compute convolutions, and Structured Sparsity Learning [20] provides a way to remove entire convolutional filters or even layers in residual networks. These methods allow to obtain practical acceleration with little to no modifications of the existing software. In this paper, we propose a tool that is able to induce an arbitrary pattern of structured sparsity on neural network parameters or intermediate data tensors. We propose a dropout-like layer with a parametric multiplicative noise and use stochastic variational inference to tune its parameters in a Bayesian way. We introduce a proper analog of sparsity-inducing log-uniform prior distribution [6, 12] that allows us to formulate a correct probabilistic model and avoid the problems that come from using an improper prior. This way we obtain a novel Bayesian method of regularization of neural networks that results in group sparsity. Our model can be represented as a separate dropout-like layer that allows for a simple and flexible implementation with almost no computational overhead, and can be incorporated into existing neural networks.

Our experiments show that our model leads to high group sparsity level and significant acceleration of convolutional neural networks with negligible accuracy drop. We demonstrate the performance of our method on LeNet and VGG-like architectures using MNIST and CIFAR-10 datasets.

2 Related Work

Deep neural networks are extremely prone to overfitting, and extensive regularization is crucial. The most popular regularization methods are based on injection of multiplicative noise over layer inputs, parameters or activations [15, 18, 6]. Different kinds of multiplicative noise have been used in practice; the most popular choices are Bernoulli and Gaussian distributions. Another type of regularization of deep neural networks is based on reducing the number of parameters. One approach is to use low-rank approximations, e.g. tensor decompositions [13], and the other approach is to induce sparsity, e.g. by pruning [3] or L_1 regularization [20]. Sparsity can also be induced by using the Sparse Bayesian Learning framework with empirical Bayes [17] or with sparsity-inducing priors [12].

High sparsity is one of the key factors for the compression of DNNs [3, 17]. However, in addition to compression it is beneficial to obtain acceleration. Recent papers propose different approaches to acceleration of DNNs, e.g. Spatial Skipped Convolutions [2] and Spatially Adaptive Computation Time [1] that propose different ways to reduce the number of computed convolutions, Binary Networks [14] that achieve speedup by using only 1 bit to store a single weight of a DNN, Low-Rank Expansions [4] that use low-rank filter approximations, and Structured Sparsity Learning [20] that allows to remove separate neurons or filters. As reported in [20] it is possible to obtain acceleration of DNNs by introducing structured sparsity, e.g. by removing whole neurons, filters or layers. However, non-adaptive regularization techniques require tuning of a huge number of hyperparameters that makes it difficult to apply in practice. In this paper we apply the Bayesian learning framework to obtain structured sparsity and focus on acceleration of neural networks.

3 Stochastic Variational Inference

Given a probabilistic model $p(y \mid x, w)$ we want to tune parameters w of the model using training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$. The prior knowledge about parameters w is defined by prior distribution $p(w)$. Using the Bayes rule we obtain $p(w \mid \mathcal{D}) = p(\mathcal{D} \mid w)p(w)/p(\mathcal{D})$. However, computing posterior distribution using the Bayes rule usually involves computation of intractable integrals, so we need to use approximation techniques.

One of the most widely used approximation techniques is *Variational Inference*. In this approach the unknown distribution $p(w \mid \mathcal{D})$ is approximated by a parametric distribution $q_\phi(w)$ by minimization of the Kullback-Leibler divergence $\mathrm{KL}(q_\phi(w) \parallel p(w \mid \mathcal{D}))$. Minimization of the KL divergence is

equivalent to maximization of the *variational lower bound* $\mathcal{L}(\phi)$.

$$\mathcal{L}(\phi) = L_D(\phi) - \mathrm{KL}(q_\phi(w) \,\|\, p(w)), \tag{1}$$

$$\text{where } L_D(\phi) = \sum_{i=1}^{N} \mathbb{E}_{q_\phi(w)} \log p(y_i \,|\, x_i, w) \tag{2}$$

$L_D(\phi)$ is a so-called expected log-likelihood function which is intractable in case of complex probabilistic model $p(y \,|\, x, w)$. Following [6] we use the Reparametrization trick to obtain an unbiased differentiable minibatch-based Monte Carlo estimator of the expected log-likelihood. Here N is the total number of objects, M is the minibatch size, and $f(\phi, \varepsilon)$ provides samples from the approximate posterior $q_\phi(w)$ as a deterministic function of a non-parametric noise $\varepsilon \sim p(\varepsilon)$.

$$L_D(\phi) \simeq L_D^{SGVB}(\phi) = \frac{N}{M} \sum_{k=1}^{M} \log p(y_{i_k} \,|\, x_{i_k}, w_{i_k} = f(\phi, \varepsilon_{i_k})) \tag{3}$$

$$\mathcal{L}(\phi) \simeq \mathcal{L}^{SGVB}(\phi) = L_D^{SGVB}(\phi) - \mathrm{KL}(q_\phi(w) \,\|\, p(w)) \to \max_\phi \tag{4}$$

$$\nabla_\phi L_D(\phi) \simeq \nabla_\phi L_D^{SGVB}(\phi) \tag{5}$$

This way we obtain a procedure of approximate Bayesian inference where we solve optimization problem (4) by stochastic gradient ascent w.r.t. variational parameters ϕ. This procedure can be efficiently applied to Deep Neural Networks and usually the computational overhead is very small, as compared to ordinary DNNs.

4 Group Sparsity with Log-normal Multiplicative Noise

Variational Inference with a sparsity-inducing log-uniform prior over the weights of a neural network is an efficient way to enforce general sparsity on weight matrices [12]. However, it is difficult to apply this approach to explicitly enforce structured sparsity. We introduce a dropout-like layer with a certain kind of multiplicative noise. We also make use of the sparsity-inducing log-uniform prior, but put it over the noise variables rather than weights. By sharing those noise variables we can enforce group-wise sparsity with any form of groups.

4.1 Variational Inference for Group Sparsity Model

We consider a single dropout-like layer with an input vector $x \in \mathbb{R}^I$ that represents one object with I features, and an output vector $y \in \mathbb{R}^I$ of the same size. The input vector x is usually supposed to come from the activations of the preceding layer. The output vector y would then serve as an input vector for the following layer. We follow the general way to build dropout-like layers (6). Each input feature x_i is multiplied by a noise variable θ_i that comes from some distribution $p_{noise}(\theta)$. For example, for binary dropout $p_{noise}(\theta)$ would be a fully factorized Bernoulli distribution with $p_{noise}(\theta_i) = \mathrm{Bernoulli(p)}$, and for Gaussian dropout it would be a fully-factorized Gaussian distribution with $p_{noise}(\theta_i) = \mathcal{N}(1, \alpha)$.

$$y_i = x_i \cdot \theta_i \qquad\qquad \theta \sim p_{noise}(\theta) \tag{6}$$

Note that if we have a minibatch $X^{M \times I}$ of M objects, we would independently sample a separate noise vector θ^m for each object x^m. This would be the case throughout the paper, but for the sake of simplicity we would consider a single object x in all following formulas. Also note that the noise θ is usually only sampled during the training phase. A common approximation during the testing phase is to use the expected value $\mathbb{E}\theta$ instead of sampling θ. All implementation details are provided and discussed in Section 4.5.

We follow a Bayesian treatment of the variable θ, as described in Section 3. In order to obtain a sparse solution, we choose the prior distribution $p(\theta)$ to be a fully-factorized improper log-uniform distribution. We denote this distribution as $\mathrm{LogU}_\infty(\cdot)$ to stress that is has infinite domain. This distribution is known for its sparsification properties and works well in practice for deep neural networks [12].

$$p(\theta) = \prod_{i=1}^{I} p(\theta_i) \qquad p(\theta_i) = \mathrm{LogU}_\infty(\theta_i) \propto \frac{1}{\theta_i} \qquad \theta_i > 0 \tag{7}$$

In order to train the model, i.e. perform variational inference, we need to choose an approximation family q_ϕ for the posterior distribution $p(\theta \mid \mathcal{D}) \approx q_\phi(\theta)$.

$$q_\phi(\theta) = \prod_{i=1}^{I} q(\theta_i \mid \mu_i, \sigma_i) = \prod_{i=1}^{I} \mathrm{LogN}(\theta_i \mid \mu_i, \sigma_i^2) \tag{8}$$

$$\theta_i \sim \mathrm{LogN}(\theta_i \mid \mu_i, \sigma_i^2) \quad \Longleftrightarrow \quad \log \theta_i \sim \mathcal{N}(\log \theta_i \mid \mu_i, \sigma_i^2) \tag{9}$$

A common choice of variational distribution $q(\cdot)$ is a fully-factorized Gaussian distribution. However, for this particular model we choose $q(\theta)$ to be a fully-factorized log-normal distribution (8–9). To make this choice, we were guided by the following reasons:

- The log-uniform distribution is a specific case of the log-normal distribution when the parameter σ goes to infinity and μ remains fixed. Thus we can guarantee that in the case of no data our variational approximation can be made exact. Hence this variational family has no "prior gap".
- We consider a model with multiplicative noise. The scale of this noise corresponds to its shift in the logarithmic space. By establishing the log-uniform prior we set no preferences on different scales of this multiplicative noise. The usual use of a Gaussian as a posterior immediately implies very asymmetric skewed distribution in the logarithmic space. Moreover log-uniform and Gaussian distributions have different supports and that will require establishing two log-uniform distributions for positive and negative noises. In this case Gaussian variational approximation would have quite exotic bi-modal form (one mode in the log-space of positive noises and another one in the log-space of negative noises). On the other hand, the log-normal posterior for the multiplicative noise corresponds to a Gaussian posterior for the additive noise in the logarithmic scale, which is much easier to interpret.
- Log-normal noise is always non-negative both during training and testing phase, therefore it does not change the sign of its input. This is in contrast to Gaussian multiplicative noise $\mathcal{N}(\theta_i \mid 1, \alpha)$ that is a standard choice for Gaussian dropout and its modifications [19, 15, 6]. During the training phase Gaussian noise can take negative values, so the input to the following layer can be of arbitrary sign. However, during the testing phase noise θ is equal to 1, so the input to the following layer is non-negative with many popular non-linearities (e.g. ReLU, sigmoid, softplus). Although Gaussian dropout works well in practice, it is difficult to justify notoriously different input distributions during training and testing phases.
- The log-normal approximate posterior is tractable. Specifically, the KL divergence term $\mathrm{KL}(\mathrm{LogN}(\mu, \sigma^2) \,\|\, \mathrm{LogU}_\infty)$ can be computed analytically.

The final loss function is presented in equation (10) and is essentially the original variational lower bound (4).

$$\mathcal{L}^{SGVB}(\phi) = L_D^{SGVB}(\mu, \sigma, W) - \mathrm{KL}(q(\theta \mid \mu, \sigma) \,\|\, p(\theta)) \to \max_{\mu, \sigma, W}, \tag{10}$$

where μ and σ are the variatianal parameters, and W denotes all other trainable parameters of the neural network, e.g. the weight matrices, the biases, batch normalization parameters, etc.

Note that we can optimize the variational lower bound w.r.t. the parameters μ and σ of the log-normal noise θ. We do not fix the mean of the noise thus making our variational approximation more tight.

4.2 Problems of Variational Inference with Improper Log-Uniform Prior

The log-normal posterior in combination with a log-uniform prior has a number of attractive features. However, the maximization of the variational lower bound with a log-uniform prior and a log-normal posterior is an ill-posed optimization problem. As the log-uniform distribution is an improper prior, the KL-divergence between a log-normal distribution $\log \mathcal{N}(\mu, \sigma^2)$ and a log-uniform distribution $\log \mathcal{U}$ is infinite for any finite value of parameters μ and σ.

$$\mathrm{KL}\left(\mathrm{LogN}(x \mid \mu, \sigma^2) \,\|\, \log \mathcal{U}(x)\right) = C - \log \sigma \qquad\qquad C = +\infty \tag{11}$$

A common way to tackle this problem is to consider the density of the log-uniform distribution to be equal to $\frac{C}{\theta}$ and to treat C as some finite constant. This trick works well for the case of a Gaussian posterior distribution [6, 12]. The KL divergence between a Gaussian posterior and a log-uniform

prior has an infinite gap, but can be calculated up to this infinite constant in a meaningful way [12]. However, for the case of the log-normal posterior the KL divergence is infinite for any finite values of variational parameters, and is equal to zero for a fixed finite μ and infinite σ. As the data-term (3) is bounded for any value of variational parameters, the only global optimum of the variational lower bound is achieved when μ is finite and fixed, and σ goes to infinity. In this case the posterior distribution collapses into the prior distribution and the model fails to extract any information about the data. This effect is wholly caused by the fact that the log-uniform prior is an improper distribution, which makes the whole probabilistic model flawed.

4.3 Variational Inference with Truncated Approximation Family

Due to the improper prior the optimization problem becomes ill-posed. But do we really need to use an improper prior distribution? The most common number format that is used to represent the parameters of a neural network is the *floating-point* format. The *floating-point* format is only able to represent numbers from a limited range. For example, a single-point precision variable can only represent numbers from the range -3.4×10^{38} to $+3.4 \times 10^{38}$, and the smallest possible positive number is equal to 1.2×10^{-38}. All of probability mass of the improper log-uniform prior is concentrated beyond the single-point precision (and essentially any practical floating point precision), not to mention that the actual relevant range of values of neural network parameters is much smaller. It means that in practice this prior is not a good choice for software implementation of neural networks.

We propose to use a truncated log-uniform distribution (12) as a proper analog of the log-uniform distribution. Here $I_{[a,b]}(x)$ denotes the indicator function for the interval $x \in [a, b]$. The posterior distribution should be defined on the same support as the prior distribution, so we also need to use a *truncated log-normal* distribution (12).

$$\mathrm{LogU}_{[\mathrm{a,b}]}(\theta_i) \propto \mathrm{LogU}_{\infty}(\theta_i) \cdot \mathrm{I}_{[\mathrm{a,b}]}(\log \theta_i) \qquad \mathrm{LogN}_{[\mathrm{a,b}]}(\theta_i) \propto \mathrm{LogN}(\theta_i \,|\, \mu_i, \sigma_i^2) \cdot \mathrm{I}_{[\mathrm{a,b}]}(\log \theta_i) \tag{12}$$

Our final model then can be formulated as follows.

$$y_i = x_i \cdot \theta_i \qquad p(\theta_i) = \mathrm{LogU}_{[\mathrm{a,b}]}(\theta_i) \qquad q(\theta_i \,|\, \mu_i, \sigma_i) = \mathrm{LogN}_{[\mathrm{a,b}]}(\theta_i \,|\, \mu_i, \sigma_i^2) \tag{13}$$

Note that all the nice facts about the log-normal posterior distribution from the Section 4.1 are also true for the truncated log-normal posterior. However, now we have a proper probabilistic model and the Stochastic Variational Inference can be preformed correctly. Unlike (11), now the KL divergence term (14–15) can be calculated correctly for all valid values of variational parameters (see Appendix A for details).

$$\mathrm{KL}(q(\theta \,|\, \mu, \sigma) \,\|\, p(\theta)) = \sum_{i=1}^{I} \mathrm{KL}(q(\theta_i \,|\, \mu_i, \sigma_i) \,\|\, p(\theta_i)) \tag{14}$$

$$\mathrm{KL}(q(\theta_i \,|\, \mu_i, \sigma_i) \,\|\, p(\theta_i)) = \log \frac{b-a}{\sqrt{2\pi e \sigma_i^2}} - \log(\Phi(\beta_i) - \Phi(\alpha_i)) - \frac{\alpha_i \phi(\alpha_i) - \beta_i \phi(\beta_i)}{2(\Phi(\beta_i) - \Phi(\alpha_i))}, \tag{15}$$

where $\alpha_i = \frac{a-\mu_i}{\sigma_i}$, $\beta_i = \frac{b-\mu_i}{\sigma_i}$, $\phi(\cdot)$ and $\Phi(\cdot)$ are the density and the CDF of the standard normal distribution.

The reparameterization trick also can still be performed (16) using the inverse CDF of the truncated normal distribution (see Appendix B).

$$\theta_i = \exp(\mu_i + \sigma_i \Phi^{-1}(\Phi(\alpha_i) + (\Phi(\beta_i) - \Phi(\alpha_i)) y_i)), \text{ where } y_i \sim \mathcal{U}(y \,|\, a, b) \tag{16}$$

The final loss and the set of parameters is the same as described in Section 4.1, and the training procedure remains the same.

4.4 Sparsity

Log-uniform prior is known to lead to a sparse solution [12]. In the variational dropout paper authors interpret the parameter α of the multiplicative noise $\mathcal{N}(1, \alpha)$ as a Gaussian dropout rate and use it as a thresholding criterion for weight pruning. Unlike the binary or Gaussian dropout, in the truncated

log-normal model there is no "dropout rate" variable. However, we can use the signal-to-noise ratio $\mathbb{E}\theta/\sqrt{\mathrm{Var}(\theta)}$ (SNR) for thresholding.

$$\mathrm{SNR}(\theta_i) = \frac{(\Phi(\sigma_i - \alpha_i) - \Phi(\sigma_i - \beta_i))/\sqrt{\Phi(\beta_i) - \Phi(\alpha_i)}}{\sqrt{\exp(\sigma_i^2)(\Phi(2\sigma_i - \alpha_i) - \Phi(2\sigma_i - \beta_i)) - (\Phi(\sigma_i - \alpha_i) - \Phi(\sigma_i - \beta_i))^2}} \tag{17}$$

The SNR can be computed analytically and has a simple interpretation. If the SNR is low, the corresponding neuron becomes very noisy and its output no longer contains any useful information. If the SNR is high, it means that the neuron output contains little noise and is important for prediction. Therefore we can remove all neurons or filters with a low SNR and set their output to constant zero.

4.5 Implementation details

We perform a minibatch-based stochastic variational inference for training. The training procedure looks as follows. On each training step we take a minibatch of M objects and feed it into the neural network. Consider a single SBP layer with input $X^{M\times I}$ and output $Y^{M\times I}$. We independently sample a separate noise vector $\theta^m \sim q(\theta)$ for each object x_m and obtain a noise matrix $\theta^{M\times I}$. The output matrix $Y^{M\times I}$ is then obtained by component-wise multiplication of the input matrix and the noise matrix: $y_{mi} = x_{mi} \cdot \theta_i^m$.

To be fully Bayesian, one would also sample and average over different dropout masks θ during testing, i.e. perform Bayesian ensembling. Although this procedure can be used to slightly improve the final accuracy, it is usually avoided. Bayesian ensembling essentially requires sampling of different copies of neural networks, which makes the evaluation K times slower for averaging over K samples. Instead, during the testing phase in most dropout-based techniques the noise variable θ is replaced with its expected value. In this paper we follow the same approach and replace θ_i with its expectation (18) during testing. The derivation of the expectation of the truncated log-normal distribution is presented in Appendix C.

$$\mathbb{E}\theta_i = \frac{\exp(\mu_i + \sigma_i^2/2)}{\Phi(\beta_i) - \Phi(\alpha_i)}\left[\Phi\left(\frac{\sigma_i^2 + \mu_i - a}{\sigma_i}\right) - \Phi\left(\frac{\sigma_i^2 + \mu_i - b}{\sigma_i}\right)\right] \tag{18}$$

Throughout the paper we introduced the SBP dropout layer for the case when input objects are represented as one-dimensional vectors x. When defined like that, it would induce general sparsity on the input vector x. It works as intended for fully-connected layers, as a single input feature corresponds to a single output neuron of a preceding fully-connected layer and a single output neuron of the following layer. However, it is possible to apply the SBP layer in a more generic setting. Firstly, if the input object is represented as a multidimensional tensor X with shape $I_1 \times I_2 \times \cdots \times I_d$, the noise vector θ of length $I = I_1 \times I_2 \times \cdots \times I_d$ can be reshaped into a tensor with the same shape. Then the output tensor Y can be obtained as a component-wise product of the input tensor X and the noise tensor θ. Secondly, the SBP layer can induce any form of structured sparsity on this input tensor X. To do it, one would simply need to use a single random variable θ_i for the group of input features that should be removed simultaneously. For example, consider an input tensor $X^{H\times W\times C}$ that comes from a convolutional layer, H and W being the size of the image, and C being the number of channels. Then, in order to remove redundant filters from the preceding layer (and at the same time redundant channels from the following layer), one need to share the random variables θ in the following way:

$$y_{hwc} = x_{hwc} \cdot \theta_c \qquad\qquad \theta_c \sim \mathrm{LogN}_{[\mathrm{a,b}]}(\theta_c \,|\, \mu_c, \sigma_c^2) \tag{19}$$

Note that now there is one sample $\theta \in \mathbb{R}^C$ for one object $X^{H\times W\times C}$ on each training step. If the signal-to-noise ratio becomes lower than 1 for a component θ_c, that would mean that we can permanently remove the c-th channel of the input tensor, and therefore delete the c-th filter from the preceding layer and the c-th channel from the following layer. This is a general approach that is not limited to reducing the shape of the input tensor. It is possible to obtain any fixed pattern of group-wise sparsity using this technique.

Similarly, the SBP layer can be applied in a DropConnect fashion. One would just need to multiply the weight tensor W by a noise tensor θ of similar shape. The training procedure remains the same. It is still possible to enforce any structured sparsity pattern for the weight tensor W by sharing the random variables as described above.

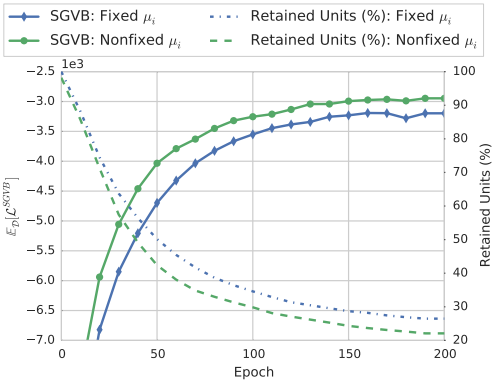


Figure 1: The value of the SGVB for the case of fixed variational parameter $\mu = 0$ (blue line) and for the case when both variational parameters μ and σ are trained (green line)

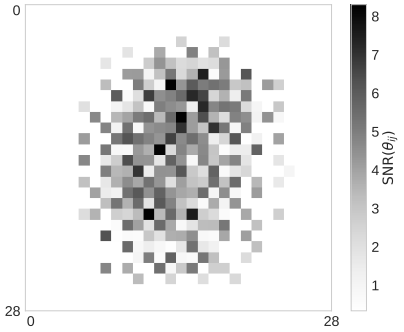


Figure 2: The learned signal-to-noise ratio for image features on the MNIST dataset.

5 Experiments

We perform an evaluation on different supervised classification tasks and with different architectures of neural networks including deep VGG-like architectures with batch normalization layers. For each architecture, we report the number of retained neurons and filters, and obtained acceleration. Our experiments show that *Structured Bayesian Pruning* leads to a high level of structured sparsity in convolutional filters and neurons of DNNs without significant accuracy drop. We also demonstrate that optimization w.r.t. the full set of variational parameters (μ, σ) leads to improving model quality and allows us to perform sparsification in a more efficient way, as compared to tuning of only one free parameter that corresponds to the noise variance. As a nice bonus, we show that *Structured Bayesian Pruning* network does not overfit on randomly labeled data, that is a common weakness of non-bayesian dropout networks. We used Theano [5] and Lasagne for implementation. We will publish the source code with the camera-ready version of the paper.

5.1 Experiment Setup

The truncation parameters a and b are the hyperparameters of our model. As our layer is meant for regularization of the model, we would like our layer not to amplify the input signal and restrict the noise θ to an interval $[0, 1]$. This choice corresponds to the right truncation threshold b set to 0. We find empirically that the left truncation parameter a does not influence the final result much. We use values $a = -20$ and $b = 0$ in all experiments.

We define redundant neurons by the signal-to-noise ratio of the corresponding multiplicative noise θ. See Section 4.4 for more details. By removing all neurons and filters with the $\mathrm{SNR} < 1$ we experience no accuracy drop in all our experiments. SBP dropout layers were put *after* each convolutional layer to remove its filters, and *before* each fully-connected layer to remove its input neurons. As one filter of the last convolutional layer usually corresponds to a group of neurons in the following dense layer, it means that we can remove more input neurons in the first dense layer. Note that it means that we have two consecutive dropout layers between the last convolutional layer and the first fully-connected layer in CNNs, and a dropout layer before the first fully-connected layer in FC networks (see Fig. 2).

5.2 More Flexible Variational Approximation

Usually during automatic training of dropout rates the mean of the noise distribution remains fixed. In the case of our model it is possible to train both mean and variance of the multiplicative noise. By using a more flexible distribution we obtain a tighter variational lower bound and a higher sparsity level. In order to demonstrate this effect, we performed an experiment on MNIST dataset with a fully connected neural network that contains two hidden layers with 1000 neurons each. The results are presented in Fig. 1.

Table 1: Comparison of different structured sparsity inducing techniques (SSL [20], SparseVD [12]) on LeNet-5-Caffe and LeNet-500-300 architectures. Our method provides the highest speed-up with a similar accuracy. We report acceleration that was computed on CPU (Intel Xeon E5-2630), GPU (Tesla K40) and in terms of Floating Point Operations (FLOPs).

Network	Method	Error %	Neurons per Layer %	CPU	GPU	FLOPs
LeNet-500-300	Original	1.54	$784 - 500 - 300 - 10$	$1.00\times$	$1.00\times$	$1.00\times$
	SparseVD	1.57	$537 - 217 - 130 - 10$	$1.19\times$	$1.03\times$	$3.73\times$
	SSL	1.49	$434 - 174 - 78 - 10$	$2.21\times$	$1.04\times$	$6.06\times$
(ours)	StructuredBP	1.55	$245 - 160 - 55 - 10$	$2.33\times$	$1.08\times$	$11.23\times$
LeNet5-Caffe	Original	0.80	$20 - 50 - 800 - 500$	$1.00\times$	$1.00\times$	$1.00\times$
	SparseVD	0.75	$17 - 32 - 329 - 75$	$1.48\times$	$1.41\times$	$2.19\times$
	SSL	1.00	$3 - 12 - 800 - 500$	$5.17\times$	$1.80\times$	$3.90\times$
(ours)	StructuredBP	0.86	$3 - 18 - 284 - 283$	$5.41\times$	$1.91\times$	$10.49\times$

Table 2: Comparison of different structured sparsity inducing techniques (SparseVD [12]) on VGG-like architectures on CIFAR-10 dataset. StructuredBP stands for the original SBP model, and StructuredBPa stands for the SBP model with KL annealing.

k	Method	Error %	Units per Layer	CPU	GPU	FLOPs
1.0	Original	7.2	$64 - 64 - 128 - 128 - 256 - 256 - 256 - 512 - 512 - 512 - 512 - 512 - 512$	$1.00\times$	$1.00\times$	$1.00\times$
	SparseVD	7.2	$64 - 62 - 128 - 126 - 234 - 155 - 31 - 81 - 76 - 9 - 138 - 101 - 413 - 373$	$2.50\times$	$1.69\times$	$2.27\times$
(ours)	StructuredBP	7.5	$64 - 62 - 128 - 126 - 234 - 155 - 31 - 79 - 73 - 9 - 59 - 73 - 56 - 27$	$2.71\times$	$1.74\times$	$2.30\times$
(ours)	StructuredBPa	9.0	$44 - 54 - 92 - 115 - 234 - 155 - 31 - 76 - 55 - 9 - 34 - 35 - 21 - 280$	$3.68\times$	$2.06\times$	$3.16\times$
1.5	Original	6.8	$96 - 96 - 192 - 192 - 384 - 384 - 384 - 768 - 768 - 768 - 768 - 768 - 768 - 768$	$1.00\times$	$1.00\times$	$1.00\times$
	SparseVD	7.0	$96 - 78 - 191 - 146 - 254 - 126 - 27 - 79 - 74 - 9 - 137 - 100 - 416 - 479$	$3.35\times$	$2.16\times$	$3.27\times$
(ours)	StructuredBP	7.2	$96 - 77 - 190 - 146 - 254 - 126 - 26 - 79 - 70 - 9 - 71 - 82 - 79 - 49$	$3.63\times$	$2.17\times$	$3.32\times$
(ours)	StructuredBP	7.8	$77 - 74 - 161 - 146 - 254 - 125 - 26 - 78 - 66 - 9 - 47 - 55 - 54 - 237$	$4.47\times$	$2.47\times$	$3.93\times$

5.3 LeNet5 and Fully-Connected Net on MNIST

We compare our method with other sparsity inducing methods on the MNIST dataset using a fully connected architecture LeNet-500-300 and a convolutional architecture LeNet-5-Caffe. These networks were trained with Adam without any data augmentation. The LeNet-500-300 network was trained from scratch, and the LeNet-5-Caffe[1] network was pretrained with weight decay. An illustration of trained SNR for the image features for the LeNet-500-300[2] network is shown in Fig. 2. The final accuracy, group-wise sparsity levels and speedup for these architectures for different methods are shown in Table 1.

5.4 VGG-like on CIFAR-10

To prove that SBP scales to deep architectures, we apply it to a VGG-like network [21] that was adapted for the CIFAR-10 [7] dataset. The network consists of 13 convolutional and two fully-connected layers, trained with pre-activation batch normalization and Binary Dropout. At the start of the training procedure, we use pre-trained weights for initialization. Results with different scaling of the number of units are presented in Table 2. We present results for two architectures with different scaling coefficient $k \in \{1.0, 1.5\}$. For smaller values of scaling coefficient $k \in \{0.25, 0.5\}$ we obtain less sparse architecture since these networks have small learning capacities. Besides the results for the standard StructuredBP procedure, we also provide the results for StructuredBP with KL annealing. KL annealing procedure leads to higher sparsity level for the first layers, thus, providing more acceleration. Despite the higher error values, we obtain the higher value of variational lower bound during KL annealing, hence, we find another local minimum.

5.5 Random Labels

A recent work shows that Deep Neural Networks have so much capacity that they can easily memorize the data even with random labeling [22]. Binary dropout as well as other standard regularization techniques do not prevent the networks from overfitting in this scenario. However, recently it was shown that Bayesian regularization may help [12]. Following these works, we conducted similar experiments. We used a Lenet5 network on the MNIST dataset and a VGG-like network on CIFAR-10. Although binary dropout does not prevent these networks from overfitting, SBP decides to remove all neurons of the neural network and provides a constant prediction. In other words, in this case SBP chooses the simplest model that achieves the same testing error rate. This is another confirmation that Bayesian regularization is more powerful than other popular regularization techniques.

[1] A modified version of LeNet5 from [9]. Caffe Model specification: https://goo.gl/4yI3dL

[2] Fully Connected Neural Net with 2 hidden layers that contains 500 and 300 neurons respectively.

6 Conclusion

We propose Structured Bayesian Pruning, or SBP, a dropout-like layer that induces multiplicative random noise over the output of the preceding layer. We put a sparsity-inducing prior over the noise variables and tune the noise distribution using stochastic variational inference. SBP layer can induce an arbitrary structured sparsity pattern over its input and provides adaptive regularization. We apply SBP to cut down the number of neurons and filters in convolutional neural networks and report significant practical acceleration with no modification of the existing software implementation of these architectures.

References

[1] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. *arXiv preprint arXiv:1612.02297*, 2016.

[2] Mikhail Figurnov, Aizhan Ibraimova, Dmitry P Vetrov, and Pushmeet Kohli. Perforatedcnns: Acceleration through elimination of redundant convolutions. In *Advances in Neural Information Processing Systems*, pages 947–955, 2016.

[3] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[4] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[5] Bergstra James, Breuleux Olivier, Bastien Frédéric, Lamblin Pascal, and Pascanu Razvan. Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*.

[6] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2575–2583. Curran Associates, Inc., 2015.

[7] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[8] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2554–2564, 2016.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Christos Louizos. Smart regularization of deep architectures. *Master's thesis, University of Amsterdam*, 2015.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[12] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

[13] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

[14] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[15] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[17] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

[18] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[19] Sida I Wang and Christopher D Manning. Fast dropout training. In *ICML (2)*, pages 118–126, 2013.

[20] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[21] Sergey Zagoruyko. 92.45 on cifar-10 in torch, 2015.

[22] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

A KL divergence for the truncated log-normal distribution

We need to introduce some notation to work with the truncated normal distribution. Consider a distribution $\mathrm{t}\mathcal{N}(x \mid \mu, \sigma^2, a, b)$, where μ and σ are the mean and the standard deviation of the corresponding normal distribution before truncation, and a and b are the left and right truncation thresholds respectively. Denote

$$\alpha = \frac{a-\mu}{\sigma} \qquad \beta = \frac{b-\mu}{\sigma} \qquad Z = \Phi(\beta) - \Phi(\alpha)$$

Now we can calculate the KL divergence between a truncated log-normal distribution $q(\theta^i)$ and a log-uniform distribution $p(\theta^i)$ with a bounded support $\theta^i \in [e^a, e^b]$.

$$\begin{aligned}
&\mathrm{KL}(q(\theta^i) \,\|\, p(\theta^i)) = \mathrm{KL}(q(\log\theta^i) \,\|\, p(\log\theta^i)) = \\
&= \mathrm{KL}(\mathrm{t}\mathcal{N}(x \mid \mu, \sigma^2, a, b) \,\|\, \mathcal{U}(x \mid a, b)) = \int_a^b \mathrm{t}\mathcal{N}(x \mid \mu, \sigma^2, a, b) \log \frac{\mathrm{t}\mathcal{N}(x \mid \mu, \sigma^2, a, b)}{\mathcal{U}(x \mid a, b)} dx = \\
&= -\mathcal{H}(\mathrm{t}\mathcal{N}(x \mid \mu, \sigma^2, a, b)) + \log(b-a)
\end{aligned}$$

Entropy for the truncated normal distribution is

$$\begin{aligned}
&\mathcal{H}(\mathrm{t}\mathcal{N}(x \mid \mu, \sigma^2, a, b)) = \log(\sqrt{2\pi e}\sigma Z) + \frac{\alpha\phi(\alpha) - \beta\phi(\beta)}{2Z} \\
&\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right) \\
&\Phi(x) = \frac{1}{2}(1 + \mathrm{erf}(x/\sqrt{2}))
\end{aligned}$$

Finally, we obtain

$$\begin{aligned}
&\mathrm{KL}(q(\theta_i \mid \mu_i, \sigma_i) \,\|\, p(\theta_i)) = \\
&= \log(b-a) - \log(\sqrt{2\pi e}\sigma_i) - \log(\Phi(\beta_i) - \Phi(\alpha_i)) - \frac{\alpha_i\phi(\alpha_i) - \beta_i\phi(\beta_i)}{2(\Phi(\beta_i) - \Phi(\alpha_i))}
\end{aligned}$$

B Sampling from the truncated log-normal distribution

To sample θ^i from the truncated log-normal distribution, we first sample $\log\theta^i \sim \mathrm{t}\mathcal{N}(\log\theta^i \mid \mu_i, \sigma_i^2, a, b)$ and then take the exponent. In order to sample from the truncated normal distribution we use the inverse cumulative density function. The CDF for the truncated normal distribution is

$$F(x) = \frac{\Phi(\frac{x-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})}{Z} = y$$

Hence inverse CDF can be written as

$$F^{-1}(y) = \mu + \sigma\Phi^{-1}\Big(\Phi(\alpha) + Zy\Big) = x$$

Sampling y from $\mathcal{U}[0,1]$ one can obtain x samples from truncated normal distribution. The final expression for the reparameterization trick for $\theta \sim \log \mathrm{t}\mathcal{N}(\theta \mid \mu, \sigma^2, a, b)$ looks like this:

$$\theta = \exp\left\{\mu + \sigma\Phi^{-1}\Big(\Phi(\alpha) + Zy\Big)\right\}, \text{ where } y \sim \mathcal{U}(y \mid a, b)$$

C Mean of the truncated log-normal ditribution

Let's derive the expected value of θ. In order to this, let's first find the PDF of the truncated log-normal distribution.

$$\begin{aligned}
&\mathrm{t}\mathcal{N}(x \mid \mu, \sigma^2, a, b) = \frac{1}{Z\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right),\ x \in [a, b], \\
&\mathrm{t}\mathcal{N}(\log x \mid \mu, \sigma^2, a, b) d(\log x) = \mathrm{t}\mathcal{N}(\log x \mid \mu, \sigma^2, a, b)\frac{dx}{x} = \mathrm{LogN}_{[\mathrm{a,b}]}(x \mid \mu, \sigma^2)dx \\
&\mathrm{LogN}_{[\mathrm{a,b}]}(x \mid \mu, \sigma^2) = \frac{1}{Zx\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(\log x - \mu)^2\right),\ \log x \in [a, b],\ x > 0
\end{aligned}$$

The obtained PDF is very similar to the log-normal distribution PDF. Hence,

$$\mathbb{E}\Big[x \sim \mathrm{LogN}_{[\mathrm{a,b}]}(x \,|\, \mu, \sigma^2)\Big] = \frac{1}{Z}\int_{e^a}^{e^b} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx =$$
$$= \frac{1}{Z}\Bigg[\exp(\mu + \sigma^2/2) - \int_0^{e^a} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx -$$
$$- \int_{e^b}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx\Bigg]$$

Now we need the formula for

$$p(a) := \int_a^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx$$
$$t = \frac{\log x - \mu}{\sigma}, \quad x = e^{t\sigma+\mu}, \quad dx = e^{t\sigma+\mu}\sigma dt$$

$$p(a) = \int_{\frac{\log a-\mu}{\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{1}{2}t^2 + t\sigma + \mu\Big)dt = \int_{\frac{\log a-\mu}{\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{1}{2}(t^2 - 2t\sigma + \sigma^2) + \frac{\sigma^2}{2} + \mu\Big)dt =$$
$$= \exp(\sigma^2/2 + \mu)\int_{\frac{\log a-\mu}{\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{1}{2}(t-\sigma)^2\Big)dt = \exp(\sigma^2/2 + \mu)\int_{\frac{\log a-\mu}{\sigma}-\sigma}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{1}{2}t^2\Big)dt$$
$$p(a) = \exp(\sigma^2/2 + \mu)\Big(1 - \Phi\Big(\frac{\log a - \mu}{\sigma} - \sigma\Big)\Big) = \exp(\sigma^2/2 + \mu)\Phi\Big(\frac{\sigma^2 + \mu - \log a}{\sigma}\Big)$$

Using the derived formula, we finally obtain the expectation $\mathbb{E}x$

$$\mathbb{E}\Big[x \sim \mathrm{LogN}_{[\mathrm{a,b}]}(x \,|\, \mu, \sigma^2)\Big] =$$
$$= \frac{1}{Z}\Big[\exp(\mu + \sigma^2/2) - [\exp(\mu + \sigma^2/2) - p(e^a)] - p(e^b)\Big]$$
$$= \frac{\exp(\mu + \sigma^2/2)}{Z}\Big[\Phi\Big(\frac{\sigma^2 + \mu - a}{\sigma}\Big) - \Phi\Big(\frac{\sigma^2 + \mu - b}{\sigma}\Big)\Big]$$

D Signal-to-noise ratio of the truncated log-normal distribution

It is useful to calculate the signal-to-noise ratio $\mathbb{E}x/\sqrt{\mathrm{Var}(x)}$ for the truncated log-normal distribution in order to investigate the sparsity of the resulting layer. We need the variance $\mathrm{Var}(x)$ to calculate it.

$$\mathrm{LogN}_{[\mathrm{a,b}]}(x \,|\, \mu, \sigma^2) = \frac{1}{Zx\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big), \ \log x \in [a, b], \ x > 0$$

$$\mathrm{Var}\Big[x \sim \mathrm{LogN}_{[\mathrm{a,b}]}(x \,|\, \mu, \sigma^2)\Big] = \frac{1}{Z}\int_{e^a}^{e^b} \frac{(x - \mathbb{E}x)^2}{x\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx =$$
$$= \frac{1}{Z}\Big[\int_{e^a}^{\infty} \frac{(x - \mathbb{E}x)^2}{x\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx - \int_{e^b}^{\infty} \frac{(x - \mathbb{E}x)^2}{x\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx\Big]$$

So we have to calculate

$$p'(a) := \int_a^{\infty} \frac{(x - \mathbb{E}x)^2}{x\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx$$

$$p'(a) = \int_a^{\infty} \frac{x}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx$$
$$- 2\mathbb{E}x\int_a^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx$$
$$+ (\mathbb{E}x)^2\int_a^{\infty} \frac{1}{x\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big)dx$$

We already have the expression for $p(a)$

$$p(a) = \int_a^\infty \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big) dx = \exp(\sigma^2/2 + \mu)\Phi\Big(\sigma - \frac{\log a - \mu}{\sigma}\Big)$$

For the rest two summands we introduce a new variable t.

$$t = \frac{\log x - \mu}{\sigma}, \quad x = e^{t\sigma+\mu}, \quad dx = e^{t\sigma+\mu}\sigma dt$$

Next, we use t in a variable substitution.

$$\int_a^\infty \frac{x}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big) dx = \int_{\frac{\log a-\mu}{\sigma}}^\infty \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{1}{2}t^2 + 2t\sigma + 2\mu\Big) dt =$$
$$= \int_{\frac{\log a-\mu}{\sigma}}^\infty \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{1}{2}(t^2 - 4t\sigma + 4\sigma^2) + 2\mu + 2\sigma^2\Big) dt =$$
$$= \exp(2\mu + 2\sigma^2) \int_{\frac{\log a-\mu}{\sigma}}^\infty \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{1}{2}(t - 2\sigma)^2\Big) dt = \exp(2\mu + 2\sigma^2) \int_{\frac{\log a-\mu}{\sigma} - 2\sigma}^\infty \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{1}{2}t^2\Big) dt =$$
$$= \exp(2\mu + 2\sigma^2)\Phi\Big(2\sigma - \frac{\log a - \mu}{\sigma}\Big)$$

$$\int_a^\infty \frac{1}{x\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big) dx = \int_{\frac{\log a-\mu}{\sigma}}^\infty \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{1}{2}t^2\Big) dt = \Phi\Big(-\frac{\log a - \mu}{\sigma}\Big)$$

Using the expression for $\mathbb{E}x$ we obtain

$$p'(a) = \exp(2\mu + 2\sigma^2)\Phi\Big(2\sigma - \frac{\log a - \mu}{\sigma}\Big) - 2\exp(\sigma^2/2 + \mu)\Phi\Big(\sigma - \frac{\log a - \mu}{\sigma}\Big)\mathbb{E}x + (\mathbb{E}x)^2\Phi\Big(-\frac{\log a - \mu}{\sigma}\Big)$$

$$\begin{aligned}
\mathrm{Var}\Big[x \sim \mathrm{LogN}_{[\mathrm{a,b}]}(x \mid \mu, \sigma^2)\Big] =& \frac{1}{Z}\int_{e^a}^{e^b} \frac{(x - \mathbb{E}x)^2}{x\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(\log x - \mu)^2\Big) dx = \\
=& \frac{\exp(2\mu + \sigma^2)}{Z}\Big[\exp(\sigma^2)(\Phi(2\sigma - \alpha) - \Phi(2\sigma - \beta)) \\
& - \frac{2}{Z}(\Phi(\sigma - \alpha) - \Phi(\sigma - \beta))^2 + \frac{1}{Z^2}(\Phi(\sigma - \alpha) - \Phi(\sigma - \beta))^2(\Phi(-\alpha) - \Phi(-\beta))\Big] \\
=& \frac{\exp(2\mu + \sigma^2)}{Z}\Big[\exp(\sigma^2)(\Phi(2\sigma - \alpha) - \Phi(2\sigma - \beta)) - \frac{1}{Z}(\Phi(\sigma - \alpha) - \Phi(\sigma - \beta))^2\Big]
\end{aligned}$$

Finally, we obtain the signal-to-noise ratio

$$\mathrm{SNR}(x) = \frac{\mathbb{E}x}{\sqrt{\mathrm{Var}(x)}} = \frac{\Big[\Phi(\sigma - \alpha) - \Phi(\sigma - \beta)\Big]}{\sqrt{Z\exp(\sigma^2)(\Phi(2\sigma - \alpha) - \Phi(2\sigma - \beta)) - (\Phi(\sigma - \alpha) - \Phi(\sigma - \beta))^2}}$$

E Stable Computation of Statistics

Straightforward computation of the SNR and the mean of the truncated log-normal distribution can lead to indeterminate values like $0 \cdot \infty$ when the values of σ are high. In order to make our calculations stable we use the scaled complementary error function $\mathrm{erfcx}(x) = \exp(x^2)\mathrm{erfc}(x)$. Given the equation

$$\Phi(a) - \Phi(b) = \frac{1}{2}\Big[\mathrm{erf}\Big(\frac{a}{\sqrt{2}}\Big) - \mathrm{erf}\Big(\frac{b}{\sqrt{2}}\Big)\Big] = \frac{1}{2}\Big[\mathrm{erfcx}\Big(\frac{b}{\sqrt{2}}\Big)\exp\Big(-\frac{b^2}{2}\Big) - \mathrm{erfcx}\Big(\frac{a}{\sqrt{2}}\Big)\exp\Big(-\frac{a^2}{2}\Big)\Big],$$

we can rewrite equations (17), (18) in the following form

$$\mathbb{E}\theta = \exp(\mu)\frac{1}{2Z}\left[\operatorname{erfcx}\left(\frac{\sigma-\beta}{\sqrt{2}}\right)\exp\left(b-\mu-\frac{\beta^2}{2}\right) - \operatorname{erfcx}\left(\frac{\sigma-\alpha}{\sqrt{2}}\right)\exp\left(a-\mu-\frac{\alpha^2}{2}\right)\right]$$

$$\begin{aligned}
\mathrm{SNR}(\theta) =& \frac{1}{\sqrt{D}}\left[\operatorname{erfcx}\left(\frac{\sigma-\beta}{\sqrt{2}}\right)\exp\left(b-\mu-\frac{\beta^2}{2}\right) - \operatorname{erfcx}\left(\frac{\sigma-\alpha}{\sqrt{2}}\right)\exp\left(a-\mu-\frac{\alpha^2}{2}\right)\right], \\
D =& 2Z\left[\operatorname{erfcx}\left(\frac{2\sigma-\beta}{\sqrt{2}}\right)\exp\left(2(b-\mu)-\frac{\beta^2}{2}\right) - \operatorname{erfcx}\left(\frac{2\sigma-\alpha}{\sqrt{2}}\right)\exp\left(2(a-\mu)-\frac{\alpha^2}{2}\right)\right] \\
& - \left[\operatorname{erfcx}\left(\frac{\sigma-\beta}{\sqrt{2}}\right)\exp\left(b-\mu-\frac{\beta^2}{2}\right) - \operatorname{erfcx}\left(\frac{\sigma-\alpha}{\sqrt{2}}\right)\exp\left(a-\mu-\frac{\alpha^2}{2}\right)\right]
\end{aligned}$$