Правительство Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

«Национальный исследовательский университет

«Высшая школа экономики»

Факультет компьютерных наук

Основная образовательная программа

Прикладная математика и информатика

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему

Исследование байесовских методов регуляризации нейронных сетей

Выполнила студентка группы БПМИ141, 4-го курса,

Кириченко Полина Егоровна

Научный руководитель:

к. ф.-м. н., профессор-исследователь,

Ветров Дмитрий Петрович

Консультанты:

старший преподаватель,

Лобачева Екатерина Максимовна

Москва 2018

Аннотация

Глубокие нейронные сети достигают высокого качества во многих задачах машинного обучения. Однако из-за того, что в этих моделях много параметров, они склонны к переобучению – запоминанию обучающей выборки и настройке на шумовые признаки, что ухудшает обобщающую способность модели. Для того, чтобы предотвратить переобучение, необходимо регуляризовывать модели. На практике часто используется метод регуляризации dropout – бинарный или гауссовский шум, который накладывается на активации нейронов или веса модели. В данной работе изучается новый подход к регуляризации нейронных сетей – внесение шума на направление и норму векторов весов вместо независимого покоординатного шума. Рассматривается несколько способов внесения шума на направление векторов, и предлагается вероятностная модель, в которой с помощью вариционного вывода можно автоматически настраивать параметры приближенного апостериорного распределения на веса. Использование специальных априорных распределений может дополнительно приводить к структурному разреживанию в модели и сжатию нейронных сетей.

Abstract

Deep neural networks have shown state-of-the-art performance in many machine learning tasks. However, such models with a large number of parameters are prone to overfitting, and this problem must be addressed for obtaining good generalization ability. Commonly used regularization techniques to tackle this issue are binary dropout or its fast approximation Gaussian dropout. In this work, we study a novel approach for neural network regularization via injecting noise on weight vector magnitude and direction instead of independently perturbing individual scalar weights. We consider several direction noise distributions and, further, propose a probabilistic model where variational inference can be applied for automatic hyperparameter tuning. Incorporating appropriate prior distributions in such a model can also potentially lead to structural sparsity and model compression.

Index terms — machine learning, neural networks, regularization, Bayesian machine learning.

Contents

1 Introduction

Recent studies have shown that deep neural networks often have excessive architectures, are over-parametrized and tend to tremendously overfit: they can simply remember training data rather than recognize patterns and learn to generalize [1]. Various regularization techniques, like binary [2] or Gaussian dropout [3], have been proposed for alleviating this problem. Such techniques usually comprise injecting some kind of noise in a model during training (either on neuron's activations or weights). They were also studied from a probabilistic perspective.

A Bayesian extension for Gaussian dropout – variational dropout [4, 5], was proven not only to be a flexible regularization technique with adaptive dropout rate but also to provide automatic relevance determination for network weights and sparsify models. Obtaining sparse models is extremely crucial if we want to use neural networks on mobile devices with limited computational resources. Thus, regularization and model compression are both important topics which are currently actively studied in machine learning research community.

Regularization techniques have been studied so far when applied to neuron activations as in standard binary or Gaussian dropout, or individual weights as in DropConnect [6] or Gaussian noise on weights. The aim of this work is to study a different way of noise injection into model weights: direction and magnitude dropout. We propose to perturb weight vector direction and norm as opposite to inserting noise to each weight vector coordinate independently. This approach is also consistent with Weight Normalization parametrization [7] where magnitude and direction of each weight vector are decoupled.

To the best of our knowledge, it is the first attempt to regularize network weights in the form of direction and magnitude noise. We provide an intuition for a potential benefit of such type of noise and propose a probabilistic model which is trained using variational inference framework. Moreover, via incorporating appropriate prior distributions, this could lead not only to a better generalization, but also sparsity which is a desirable property for modern deep neural network architectures.

2 Related Work

2.1 Binary Dropout

Binary dropout is a simple and popular regularization technique. Neural networks have high risk of memorizing input data due to a large number of parameters. In order to prohibit this kind of behaviour, during processing current mini-batch of data, a subset of neurons is randomly dropped from the model. This way, network will not rely too much on individual neurons while making prediction. From the other side, it will learn an ensemble of all networks that can be formed by removing corresponding neurons. From bias-variance trade-off point of view, it is similar to bagging procedure which reduces the variance of model while non-increasing the bias.

The procedure is the following. Let x_i be a vector of activations of layer i. We sample Bernoulli random vector with parameter p and apply this mask to the activation vector:

$$z_i \sim \text{Bernoulli}(p) \tag{1}$$

$$\widetilde{x_i} = x_i \odot z_i \tag{2}$$

$$x_{i+1} = f(W_i \, \widetilde{x_i} + b_i) \tag{3}$$

2.2 Gaussian Dropout

When binary dropout is used in a network, each neuron computes a weighted sum of Bernoulli random variables:

$$Y(z) = \sum_{i=1}^{n} w_i x_i z_i \qquad z_i \sim \text{Bernoulli}(p) \tag{4}$$

where $Y(z)$ is the pre-activation of the neuron. In [3], it was shown that $Y(z)$ can be well approximated by a normal distribution even when the size of the layer is relatively small. By the Lyapunov's central limit theorem, $Y(z)$ tends to a normal distribution. Its moments are:

$$\mathbb{E}_z Y(z) = \sum_{i=1}^{n} w_i x_i p_i \tag{5}$$

$$\text{Var}_z[Y(z)] = \sum_{i=1}^{n} p_i(1-p_i)(w_i x_i)^2 \tag{6}$$

Thus, Gaussian dropout, or injecting Gaussian noise $\mathcal{N}(1, \alpha)$, corresponds to binary dropout with probability $1-p$ of dropping neuron where $\alpha = (1-p)/p$. This fast approximation to binary dropout shows a similar regularization effect. It was also shown that when the noise is transmitted from activations to weights, such regularization is closely related to Bayesian inference.

2.3 Bayesian Neural Networks

In Bayesian machine learning, instead of finding maximum likelihood point estimate, our goal is to estimate the posterior distribution of model parameters w conditioned on a training dataset $\mathcal{D}$. We consider distribution $p(w)$ representing our a priori beliefs. Then, after training data $\mathcal{D} = (x_i, \, y_i)_{i=1}^{N}$ arrives, we use the Bayes Rule to obtain a posteriori distribution over parameters:

$$p(w \,|\, \mathcal{D}) = \frac{p(\mathcal{D} \,|\, w) p(w)}{p(\mathcal{D})} \tag{7}$$

This process is called Bayesian inference. In complex models like neural networks, $p(\mathcal{D})$ often cannot be computed analytically, and it is hard to numerically estimate it because of the high dimensionality of parameter space. Possible solutions include either simplification of the model, for example, considering diagonal multivatiate Gaussian distribution over vector of all parameters, or approximate inference, which will be discussed in the next section.

Bayesian view on neural networks provides a range of benefits. Distribution over parameters enables uncertainty estimation of the model over its parameters and predictions. This may be crucial in medical applications where risks calculations are important. Also, we obtain a way to ensemble an infinite number of neural networks by sampling its parameters which increases the model's performance.

2.4 Stochastic Variational Inference

Variational inference is a technique to find an approximation $q_\phi(w)$ of true posterior distribution $p(w \mid \mathcal{D})$ within some parametric family. In order to find such approximation (optimal distribution parameters ϕ), variational lower bound on evidence is introduced and maximized:

$$\mathcal{L}(\phi) = L_{\mathcal{D}}(\phi) - \mathrm{KL}(q_\phi(w) \,||\, p(w|\mathcal{D})) \to \max_\phi \tag{8}$$

$$L_{\mathcal{D}}(\phi) = \sum_{i=1}^{N} \mathbb{E}_{q_\phi(w)} \log p(y_i \mid x_i, w) \tag{9}$$

The second term in the lower bound called Kullback-Leibler divergence is a common measure of distributions dissimilarity. It has a useful property: it is non-negative for any q and p, and equals zero when distributions match almost everywhere. Thus, the lower bound is tighter, when distribution $p(w \mid \mathcal{D})$ and its approximation $q_\phi(w)$ are closer.

Next, due to a large number of parameters in neural networks, we need a lot of gradient updates. However, with limited computational time, it is impossible to compute gradient on the whole dataset sufficiently many times. Thus, neural networks are usually optimized via stochastic mini-batch gradient descend or its modifications, where stochastic gradient is computed on a subset of size M of training data, $M \ll N$.

With the same motivation, we are interested in stochastic version of variational inference for neural networks:

$$L_D(\phi) \simeq L_D^{SGVB}(\phi) = \frac{N}{M} \sum_{k=1}^{M} \log p(y_{i_k} \mid x_{i_k}, w_{i_k} = f(\phi, \varepsilon_{i_k})) \tag{10}$$

$$\mathcal{L}(\phi) \simeq \mathcal{L}^{SGVB}(\phi) = L_D^{SGVB}(\phi) - \mathrm{KL}(q_\phi(w) \,||\, p(w)) \tag{11}$$

$$\nabla_\phi L_D(\phi) \simeq \nabla_\phi L_D^{SGVB}(\phi) \tag{12}$$

We have obtained an unbiased estimator of gradient which can be used to optimize the variational lower bound. However, in practice, this estimator have impermissibly high variance. In [8], an alternative way to generate samples from distribution $q_\phi(w)$ was proposed to reduce this variance (which is though only applicable to a limited number of family of distributions). Instead of direct sampling $w \sim q_\phi(w)$, we can sample noise ϵ from simple non-parametric distribution and express w as a deterministic function of noise ϵ and parameters ϕ: $w = g(\epsilon, \phi)$. For instance, to obtain sample $w \sim \mathcal{N}(\mu, \sigma^2)$ we can sample $\epsilon \sim \mathcal{N}(0, 1)$ and get $w = \mu + \sigma \cdot \epsilon$. This technique (called reparametrization trick) yields lower variance of gradient estimator of the variational lower bound. A further discussion on the gradient variance will be presented in section 4.2.

2.5 Variational Dropout

The relation of Gaussian dropout and approximate Bayesian inference established in [3] was further developed in [4]. It was shown that training a network with Gaussian dropout (with fixed variances α) is equivalent to putting log-uniform prior on weights w and optimizing the evidence lower bound. Furthermore, as proposed in [4], procedure can be extended to tune individual dropout

rates α_{ij} via variational inference. Note that $\alpha_{ij} \to \infty$ corresponds to a binary dropout rate that approaches $p = 1$. In [4], authors show that with proper approximation of KL-divergence in this model, which can be computed only up to an additive (infinitely large) constant, variational procedure tends to set large values to the most of individual dropout rates α_{ij}. It forces the network to set corresponding weights θ_{ij} close to zero so that the variance of posterior approximation $\alpha\theta^2$ goes to zero as well which results in model sparsification:

$$\begin{gathered} \theta_{ij} \to 0, \quad \alpha_{ij}\theta_{ij}^2 \to 0 \\ \Downarrow \\ q(w_{ij} \,|\, \theta_{ij}, \alpha_{ij}) \to \mathcal{N}(w_{ij} \,|\, 0, 0) = \delta(0) \end{gathered} \tag{13}$$

2.6 Weight Normalization

Another concern for training neural networks is optimization stability. One of the ways to address this problem is to use Weight Normalization [7] which shows to improve the learning progress and speed up the convergence of stochastic gradient descent. The weight vector's new parametrization is the following:

$$w = g \cdot \frac{v}{\|v\|} \tag{14}$$

This approach is beneficial compared to another popular solution – Batch Normalization [9] as it can be also applied to other neural network architectures such as recurrent neural networks. The proposed way to normalize weight matrix is to normalize weight vectors for output neurons, but the dimension for normalization is not restricted.

3 Direction and magnitude noise

In this section, we provide a motivation for applying direction and magnitude noise for weight vectors instead of commonly used independent coordinate-wise noise. Then, we describe several ways for sampling direction noise and discuss the difference between magnitude noise and standard noise injection methods.

3.1 Motivation

In a standard multi-layer fully-connected neural network with Rectified Linear Unit nonlinearity $f(y) = \max(0, y)$, consider a single neuron in a hidden layer: its pre-activation value is $y = w^T x$ where w is a weight vector for all incoming neuron's connections and x is a vector of activations of the preceding layer (bias is omitted for simplicity). A neuron gets activated if y exceeds the threshold, namely, zero. In fact, the pre-activation value is $y = w^T x = \|x\|\|w\| \cos(x, w)$ where $\cos(x, w)$ is a cosine of an angle between vectors x and w.

The two terms $\|w\|$ and $\cos(x, w)$ can be well interpreted: the norm $\|w\|$ can be treated as a significance of the neuron, and $\cos(x, w)$ – as a strength of a response to a particular incoming signal x. The higher the magnitude $\|w\|$ is, the more will be the activation value to any incoming signal; the more co-directed the signal and the weight vector are, the higher will be the activation to this particular signal.

The intuition behind the direction of weight vector can be extended further. A single neuron can be seen as a small individual model which seeks informative incoming signal directions and tunes its weights w accordingly when the network is being trained. For a fixed signal magnitude m, the neuron's response is the strongest for a signal which is co-directed with w, so the neuron is seeking for this most important signal direction. However, incoming signal is noisy and it is reasonable to expect that the neuron would be responsive to various directions and not just the single one. Consider a set of all signals which have the same magnitude m and form a fixed angle α with w: $X_{m,\alpha} = \{x : \|x\| = m,\ \cos(x, w) = \alpha\}$. Note that for any point x on a hypersphere $X_{m,\alpha}$, the activation of the neuron would be the same. Thus, the neuron searches for some most relevant signal hyperspheres $X_{m,\alpha}$ on which it would produce high activation values.

Noise injection into weight's direction would directly influence the search of these hyperspheres. Moreover, in a probabilistic setting where we would want to find posterior distribution approximation over w, if an approximation family is flexible enough to describe a hypersphere, it could induce the distribution over the informative signal hyperspheres to be captured by w.

In addition to the points mentioned above, the influence of $\|x\|$ can sometimes be assumed to be negligible if the normalization technique like Layer Normalization [10] is used: it would lead to somewhat similar signal magnitude $\|x\|$, and, thus, only $\|w\|$ and $\cos(w, x)$ would mostly determine the pre-activation value.

Our hypothesis is that it can be beneficial to take these observations into account and apply noise separately to magnitude $\|w\|$ and direction of w for regularization.

In Weight Normalization parametrization [7], norm and direction are decoupled to be different parameters: $w = gv/\|v\|$ where g is a scalar magnitude (which is actually not limited to be positive), and v is a direction parameter. This network parametrization allows to inject noise into such network parameters in a straightforward manner.

For convolutional layers, the reasoning is similar. Instead of a vector of activations x, there is a tensor of activations x of size ($input\_channels$, n, n). Instead of a single output neuron, we consider a single output channel for which a feature map is obtained via applying a convolution (actually, a cross-correlation) w of size ($input\_channels$, k, k) to x. The value in every position in the feature map is essentially a dot product of w and a subtensor of x of size ($input\_channels$, k, k) so the aforementioned arguments remain the same.

3.2 Direction noise

In Weight Normalization where $w = gv/\|v\|$, the direction parameter is v. $v_{norm} = v/\|v\|$ is essentially a n-dimensional unit vector, thus, the domain for direction noise distribution is a n-dimensional unit sphere S_n. A popular distribution on S_n is the von Mises-Fisher distribution which is an analogues of the Normal distribution on a hypersphere. However, due to a high computational cost of its normalization constant, rejection sampling is usually used to draw samples from this distribution. This sampling technique does not scale to deep neural networks. Thus, it is necessary to find simpler distributions on S_n with more efficient sampling procedures. In this work, we investigated the three following methods to sample points from a unit hypersphere which induce different distributions on S_n.

3.2.1 Projected Gaussian distribution

Draw a sample ε from an isotropic normal noise $\mathcal{N}(0, \alpha I)$, add it to the unit vector v_{norm}, and then normalize back to the unit sphere.

$$\varepsilon \sim \mathcal{N}(0, \alpha I) \tag{15}$$

$$\hat{v}_{norm} = \frac{v_{norm} + \varepsilon}{\|v_{norm} + \varepsilon\|} \tag{16}$$

On figure 1, different resulting distributions on S_3 are shown produced by this sampling procedure: for small values α samples are concentrated around the mean of the Gaussian and, with increasing α, the distribution converges to the uniform. However, in high dimensional space, the properties of this distribution become non-intuitive which is well mathematically explained and was observed experimentally (see section 5.1.1 and appendix D for details).

Projected non-isotropic noise $\mathcal{N}(0, \bar{\alpha} I)$ with vector $\bar{\alpha}$ having different values would be a more rich family of distributions (examples of resulting distributions in $\mathbb{R}^3$ are shown on figure 7 of appendix) but tuning $\bar{\alpha}$ manually with grid search would be intractable. Using this distribution would become tractable in the case of auto-tuning distribution parameters in variational setting which will be discussed further in section 4.

Note that when scalar variance α is large enough and the distribution on S_n approaches uniform one, perturbed vector $\hat{v}_{norm}$ would turn to the direction opposite to v_{norm} with high probability, and such noise might be too harsh for the model and lead to unstable training. The next sampling method addresses this observation.

3.2.2 Projected Gaussian distribution from tangent subspace

Draw a sample ε from an isotropic normal noise $\mathcal{N}(0, \alpha I)$, project it to the tangent space attached to v_{norm}, then add this noise to v_{norm} and project it back to the unit sphere.

$$\varepsilon \sim \mathcal{N}(0, \alpha I) \tag{17}$$

$$\varepsilon_{tang} = \left(I - v_{norm} v_{norm}^T\right) \varepsilon = \varepsilon - v_{norm}(v_{norm}^T \varepsilon) \tag{18}$$

$$\hat{v}_{norm} = \frac{v_{norm} + \varepsilon_{tang}}{\|v_{norm} + \varepsilon_{tang}\|} = \frac{v_{norm} + \varepsilon_{tang}}{\sqrt{1 + \|\varepsilon_{tang}\|^2}} \tag{19}$$

The last equation holds because v_{norm} and ε_{tang} are perpendicular. ε_{tang} has a normal distribution with variance αI in a subspace of dimension $n - 1$ which is tangent to v_{norm}. This way the angle between v_{norm} and $\hat{v}_{norm}$ is at most $\pi/2$ and, thus, we limit perturbations to the semi-sphere.

The peculiarities of such distribution in high dimensional spaces still hold true so the following sampling method will be also studied.

3.2.3 Rotation in a randomly chosen plane

Draw a random point ϕ from S_{n-1} in a space tangent to v_{norm} from some distribution (for example, uniform) – this point defines a one-dimensional plane in which v_{norm} would be rotated. Then an angle is sampled from some one-dimensional distribution on $[-\pi/2, \pi/2]$ or $[0, \pi/2]$ (like truncated normal or stretched symmetric Beta distribution). This procedure is illustrated on the figure 2.

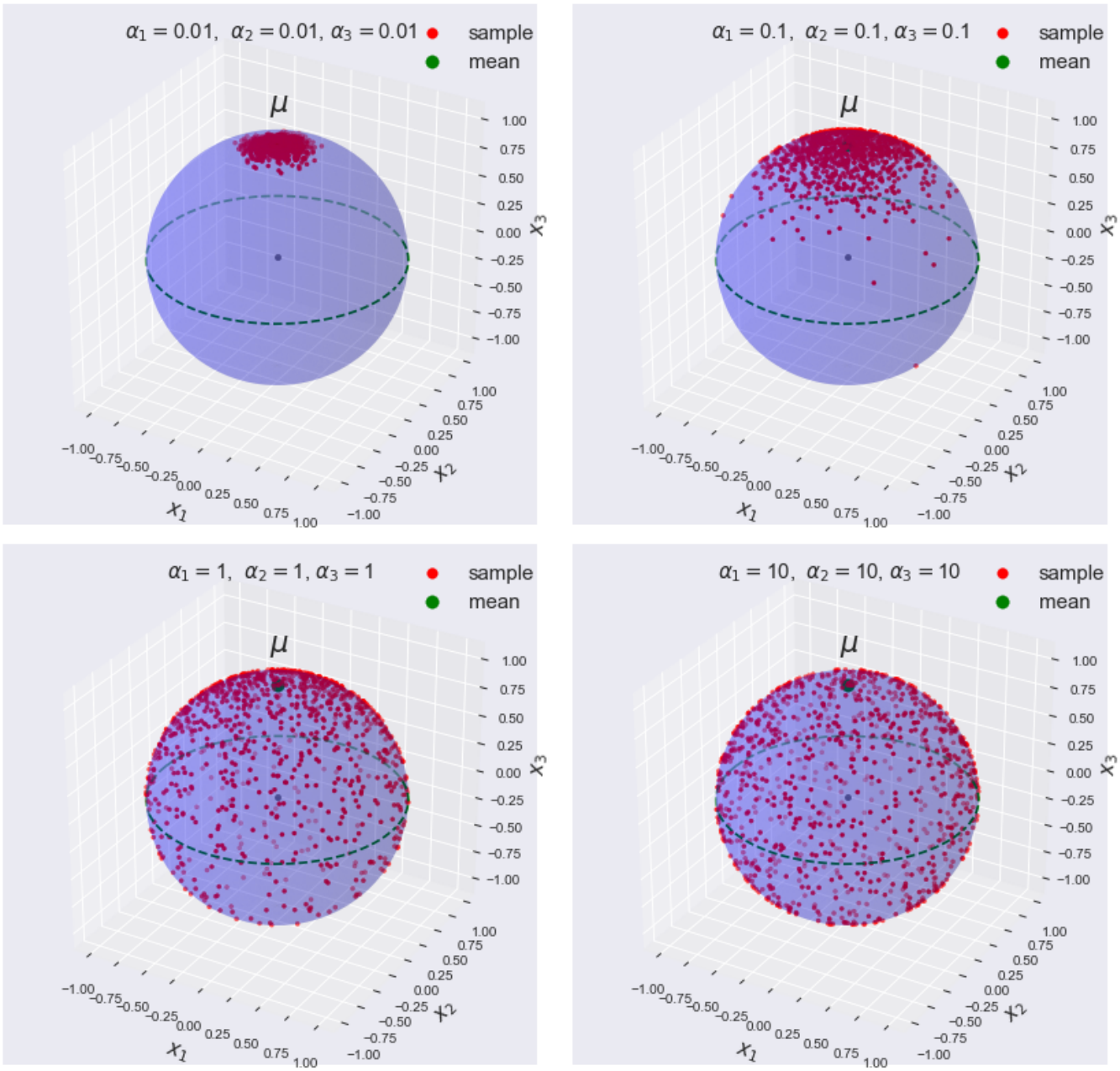

Figure 1: Isotropic Gaussian $\mathcal{N}(e_3, \alpha I)$ projected on a unit sphere in $\mathbb{R}^3$ for different values of α on every subplot ($e_3 = (0, 0, 1)$).

$$\phi \sim U[S_{n-1,\ tang}] \tag{20}$$

$$\varepsilon \sim t\mathcal{N}_{[-\pi/2,\pi/2]}(0, \sigma^2) \quad \text{or} \quad \varepsilon \sim s\mathcal{B}_{[-\pi/2,\pi/2]}(\beta,\ \beta) \tag{21}$$

$$\hat{v}_{norm} = \frac{v_{norm} + \tan\varepsilon \cdot \phi}{\|v_{norm} + \tan\varepsilon \cdot \phi\|} = \frac{v_{norm} + \tan\varepsilon \cdot \phi}{\sqrt{1 + (\tan\varepsilon)^2}} = \cos\varepsilon(v_{norm} + \tan\varepsilon \cdot \phi) = \tag{22}$$

$$= \cos\varepsilon \cdot v_{norm} + \sin\varepsilon \cdot \phi \tag{23}$$

The last equation holds because v_{norm} and ϕ are perpendicular, and ϕ is a unit vector from S_{n-1}.

By $t\mathcal{N}_{[-\pi/2,\pi/2]}(0,\sigma^2)$, we denote truncated Normal distribution on $[-\pi/2,\pi/2]$ with zero mean and variance σ^2. By stretched symmetric Beta distribution, we mean that $\varepsilon \sim s\mathcal{B}_{[-\pi/2,\pi/2]}(\beta,\ \beta)$ if $\theta \sim \mathcal{B}(\beta,\beta)$ and then $\varepsilon = \pi(\theta - 0.5)$.

Uniform distribution on S_{n-1} can be obtained via normalizing any isotropic distribution (for example, standard Normal $\mathcal{N}(0, I)$).

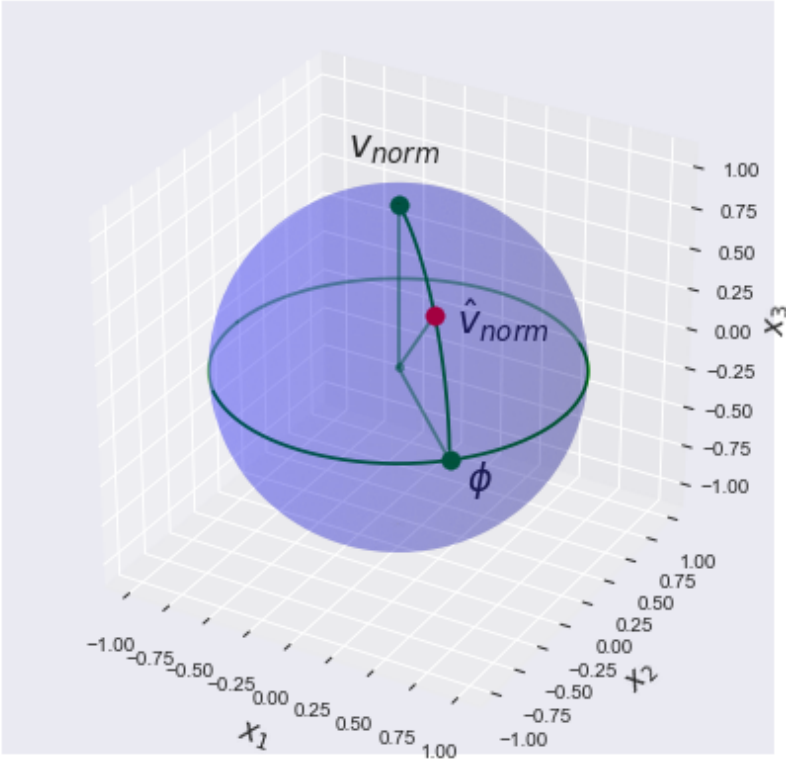

Figure 2: Direction noise sampling procedure demonstrated in $\mathbb{R}^3$: ϕ is sampled from uniform distribution on a unit circle which determines the direction in which v_{norm} is rotated for an angle ε and results in $\hat{v}_{norm}$.

3.3 Magnitude noise

For noise on magnitude parameter g, we take multiplicative Normal noise $\mathcal{N}(1,\alpha)$ similar to Gaussian dropout.

$$\varepsilon \sim \mathcal{N}(1,\alpha) \tag{24}$$

$$\hat{g} = g \cdot \varepsilon \tag{25}$$

Applying such a noise on magnitude with value m is equivalent to sampling magnitude from $\mathcal{N}(g \,|\, m, m^2\alpha)$. In section 4.2, the importance of independent noise for every object in a batch will be discussed, and for such type of noise it can be achieved via local reparametrization trick [4]. The pre-activation value is $y = g \cdot v_{norm}^T x$, and instead of sampling magnitudes g, we can translate noise to pre-activations and directly sample each pre-activation from $\mathcal{N}(x^T w,\ (x^T w)^2\alpha)$.

Another choice of noise distribution for magnitude could be Gamma distribution with domain $(0; +\infty)$ and expectation equal 1: this multiplicative noise would not change the sign of the vector which might be better for training stability.

3.4 Difference between Gaussian noise on input, on weights and magnitude noise

Let us carefully point out the difference between Gaussian noise on input, weights and magnitude. Consistently with previous sections, let X be a batch of activations of the preceding layer which is a matrix of size ($batch\_size$, in), x – one row of this matrix (vector of activations corresponding to one object from batch), and W – a weight matrix of size (in, out). We will apply multiplicative Gaussian noise in different settings.

1. Gaussian noise on weights W: sample random matrix $\mathcal{E}$ of size (in, out) where each $\mathcal{E}_{ij} \sim \mathcal{N}(1, \alpha)$, then apply noise elementwise to weights $W \odot \mathcal{E}$. It is important that we have different (independent) noise for each object in a mini-batch which would mean sampling not one but $batch\_size$ matrices $\mathcal{E}$. However, due to properties of Normal distribution, it can be done more efficiently via so-called local reparametrization trick [4] (see section 4.2 for more details).

   Looking closely at one object: instead of computing $x^T W$, we compute $x^T(W \odot \mathcal{E})$ which is equivalent to independently sampling pre-activation matrix from the distribution $y_j \sim \mathcal{N}(x^T w_{\bullet j}, \alpha \sum_k (x_k w_{kj})^2)$ where $w_{\bullet j}$ is a weight vector (single column in W).

   The final procedure is precomputing Gaussian means as XW and variances as $\alpha X^2 W^2$ (here, squaring is elementwise), and sampling each pre-activation value with correspondent distribution parameters.

2. Gaussian noise on input X: sample $\mathcal{E}$ – a random matrix of size ($batch\_size$, in) where $\mathcal{E}_{ij} \sim \mathcal{N}(1, \alpha)$, then apply the noise elementwise to activations $X \odot \mathcal{E}$.

   Looking closely at one object: instead of computing $x^T W$, we compute $(x \odot \varepsilon)^T W = x^T(\varepsilon \odot W)$ where the operation $\varepsilon \odot W$ means elementwise multiplication of the same noise vector ε with weight vectors (columns of W) $w_{\bullet 1}, \ldots, w_{\bullet\, out}$.

   Thus, each pre-activation y_j is distributed as $\mathcal{N}\left(x^T w_{\bullet j}, \alpha \sum_k (x_k w_{kj})^2\right)$ as a sum of independent Gaussian random variables. But the whole noise application operation is not equivalent to *independently* sampling pre-activations from Gaussian distribution with mean $x^T W$ and corresponding variances because of the same noise ε for different weight vectors $w_{\bullet j}$.

   The final procedure is computing pre-activations matrix as $Y = (X \odot \mathcal{E})W$.

3. Gaussian noise for magnitude g in weight normalization reparametrization, or Gaussian noise on pre-activations: in Weight Normalization parametrization, for each weight vector $w_{\bullet j}$ there is a magnitude parameter g_j on which we can apply multiplicative Gaussian noise. Sample ε – random vector of size out with $\varepsilon_i \sim \mathcal{N}(1, \alpha)$. Applying local reparametrization trick, it will be equivalent to independently sampling pre-activations from $\mathcal{N}\left(x^T w_{\bullet j}, \alpha \left(\sum_k x_k w_{kj}\right)^2\right)$. Note that the variance of the Normal distribution is not the same as in Gaussian weight noise.

   The final procedure is computing pre-activations as $Y = (XW) \odot \mathcal{E}$.

Gaussian noise on input and magnitude are different in a way whether we inject noise before or after the nonlinearity is applied.

3.5 Signal rotation

In section 4.2, the importance of sampling independent noise for objects in a mini-batch is stressed. It is not only crucial in the case of stochastic variational inference, but also for regular noise injection methods (like binary or Gaussian dropout). Independent noise provides more stochasticity and performs better at preventing overfitting. Similar to how Gaussian noise can be put on either weights or inputs, the magnitude and direction noise can be translated on neuron's activations vector. For multiplicative magnitude noise, it makes no difference whether it was applied on w or x as it results in the same pre-activation distribution. For direction noise, all the methods in section 3.2 can be straightforwardly applied to x_{norm} instead of v_{norm}: each activation can be also split into magnitude and direction $x = \|x\| \cdot x_{norm}$, and the noise would be applied to x_{norm}.

A combination of direction $p(v_{norm})$ and magnitude $p(g)$ noise produces a complex distribution over each weight vector w, but its marginal distribution $p(w)$ cannot be computed analytically. It is not obvious how the parameters of $p(v_{norm})$ and $p(g)$ affect the marginal distribution, in particular, how variances of $p(v_{norm})$ and $p(g)$ translate to the variance of the induced distribution over w, so the parameters of these distributions should be carefully chosen.

These parameters can be fixed during training and testing and treated as hyperparameters. Then, it is necessary to experiment with different values for them and perform a grid search. However, grid search is an exhaustive procedure and would not scale to tuning parameters for distributions individually for each neuron (we would instead have an identical distribution for all neurons in a network). It might be desirable to tune these hyperparameters automatically along with training network's weights and separately for each neuron, which can be efficiently done using variational inference framework.

4 Variational inference for proposed regularization

4.1 Probabilistic model

Variational inference framework allows performing approximate Bayesian inference in very complex models like deep neural networks (more details in section 2.4). In order to perform variational inference, we need to choose prior distributions as well as an approximation family for posterior distribution. Here, we introduce a probabilistic model consistent with the type of direction noise described in 3.2.3 where v_{norm} is rotated in a random plane.

Let $w = g \cdot v_{vorm}$ be a weight vector with magnitude g and direction v_{norm}. Let ϕ be a random point on S_{n-1} and ε – an angle from 0 to $\pi/2$. We assume that both the prior and the posterior distributions are factorized.

For ε, it is natural to take a uniform prior on $[0; \pi/2]$. Posterior approximation is the truncated Normal distribution $t\mathcal{N}_{[0,\pi/2]}(\mu, \sigma^2)$ with learnable parameters μ and σ^2. Note that in variational model we chose the domain $[0; \pi/2]$ for ε as opposite to having $t\mathcal{N}_{[-\pi/2,\pi/2]}(0, \sigma^2)$ as an approximate

posterior (analogously to the model in 3.2.3). In a standard non-variational setting, we wanted the expected value of $\hat{v}_{norm}$ to be v_{norm} as it would not make sense to set the parameter μ to any value other than 0. However, in variational setting, our model is more flexible, and μ can be tuned along with σ^2 which can lead to more complex distributions over $\hat{v}_{norm}$. To leave the distribution isotropic, we limit ε to the interval $[0; \pi/2]$.

For ϕ, we take the uniform distribution on $S_{n-1,\,tang}$ for both prior and posterior and, thus, do not tune any parameters for it.

For magnitude, the prior distribution is chosen to be the log-uniform which is known to be a sparsity inducing prior. It causes non-structural model sparsity when applied directly for weights w_{ij} in Variational Dropout [5] and structural sparsity in Structured Bayesian Pruning [11]. Applying it to weight vector magnitude can also potentially lead to structural sparsity since if the neuron's magnitude is set to zero, no signal would be passed forward through it, and it could be dropped from a network.

Prior distributions in the model:

$$p(g, \phi, \varepsilon) = p(g)\, p(\phi)\, p(\varepsilon) \tag{26}$$

$$p(g) \propto \frac{1}{|g|}, \quad p(\phi) \sim U[S_{n-1,\,tang}], \quad p(\varepsilon) \sim U[0; \pi/2] \tag{27}$$

Posterior approximations in the model:

$$q(g, \phi, \varepsilon) = q(g)\, p(\phi)\, q(\varepsilon) \tag{28}$$

$$q_{m,\,\alpha}(g) \sim \mathcal{N}(g \,|\, m, \alpha m^2), \quad p(\phi) \sim U[S_{n-1,\,tang}], \quad q_{\mu,\,\sigma^2}(\varepsilon) \sim t\mathcal{N}_{[0;\,\pi/2]}(\mu, \sigma^2) \tag{29}$$

The model is trained via maximizing variational lower bound (equation 8 in section 2):

$$\begin{aligned}
\mathcal{L}(v, m, \alpha, \mu, \sigma^2) &= L_{\mathcal{D}}(v, m, \alpha, \mu, \sigma^2) - \mathrm{KL}(q(g, \varepsilon) \,||\, p(g, \varepsilon)) = \\
&= L_{\mathcal{D}}(v, m, \alpha, \mu, \sigma^2) - \mathrm{KL}(q_{m,\,\alpha}(g) \,||\, p(g)) - \mathrm{KL}(q_{\mu,\,\sigma^2}(\varepsilon) \,||\, p(\varepsilon)) = \\
&= \int q_{m,\,\alpha}(g)\, p(\phi)\, q_{\mu,\,\sigma^2}(\varepsilon) \log p(Y \,|\, X, v, g, \phi, \varepsilon)\, dg\, d\phi\, d\varepsilon - \\
&\quad - \int q(g) \log \frac{q(g)}{p(g)}\, dg - \int q(\varepsilon) \log \frac{q(\varepsilon)}{p(\varepsilon)}\, d\varepsilon \quad \longrightarrow \max_{v,\,m,\,\log\alpha,\,\mu,\,\log\sigma^2}
\end{aligned} \tag{30}$$

In such a model, we can perform reparametrization trick, make unbiased gradient estimates using sampling and optimize the variational lower bound using stochastic optimization (see section 2.4). Reparametrization trick for g:

$$\xi \sim \mathcal{N}(0, 1) \qquad g = m(1 + \sqrt{\alpha}\,\xi) = m + \sqrt{\alpha}\, m\, \xi \tag{31}$$

Following [5], we use additive noise parametrization to reduce the variance of stochastic gradients: we introduce the parameter $\lambda = m\sqrt{\alpha}$, which corresponds to additive noise, and optimize with respect to $\log \lambda$ instead of $\log \alpha$.

Reparametrization trick for ε:

$$\gamma \sim U[0; 1] \qquad \varepsilon = \mu + \sigma \Phi^{-1}\left(\Phi(\alpha) + Z\gamma\right) \tag{32}$$

where $\alpha = -\mu/\sigma$, $\beta = (\pi/2 - \mu)/\sigma$ and $Z = \Phi(\beta) - \Phi(\alpha)$. The derivation of reparametrization trick for truncated Normal is presented in appendix B. Let us denote all the trainable parameters by $\theta = \{v, m, \alpha, \mu, \sigma^2\}$. Finally, a mini-batch-based Monte Carlo estimator is obtained:

$$L_D(\theta) \simeq L_D^{SGVB}(\theta) = \frac{N}{M} \sum_{k=1}^{M} \log p(y_{i_k} \,|\, x_{i_k}, w_{i_k} = f(\theta, \hat{\phi}, \hat{\xi}_{i_k}, \hat{\gamma}_{i_k})) \tag{33}$$

$$\mathcal{L}(\theta) \simeq \mathcal{L}^{SGVB}(\theta) = L_D^{SGVB}(\theta) - \mathrm{KL}(q_\theta(g, \varepsilon) \,\|\, p(g, \varepsilon)) \tag{34}$$

$$\nabla_\theta L_D(\theta) \simeq \nabla_\theta L_D^{SGVB}(\theta) \tag{35}$$

where N is the training set size and M is the batch size. For ε and g, noise is sampled independently for every object in a batch whereas for ϕ there is no straightforward way to efficiently achieve it so one sample is used for the whole batch. Stochastic gradients might have high variance due to dependence of noise for different object in a batch (see section 4.2).

KL-divergence for magnitude variable:

$$\mathrm{KL}(q_{m,\alpha}(g) \,||\, p(g)) = \sum_{i=1}^{L} \mathrm{KL}(q_{m_i,\alpha_i}(g_i) \,||\, p(g_i)) \tag{36}$$

$$\mathrm{KL}(q_{m_i,\alpha_i}(g_i) \,||\, p(g_i)) = \int q_{m_i,\alpha_i}(g_i) \log \frac{q_{m_i,\alpha_i}(g_i)}{p(g_i)}\, dg_i \approx$$

$$\approx \sum_i \left(k_1 \sigma(k_2 + k_3 \log \alpha_i) - 0.5 \log(1 + \alpha_i^{-1}) + \underbrace{C}_{-k_1} \right) \tag{37}$$

$$k_1 = 0.63576 \qquad k_2 = 1.87320 \qquad k_3 = 1.48695$$

The closed-form approximation for this KL-divergence is taken from [5].

KL-divergence for direction noise ε: if $p(\varepsilon)$ is uniform, KL-divergence is the negative entropy of $q_{\mu,\sigma^2}(\varepsilon)$ up to an additive constant.

$$\mathrm{KL}(q_{\mu,\sigma^2}(\varepsilon) \,||\, p(\varepsilon)) = \sum_{i=1}^{L} \mathrm{KL}(q_{\mu_i,\sigma_i^2}(\varepsilon_i) \,||\, p(\varepsilon_i)) \tag{38}$$

$$\mathrm{KL}(q_{\mu_i,\sigma_i^2}(\varepsilon_i) \,||\, p(\varepsilon_i)) = -\mathcal{H}(q_{\mu_i,\sigma_i^2}(\varepsilon_i)) + C = \int q_{\mu_i,\sigma_i^2}(\varepsilon_i) \log q_{\mu_i,\sigma_i^2}(\varepsilon_i)\, d\varepsilon_i + C =$$

$$= -\log(\sqrt{2\pi e}\, \sigma_i Z_i) - \frac{\alpha_i \phi(\alpha_i) - \beta_i \phi(\beta_i)}{2 Z_i} + C \tag{39}$$

where α_i, β_i and Z_i are defined above as well as in appendix B.

4.2 Gradient variance induced by direction noise dependence

To train neural networks on large datasets, we use stochastic optimization, and on every iteration process only a small mini-batch of data to make unbiased gradient estimate. Using batch size one would make one gradient update very fast, but gradients would become very noise and we would

need a lot of steps to reach the optima. Increasing the batch size reduces the gradient variance so we need to find a tradeoff between the time for processing a single batch and the number of steps needed for convergence.

However, in stochastic neural networks, mini-batch optimization is not the only source of high gradient variance. When injecting noise in network's weights, correlated noise greatly contributes to the expected log-likelihood's variance as well as its gradient's variance. In [4], the analysis of noise influence on gradient variance was presented:

$$L_i = \log p(y_i \,|\, x_i, w = f(\epsilon_i, \phi)) \qquad L_{\mathcal{D}}^{SGVB}(\phi) = \frac{N}{M} \sum_{i=1}^{M} L_i \tag{40}$$

$$\begin{aligned} \operatorname{Var}\left[L_{\mathcal{D}}^{\mathrm{SGVB}}(\phi)\right] =& \frac{N^2}{M^2}\Big(\sum_{i=1}^{M} \operatorname{Var}\left[L_i\right] + 2\sum_{i=1}^{M}\sum_{j=i+1}^{M} \operatorname{Cov}\left[L_i, L_j\right]\Big) && (41) \\ =& N^2\Big(\frac{1}{M}\operatorname{Var}\left[L_i\right] + \frac{M-1}{M}\operatorname{Cov}\left[L_i, L_j\right]\Big), && (42) \end{aligned}$$

$$\operatorname{Var}\left[L_i\right] = \operatorname{Var}_{\epsilon, x^i, y^i}\left[\log p(y^i | x^i, w = f(\epsilon, \phi))\right] \tag{43}$$

where the notation is consistent with the section 2.4. The second term in equation 42 can break the variance reduction effect achieved by a higher batch size: its coefficient $(M-1)/M$ grows with increasing batch size M so it can dominate the whole variance expression. Therefore, it is very important that we use uncorrelated noise to make $\operatorname{Cov}\left[L_i, L_j\right]$ as small as possible (preferably, zero).

Yet, it is expensive to sample and store separate weight perturbations for every object from a batch in terms of time and memory consumption so a single sample per mini-batch has to be used (which explodes the second term in equation 42). In some cases, it is possible to reformulate weight perturbations as activation perturbations via local reparametrization trick [4], the following is the case of Gaussian dropout in fully-connected layer:

$$\begin{aligned} q_\theta(w_{ij}) = \mathcal{N}(\mu_{ij}, \sigma_{ij}^2) \quad \forall w_{ij} \in W \Longrightarrow q_\theta(b_{mj}|X) = \mathcal{N}(\gamma_{mj}, \delta_{mj}) \\ \gamma_{mj} = \sum_{i=1} x_{mi}\mu_{ij}, \quad \text{and} \quad \delta_{m,j} = \sum_{i=1} x_{m,i}^2 \sigma_{ij}^2, \end{aligned} \tag{44}$$

where b_{mj} is the noised activation. Let us analyze whether local reparametrization trick can be performed in magnitude and direction noise. Instead of straightforwardly sampling magnitudes using equation 31, we can notice that there is a multiplicative dependence between pre-activation and magnitude, thus, multiplicative Normal noise can be translated to pre-activations in both fully-connected and convolutional layers:

$$\xi \sim \mathcal{N}(0, 1) \qquad y = m v_{norm}^T x + \sqrt{\alpha}\,(m v_{norm}^T x) \cdot \xi \tag{45}$$

This way independent noise ξ realted to magnitude can be sampled for every object in a batch.

In direction noise methods based on projecting Gaussian (introduced in sections 3.2.1 and 3.2.2), it is not possible to perform local reparametrization trick due to a complex dependence of pre-activations on noise and normalization operation. However, in direction noise from section 3.2.3,

it is partially possible: there, we have two sources of stochasticity $\phi \in S_{n-1}$ and $\varepsilon \in [0; \pi/2]$. The equation 23 presents a perturbed direction vector in a convenient way:

$$\hat{v}_{norm} = \cos\varepsilon \cdot v_{norm} + \sin\varepsilon \cdot \phi \tag{46}$$

Thus, a dot product with activations of the preceding layer x:

$$\hat{v}_{norm}^T x = \cos\varepsilon \cdot v_{norm}^T x + \sin\varepsilon \cdot \phi^T x \tag{47}$$

Therefore, we can sample matrix $\mathcal{E} \sim t\mathcal{N}_{[0;\pi/2]}(\mu, \sigma^2)$ of size $(batch\_size, out)$ and apply it elementwise directly on pre-activations. As for the ϕ variable, it is not tractable to obtain a separate sample for every object, and the tensor computation of pre-activations is as follows:

$$Y = \cos\mathcal{E} \odot XW + \sin\mathcal{E} \odot X\Phi \tag{48}$$

Ideally, we would have an independent matrix Φ_i for every object in a batch which is computationally inefficient. This inevitably leads to correlated noise and higher variance of gradients. This problem could be potentially tackled by recently proposed FlipOut technique [12] based on random sign matrix sampling which allows de-corellating weight samples and drastically reduces the variance for a negligible amount of extra computation. Another idea would be to notice that $\phi^T x = \|x\| \cos(\phi, x)$ as $\phi \in S_{n-1}$ and create a procedure to sample the angle between ϕ and x conditioned on x.

In practice, we did observe how crucial the local reparametrization trick is for model performance and the speed of convergence: in experiments for non-variational model (section 5.1.2), the performance of models where the noise was put on activations and, thus, was uncorrelated (section 3.5) was substantially better than that of the models with noise on weights.

4.3 Weight Normalization for input units

Standard Weight Normalization procedure normalizes output neurons (or channels, for convolutional layers). However, the dimension to be used for normalization is not restricted. For fully-connected layers, it might be desirable to decide which of the input neurons are significant and which can be dropped from the model (perform the relevance determination). This way, we can also perform feature selection for data. Therefore, we also consider a variational model where magnitude and direction noise in fully-connected layers is applied for input neurons. It is not exactly aligned with the initial motivation of two terms – magnitude and direction, directly contributing to pre-activation value, but the interpretation of neuron's significance remains.

5 Experiments

We use standard classification datasets of hand-written digits MNIST. In all the experiments, we compare a performance of the proposed regularization procedure with unregularized network and Gaussian dropout. All networks use Weight Normalization parametrization. We use Adam optimizer [13] and train models for 200 epochs (for the last 100 epochs, we linearly decrease learning rate to zero). We are interested in comparing 3 types of performance: the performance of

deterministic network, the performance of one randomly sample network and ensemble of networks (the predictions are averaged for 30-50 models).

5.1 Non-variational model

To test regularization abilities, we need to design an experiment where overfitting would occur: this can be done by taking less data in a training set and taking a neural net architecture with a sufficient number of parameters. Accordingly with [2], we vary train set size in a range 1000, 5000, 10000 and 50000 examples (almost the whole train set) and use a fully-connected neural net with 3 hidden layers of sizes 1024-1024-2048. We perform a grid search over a variance parameter of the noise distribution for all the proposed models as well as Gaussian dropout and report the best scores. We test direction and magnitude noise applied separately as well as injecting them simultaneously.

5.1.1 Effects of projected Gaussian noise injection

As long as network's weights are sampled from some distribution during training, a proper way to perform inference during the test phase would be to do ensembling or taking an expectation of the model's prediction over weights' distribution:

$$p(y \mid x) = \int q(w)p(y \mid x, w)dw = \mathbb{E}_{q(w)}\left[p(y \mid x, w)\right] \approx \sum_{i=1}^{n} p(y \mid x, \hat{w}_i) \qquad \hat{w}_i \sim q(w) \tag{49}$$

Nevertheless, weights are usually fixed on the test phase, or, in other words, we propagate the expectation for making the fast prediction. This still shows great test performance for both binary and Gaussian dropout.

However, for Gaussian projected noise (both simple and tangent subspace cases), an interesting effect was observed: expectation can not be propagated, and a deterministic model with fixed weights (where each distribution is replaced with Dirac delta-function in the mean) fails to work on both train and test sets. At the same time, Monte-Carlo ensembling and even one random model sample work well. On figure 3, the accuracy for train and test sets is presented for deterministic, stochastic and ensemble models, and the deterministic model's quality starts to decrease within the first few epochs while both stochastic and ensemble performance improve throughout the training. This can be explain by the fact that the probability mass of a Gaussian random variable in a high dimensional space is concentrated on the surface of hypersphere of radius proportional to its dimensionality and standard deviation [14]. Thus, neural net's weight samples essentially come from some hypersphere, and its center becomes an unrepresentative point for the net especially for large values of α. In appendix D, the distributions of angles between v_{norm} and $\hat{v}_{norm}$ are shown for all the direction noise methods, and it is clearly seen that even for small values of α, $\hat{v}_{norm}$ has a significant deviation in direction from v_{norm}.

Another experiment which was conducted to explore the peculiarities of Gaussian projected noise was the following: we trained a network (with the architecture mentioned above) using Gaussian projection noise on direction with variance $\alpha = 0.02$ (which shows the behaviour of deterministic network deterioration similar to the one plotted on the figure 3). After training for 200 epochs, we take the deterministic network and choose a few test pictures on which this

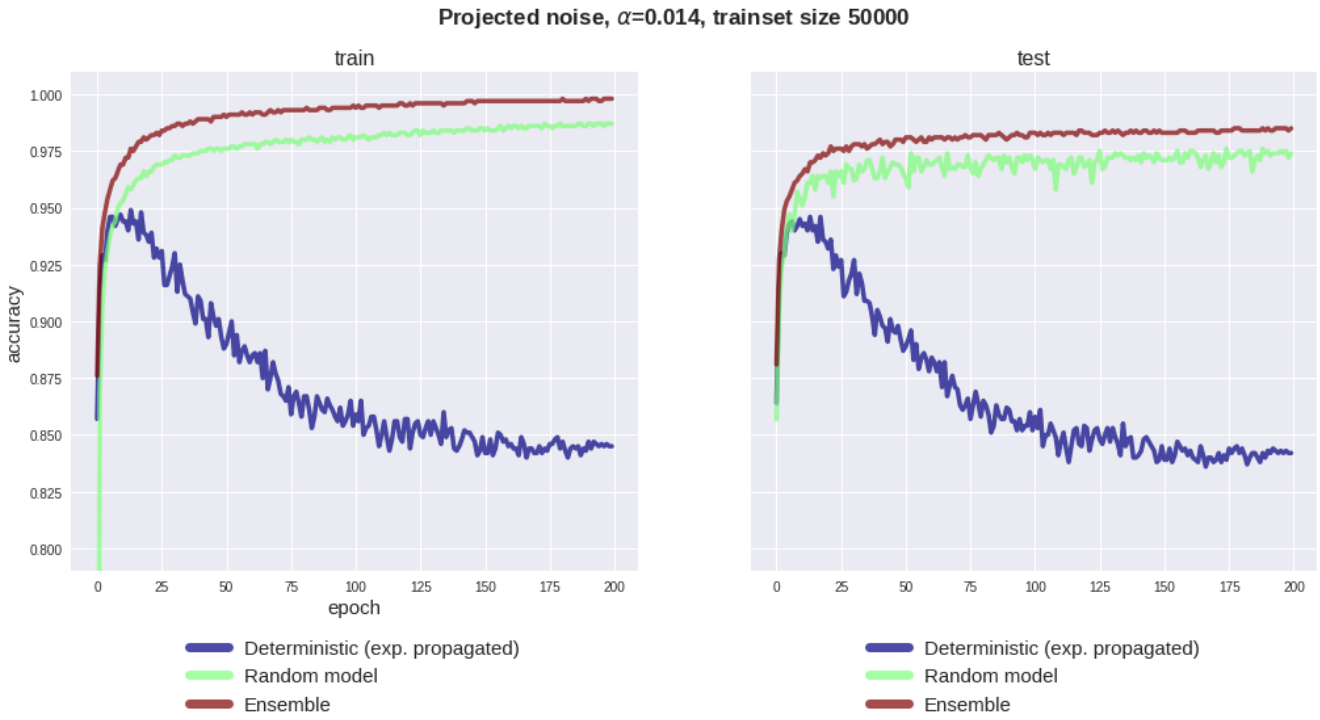

Figure 3: Accuracy on MNIST on train dataset (left) which contained 50000 images and test dataset (right) during training. A fully-connected neural network with 1024-1024-2048 neurons was trained for 200 epochs using Adam optimizer. The regularization method used was the projected Gaussian noise with $\alpha = 0.014$ (see section 3.2.1), no magnitude noise was applied.

model makes a wrong prediction. Eventually, we test the behaviour of stochastic networks which have the same mean weights as the deterministic one and a projected Gaussian noise injected with different variances α, smaller and higher than the one used during training, from the range $\alpha = \{0, \dots 1\}$. It corresponds to a range of stochastic networks starting from the deterministic one, then some networks with α smaller than during training, up to the initial network with $\alpha = 0.02$, and then with larger noise $\alpha > 0.02$. For these networks, we plot the output softmax distribution over MNIST classes (namely, its mean and standard deviation for 50 randomly sampled models for each value of α). The results are shown on figures 4 and 5 (more pictures are provided in appendix C). With increasing α, the distribution on labels changes as follows: for $\alpha \ll 0.02$, the model is certain in a wrong label; as α increases, distribution becomes bi-modal where the second mode is in the true answer; the closer value of α gets to the one which was used during training, the higher the probability of true label becomes, and for $\alpha = 0.02$ the model is highly confident in the true label; if we further increase the variance, surprisingly, for all the pictures, the mode at 8 appears; and when α is close to 1, the distribution becomes uniform.

This behavior is related to the effect observed in Variance Networks [15]: the important information about the data can be stored in variances instead of weights' means. The high variance of noise injected during training induces that weights sampled for the network lie on the surface of a hypersphere (see angle distributions in appendix D) so, when they are substituted with the mean of distribution, the model works poorly as such kind of weights were not seen during training.

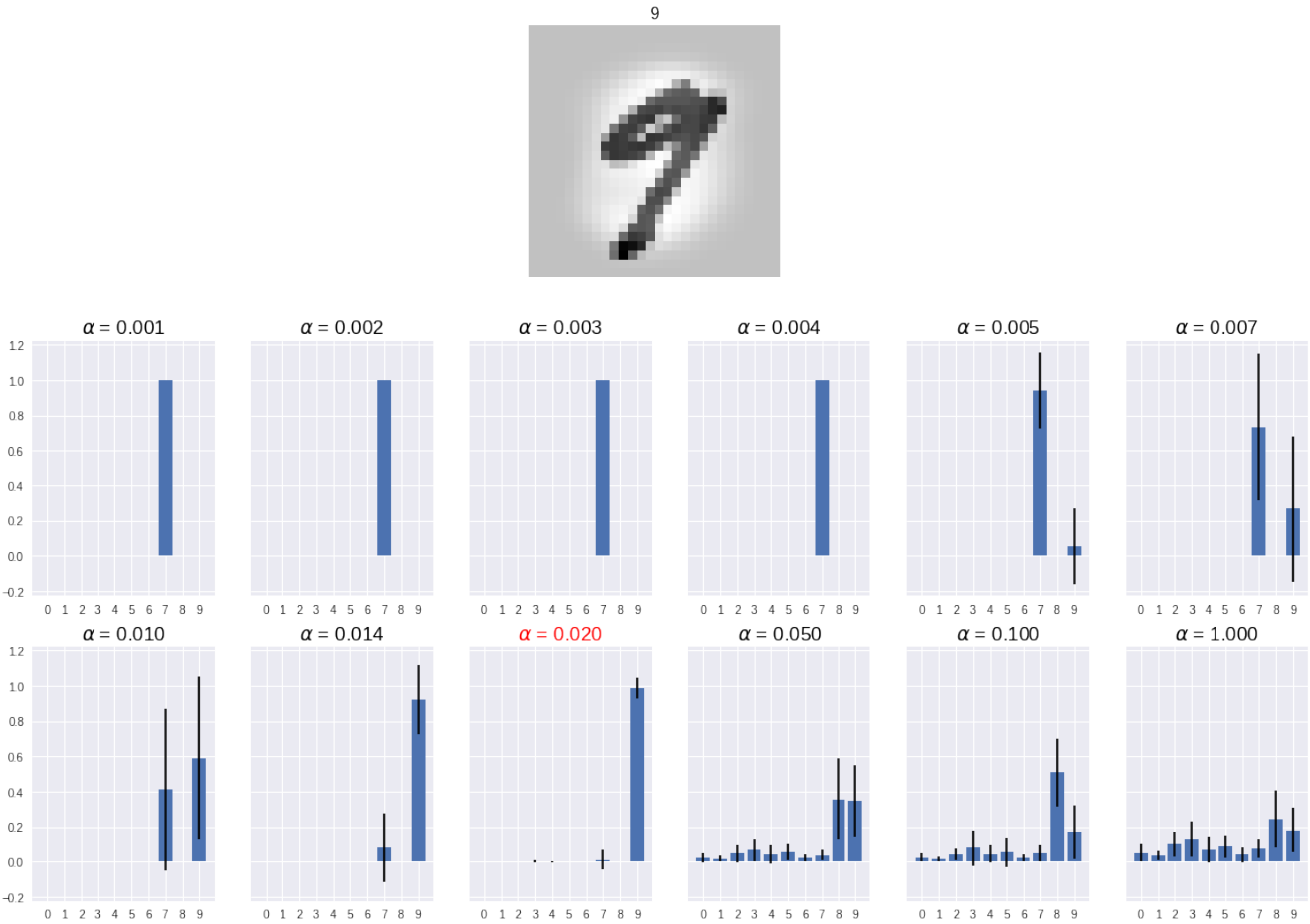

Figure 4: Softmax distribution over MNIST classes for test image with label 9. The initial model was trained with Gaussian projected noise with $\alpha = 0.02$, then its mean weights were fixed. Each picture corresponds to a model with mean weights form deterministic network and various values of α.

5.1.2 Performance comparison

We performed extensive grid search for all the regularization methods, and the results (with best hyperparameters for each model) are reported in tables 1 and 2. Table 2 shows the results for models where the magnitude and direction noise was applied for weights, whereas table 1 presents the performance of these methods when applied to activations. It can be easily noticed that signal rotation significantly outperforms weight rotations which is caused by the gradient variance discussed in section 4.2. The same explanation goes for the fact that in the weight noise injection case, magnitude noise model performs better than any of the other proposed models: a single direction noise sample per mini-batch was used whereas local reparametrization trick was applied for magnitude.

Although Gaussian dropout shows better performance, the regularization effect is notable for direction and magnitude noise. It is interesting to explore how these models perform in variational setting and automatic hyperparameter tuning.

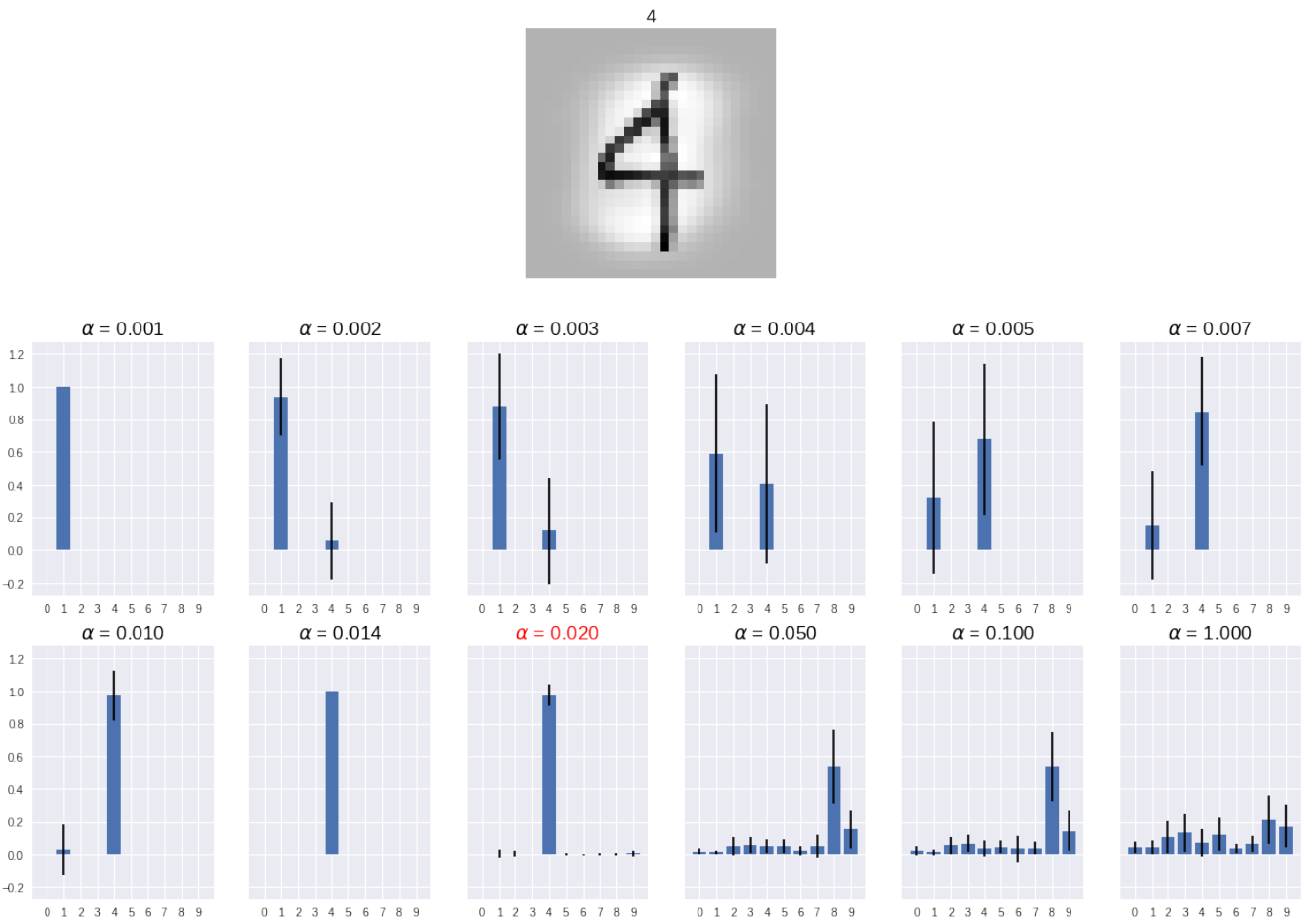

Figure 5: Softmax distribution over MNIST classes for test image with label 4. The initial model was trained with Gaussian projected noise with $\alpha = 0.02$, then its mean weights were fixed. Each picture corresponds to a model with mean weights form deterministic network and various values of α.

Table 1: Test accuracy (ensembling) on MNIST of FC 1024-1024-2048 network, comparison of different noise injection methods: direction and magnitude noise was applied to activations, not weights, to enable independent noise per object in a mini-batch. From left to right: unregularized network, Gaussian dropout (on activations). Further: Rot stands for rotation-in-a-plane direction noise, Magn stands for magnitude noise, $s\mathcal{B}$ for stretched symmetric Beta distribution, $\mathcal{N}$ for Normal distribution, $\mathcal{G}$ for Gamma distribution.

Trainset size	No reg	Gauss inp	Rot $s\mathcal{B}$ (X)	Rot $s\mathcal{B}$, Magn $\mathcal{G}$ (X)	Rot $s\mathcal{B}$, Magn $\mathcal{N}$ (X)	Rot $\mathcal{N}$ (X)	Rot $\mathcal{N}$, Magn $\mathcal{N}$ (X)
1000	0.904	**0.931**	0.928	0.928	0.927	0.920	**0.930**
5000	0.958	**0.971**	0.968	**0.969**	0.967	0.966	0.967
10000	0.965	**0.979**	0.978	**0.979**	0.977	0.976	0.974
50000	0.985	0.989	0.989	0.989	0.988	0.989	0.989

Table 2: Test accuracy (ensembling) on MNIST of FC 1024-1024-2048 network, comparison of different noise injection methods: direction and magnitude noise was applied to weights. From left to right: unregularized network, Gaussian dropout (on activations). Further: Rot stands for rotation-in-a-plane direction noise, Proj for projected Gaussian, Tang for projected Gaussian from tangent subspace, Magn stands for magnitude noise, $s\mathcal{B}$ for stretched symmetric Beta distribution, $\mathcal{N}$ for Normal distribution.

Trainset size	No reg	Gauss inp	Magn $\mathcal{N}$	Rot $\mathcal{N}$, Magn $\mathcal{N}$	Gauss weight	Proj	Tang	Rots$\mathcal{B}$	Rot$\mathcal{N}$
1000	0.9037	**0.931**	**0.921**	0.920	0.910	0.913	0.911	0.910	0.912
5000	0.958	**0.971**	**0.968**	0.965	0.960	0.964	0.959	0.959	0.962
10000	0.965	**0.979**	**0.975**	0.974	0.972	0.972	0.973	0.973	0.972
50000	0.985	**0.989**	**0.989**	0.986	0.987	0.987	0.988	0.988	0.987

5.2 Variational model

We trained the model described in section 4.1 for three neural networks architectures: LenNet300-100 (having two hidden layers and 300 and 100 units), LeNet500-300 (having two hidden layers with 500 and 300 units) and LeNet5 (two convolutional layers with 20 and 50 filters followed by a fully-connected layer with 500 units) on the whole MNIST dataset of 60000 images. A single direction noise sample was used for the whole mini-batch while local reparametrization trick was used for magnitude noise. We test both standard variational model as well as the one where Weight Normalization in fully-connected neurons is performed for the input neurons. The results are presented in the table 3.

All the models show very good performance with standard variational model slightly outperforming Gaussian dropout on LeNet5 architecture. However, MNIST classification task is too simple for fair model quality estimation while being a good sanity-check for any model's capacity.

We also investigate the learned distributions over g and v_{norm} (see figure 6). Note that magnitude and direction distributions are connected: for magnitudes, distributions which have a high peak in zero, the direction distribution is close to the uniform; while for magnitude distributions centered at nonzero value, corresponding direction distribution is centered at zero and has lower variance. Also, some of the direction distributions are bi-modal which corresponds to the concentration of probability mass on a hypersphere surface. Magnitudes for which posterior approximation distributions are close to delta-function in zero can be removed from the network with no drop in quality.

Table 3: Test accuracy (ensembling) on MNIST full data set for three network architectures. Var stands for the standard variational model, Var FC reverse – for the one with fully-connected layers normalized for input neurons.

Arch	No reg	Gauss inp	Var	Var FC reverse
LeNet300-100	0.9860	**0.9881**	0.9876	0.9870
LeNet500-300	0.9842	**0.9901**	0.9858	0.9872
LeNet5	0.9940	0.9948	**0.9955**	0.9906

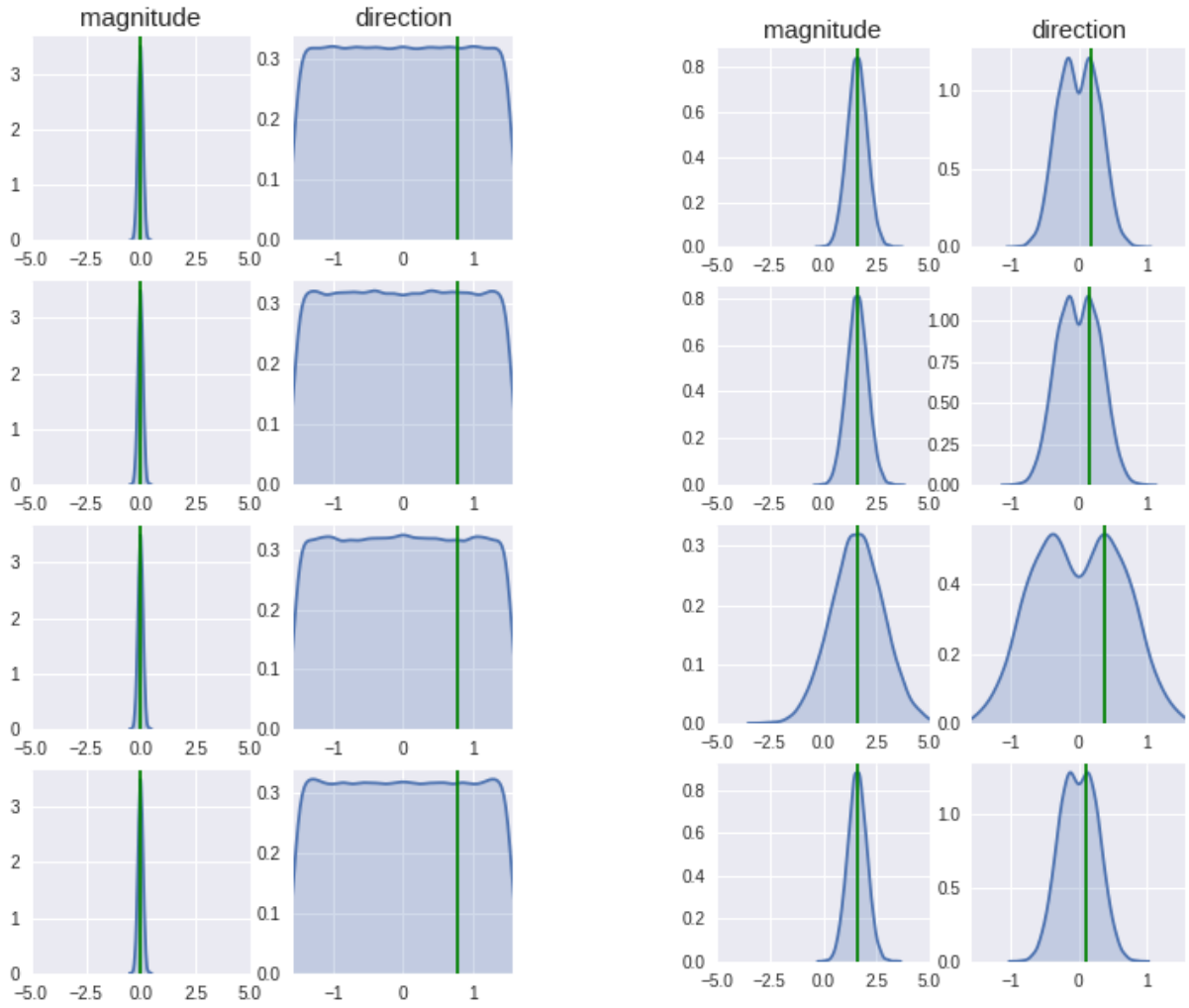

Figure 6: Learned posterior approximation distributions for neurons within one layer. On the left, magnitudes have zero value with high probability and, thus, these neurons are not significant. On the right, the magnitude is centered at nonzero value and such neurons do contribute to making predictions.

6 Conclusion

In this work, we presented a new way of neural network regularizaion: direction and magnitude perturbations for weight vectors. Although this technique prevents overfitting and improves unregularized network's performance, it does not significantly outperform Gaussian dropout baseline. This can be due to a high gradients variance induced by correlated direction noise samples for mini-batch. Some interesting effects were also observed in projected Gaussian direction noise model: a trained network shows good ensembling performance while performing very poorly in deterministic mode. This property is explained by the fact that high variance for direction parameter encourages model to seek optimal weights on a hypersphere surface. There is a number of challenges in both non-variational and variational models to be addressed in the future work.

Firstly, local reparametrization trick should be applied for scalar parameter ε in rotation-in-a-plane direction noise. It is expected to decrease gradients variance and boost model's performance.

Along with it, recently proposed FlipOut method [12] can be incorporated to further promote the variance reduction (applied to parameter ϕ in the same type of noise). We could also change posterior approximation for parameter ϕ: instead of uniform distribution on S_{n-1}, we could use non-isotropic projected Gaussian and tune its variances to make the model more flexible in finding useful directions for weight vectors. Moreover, unstable training of variational models might require using tricks like model pre-training or treating KL-divergence as a regularizer and multiplying it by some coefficient λ (with $\lambda > 1$ an optimized objection would still be an evidence lower bound). Further, we might explore other prior probabilities which would encourage different properties in models: for instance, changing magnitude prior from sparsity-inducing to some distribution on $(0; +\infty)$ with mode or expectation 1; or a prior on ε which would induce rotation for an angle $\pi/2$.

Also, more experiments should be conducted on other network architectures (like VGG or ResNet) and other datasets (CIFAR-10, CIFAR-100) to better estimate the applicability of the proposed methods.

References

[1] Understanding deep learning requires rethinking generalization / C. Zhang, S. Bengio, M. Hardt [et al.] // ArXiv e-prints. 2016. .

[2] Dropout: A Simple Way to Prevent Neural Networks from Overfitting / Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky [et al.] // Journal of Machine Learning Research. 2014. . 15. . 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[3] Wang Sida, Manning Christopher. Fast dropout training. Atlanta, Georgia, USA, 2013. 17–19 Jun. . 28, 2. . 118–126. URL: http://proceedings.mlr.press/v28/wang13a.html.

[4] Kingma D. P., Salimans T., Welling M. Variational Dropout and the Local Reparameterization Trick // ArXiv e-prints. 2015. .

[5] Molchanov D., Ashukha A., Vetrov D. Variational Dropout Sparsifies Deep Neural Networks // ArXiv e-prints. 2017. .

[6] Regularization of Neural Networks using DropConnect / Li Wan, Matthew Zeiler, Sixin Zhang [et al.] // Proceedings of the 30th International Conference on Machine Learning / . Sanjoy Dasgupta, David McAllester. . 28 *Proceedings of Machine Learning Research*. Atlanta, Georgia, USA: PMLR, 2013. 17–19 Jun. . 1058–1066. URL: http://proceedings.mlr.press/v28/wan13.html.

[7] Salimans T., Kingma D. P. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks // ArXiv e-prints. 2016. .

[8] Kingma D. P, Welling M. Auto-Encoding Variational Bayes // ArXiv e-prints. 2013. .

[9] Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // ArXiv e-prints. 2015. .

[10] Lei Ba J., Kiros J. R., Hinton G. E. Layer Normalization // ArXiv e-prints. 2016. .

[11] Structured Bayesian Pruning via Log-Normal Multiplicative Noise / K. Neklyudov, D. Molchanov, A. Ashukha [et al.] // ArXiv e-prints. 2017. .

[12] Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches / Y. Wen, P. Vicol, J. Ba [et al.] // ArXiv e-prints. 2018. .

[13] Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization // ArXiv e-prints. 2014. .

[14] Bishop Christopher M. Pattern Recognition and Machine Learning (Information Science and Statistics). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[15] Variance Networks: When Expectation Does Not Meet Your Expectations / K. Neklyudov, D. Molchanov, A. Ashukha [et al.] // ArXiv e-prints. 2018. .

Appendices

A Non-isotropic distributions on a unit sphere

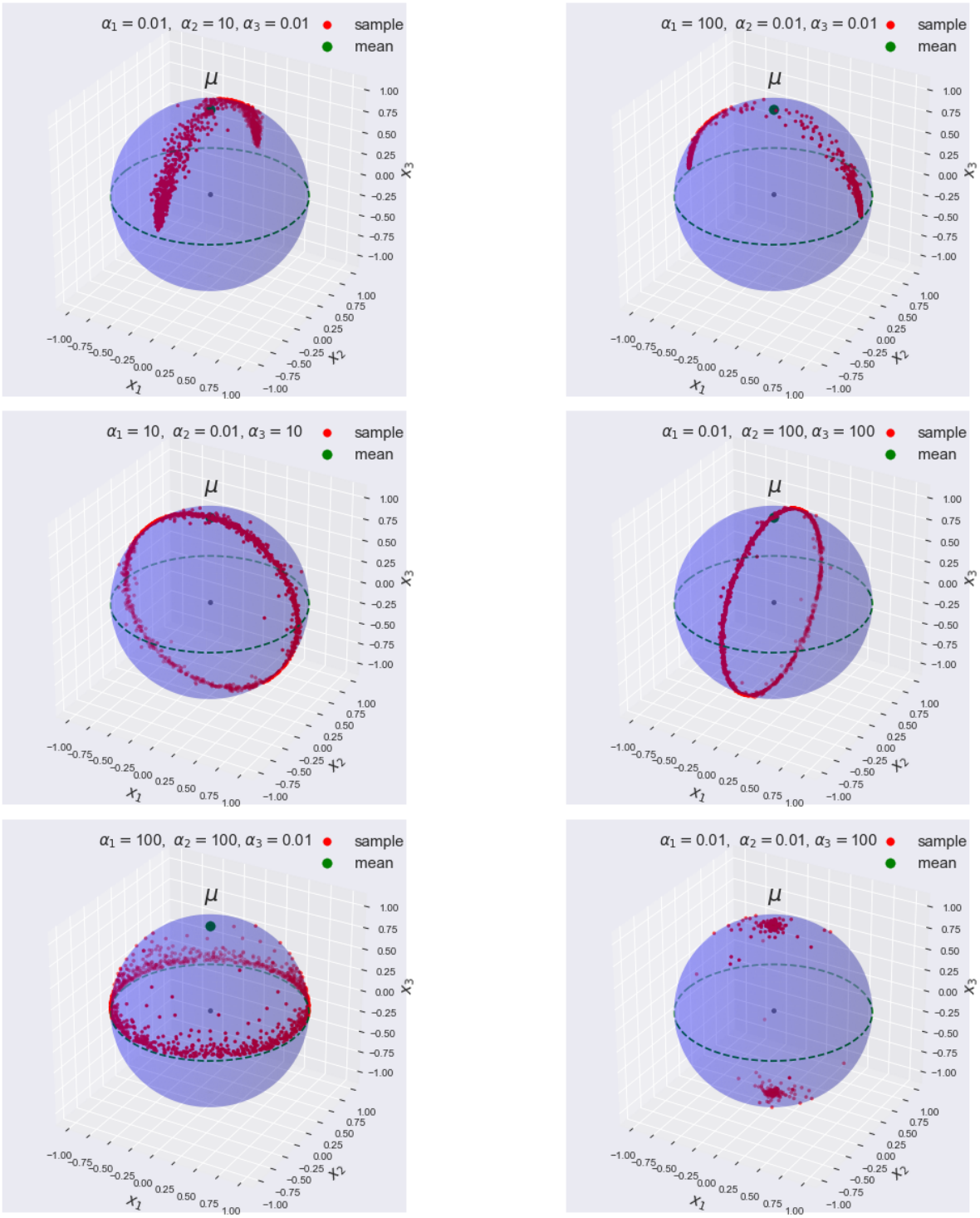

Figure 7: Non-isotropic Gaussian $\mathcal{N}(e_3, \bar{\alpha}I)$ projected on a unit sphere in $\mathbb{R}^3$ for different values of vector $\bar{\alpha}$ on every subplot ($e_3 = (0, 0, 1)$).

B Sampling from the truncated normal distribution

To compute gradients with respect to distribution parameters of $q_{\mu,\sigma^2}(\varepsilon) = t\mathcal{N}_{[a;\,b]}(\mu,\,\sigma^2)$, we need reparametrization trick for truncated Normal distribution which can be achieved using the inverse CDF. Some auxilary notations:

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

The CDF for truncated normal $t\mathcal{N}_{[a;\,b]}(x \,|\, \mu, \sigma^2)$:

$$F(x) = \frac{\Phi(\xi(x)) - \Phi(\alpha)}{Z} = y$$

$$\Phi(\xi(x)) = \Phi(\alpha) + Zy$$

$$\frac{x-\mu}{\sigma} = \Phi^{-1}\left(\Phi(\alpha) + Zy\right)$$

$$x = \mu + \sigma\Phi^{-1}\left(\Phi(\alpha) + Zy\right)$$

In our particular case $t\mathcal{N}_{[0,\pi/2]}(x \,|\, \mu, \sigma^2)$:

$$\alpha = \frac{-\mu}{\sigma} \qquad \beta = \frac{\pi/2-\mu}{\sigma} \qquad Z = \Phi\left(\frac{\pi/2-\mu}{\sigma}\right) - \Phi\left(\frac{-\mu}{\sigma}\right)$$

In implementation, it is important to take care of numerical instabilities arising in Φ^{-1} operation when its argument is close to 1: it would explode to infinity and overflow, so its argument values should be clipped.

C Projected Gaussian MNIST predictions for different variances

On figures 8, 9 and 10, more illustrations for experiment described in 5.1.1 are provided. The effect shows how the model with high direction noise during training learns to find an optimal hypersphere for its weights.

D Angles between true and perturbed vectors

The figures 11, 12, 13 and 14 show the distribution of an angle between the vector v_{norm} and its noised version $\hat{v}_{norm}$ for different types of direction noise.

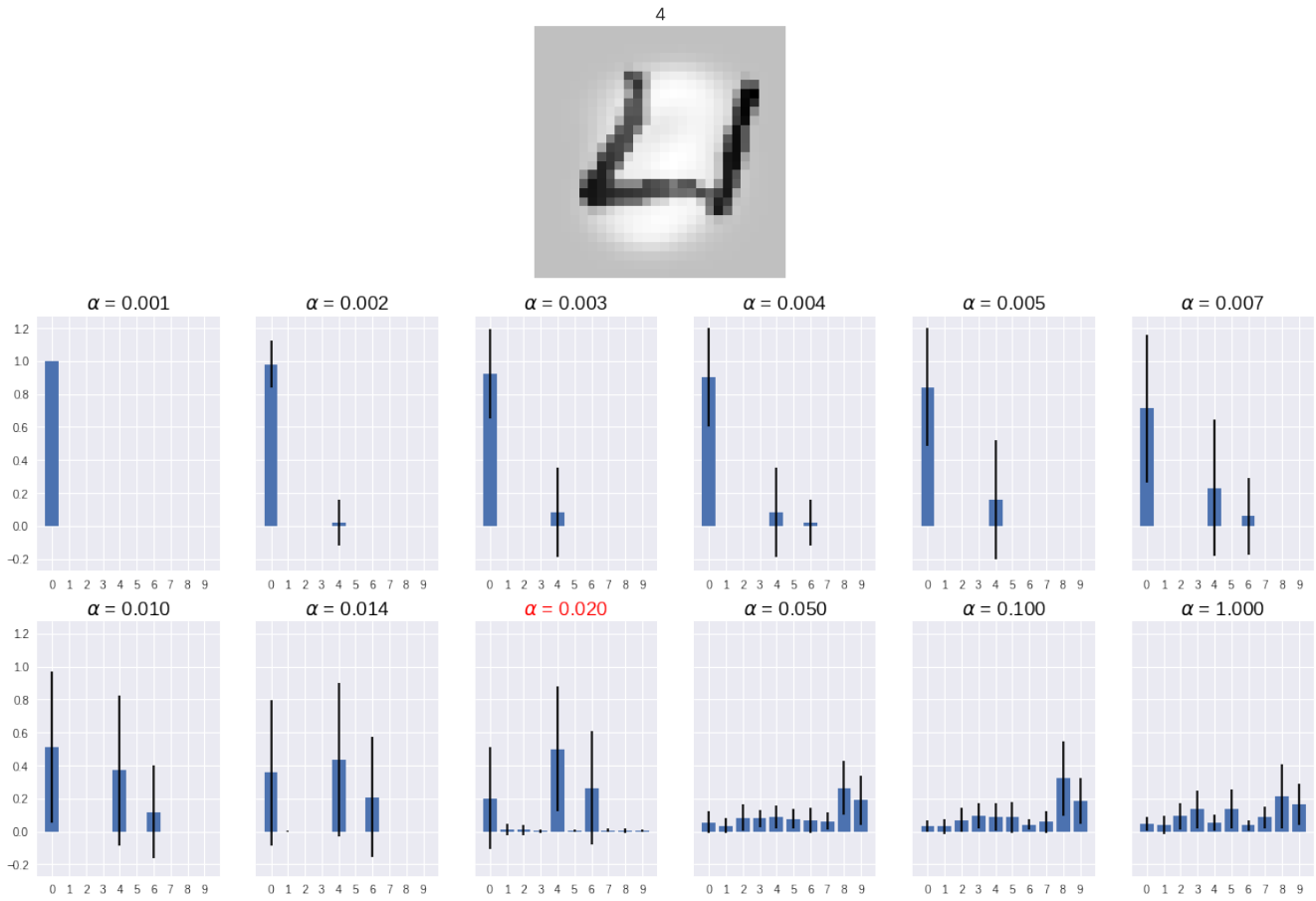

Figure 8: Softmax distribution over MNIST classes for test image with label 4. The initial model was trained with Gaussian projected noise with $\alpha = 0.02$, then its mean weights were fixed. Each picture corresponds to a model with mean weights form deterministic network and various values of α.

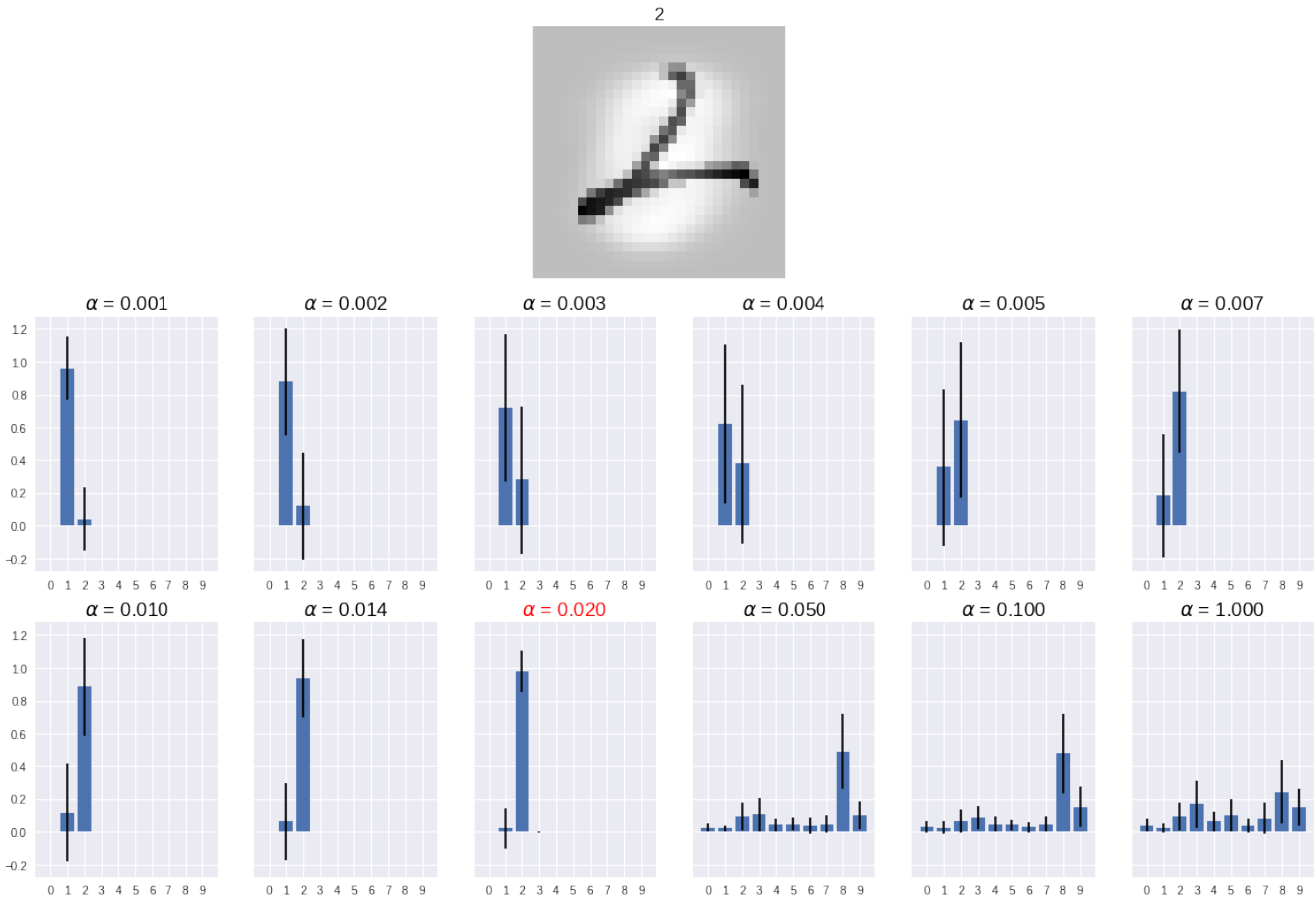

Figure 9: Softmax distribution over MNIST classes for test image with label 2. The initial model was trained with Gaussian projected noise with $\alpha = 0.02$, then its mean weights were fixed. Each picture corresponds to a model with mean weights form deterministic network and various values of α.

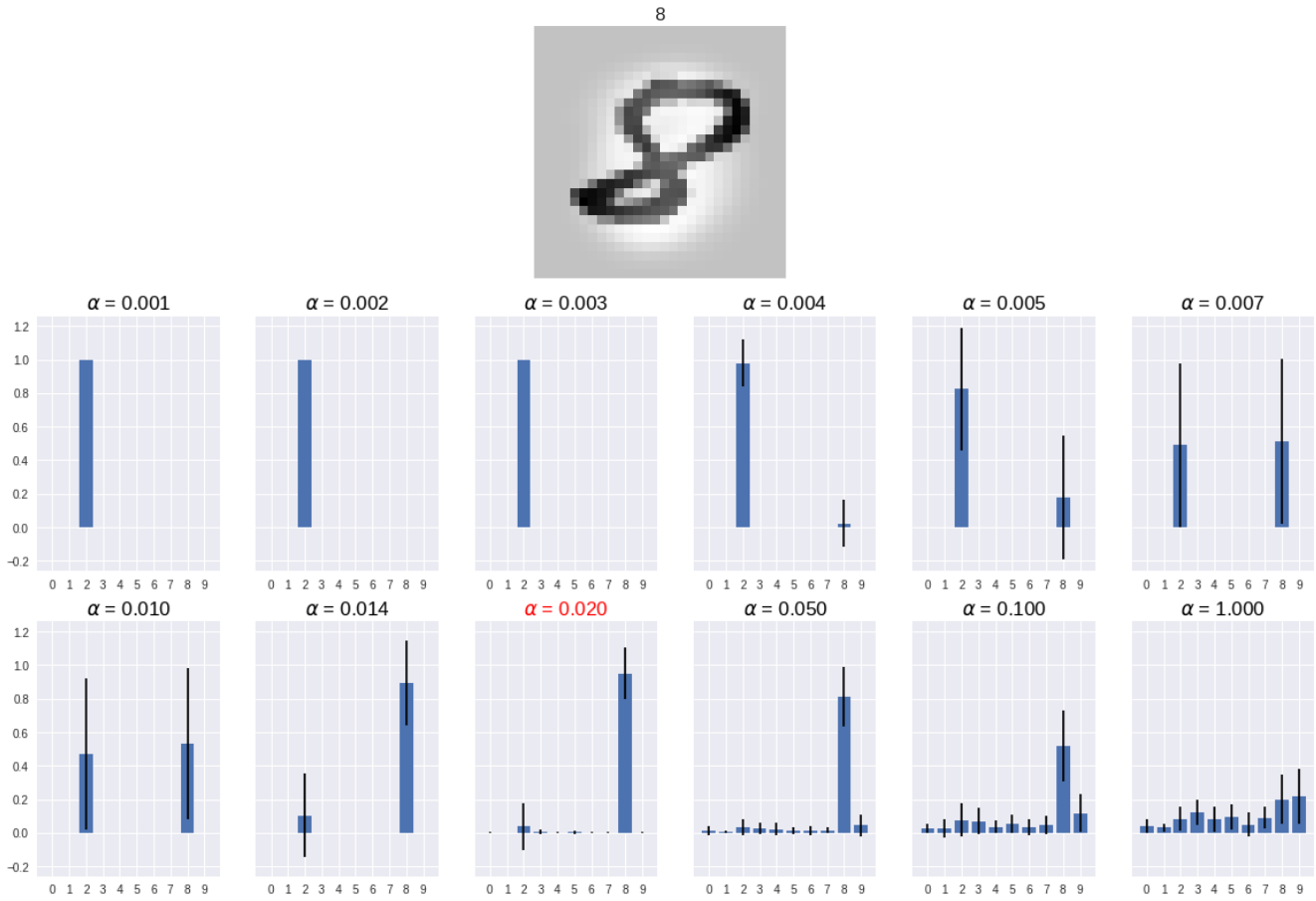

Figure 10: Softmax distribution over MNIST classes for test image with label 8. The initial model was trained with Gaussian projected noise with $\alpha = 0.02$, then its mean weights were fixed. Each picture corresponds to a model with mean weights form deterministic network and various values of α.

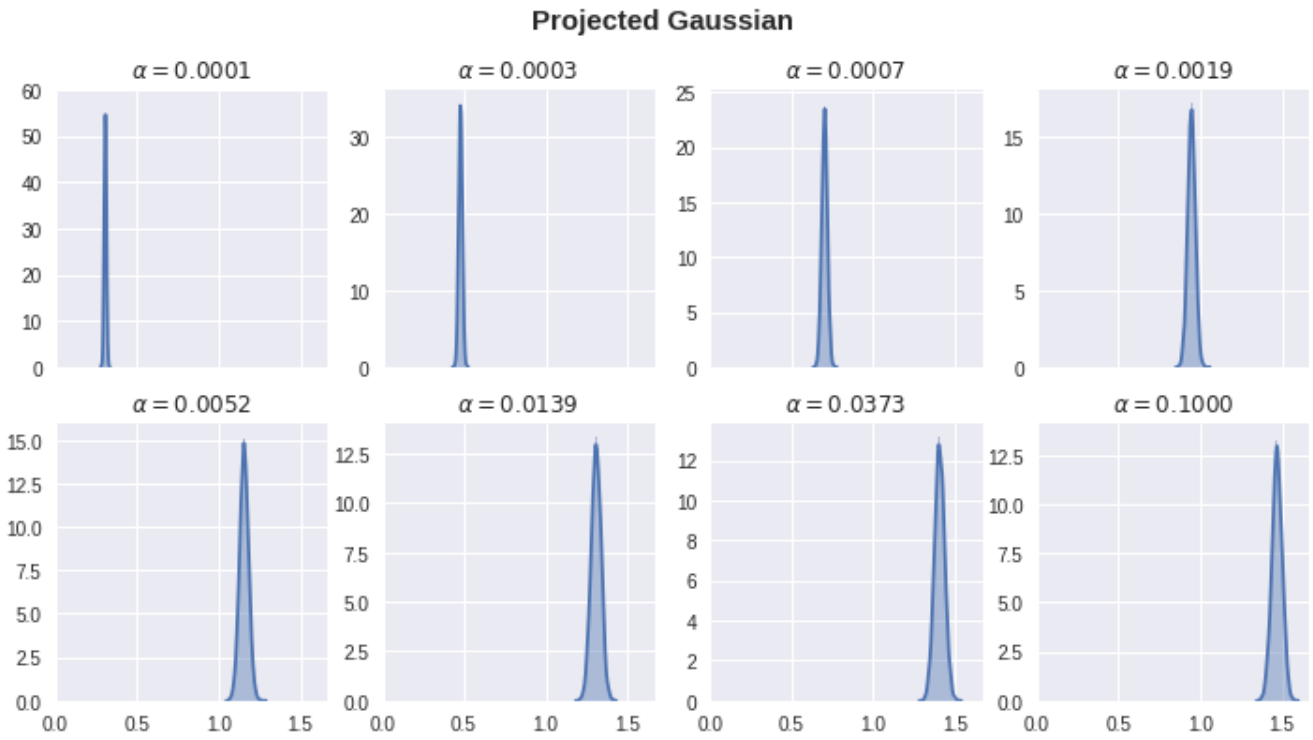

Figure 11: The distribution of angle between v_{norm} and $\hat{v}_{norm}$ for Gaussian projected direction noise for different values of α. Each picture is obtained with 10000 samples of $\hat{v}_{norm}$ in $\mathbb{R}^{1000}$.

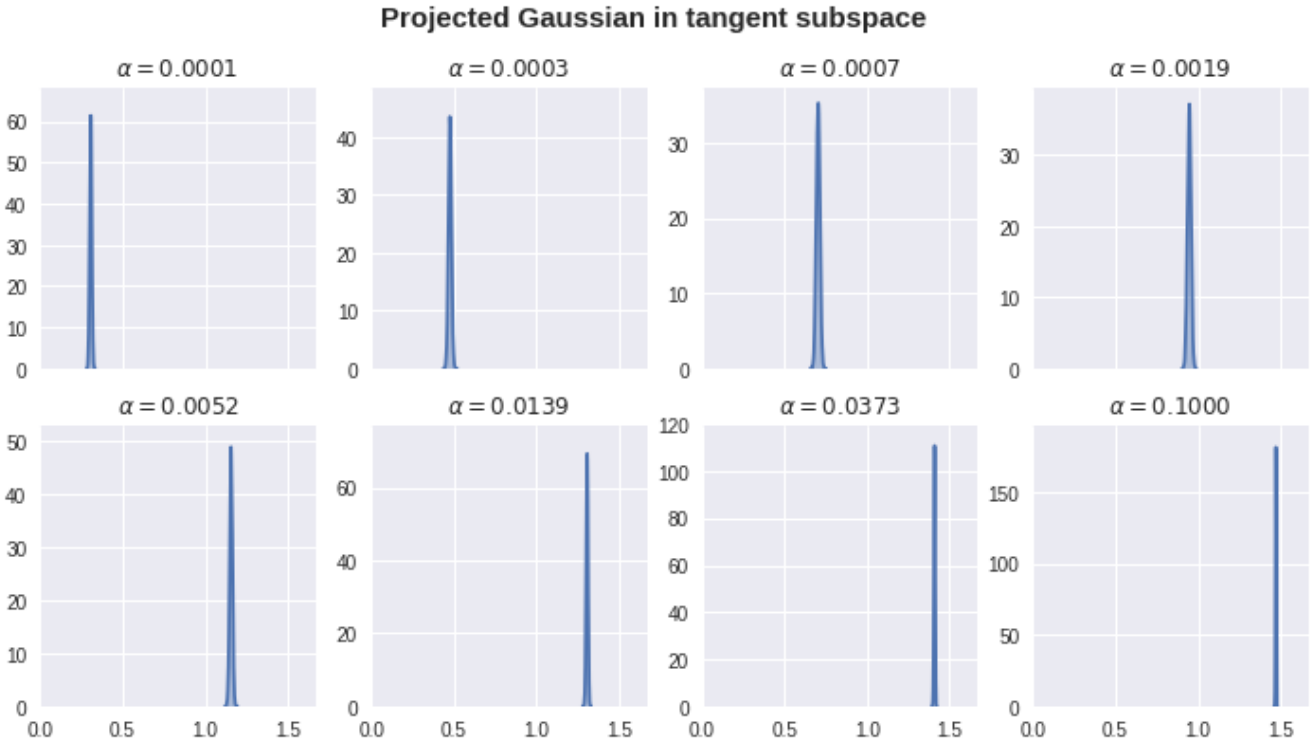

Figure 12: The distribution of angle between v_{norm} and $\hat{v}_{norm}$ for projected Gaussian noise from tangent subspace for different values of α. Each picture is obtained with 10000 samples of $\hat{v}_{norm}$ in $\mathbb{R}^{1000}$.

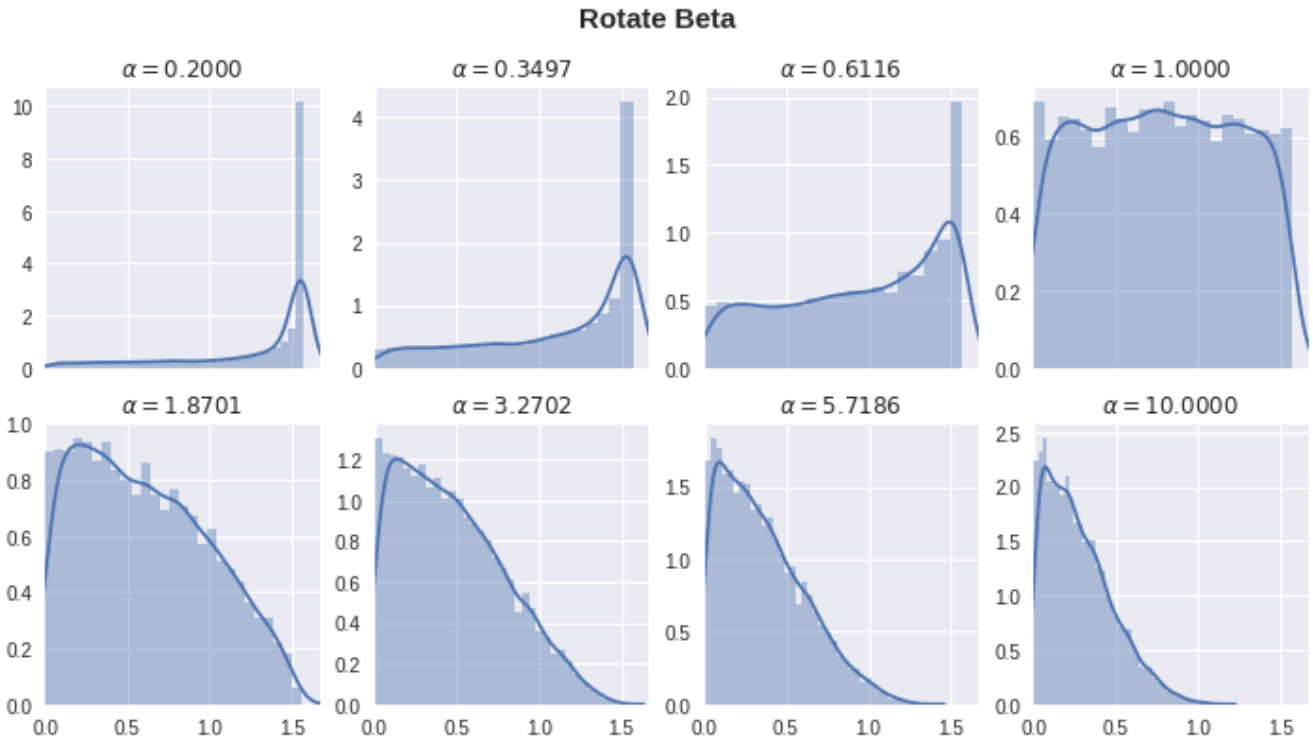

Figure 13: The distribution of angle between v_{norm} and $\hat{v}_{norm}$ for rotation-in-a-plane direction noise with stretched symmetric Beta distriibution. Each picture is obtained with 10000 samples of $\hat{v}_{norm}$ in $\mathbb{R}^{1000}$.

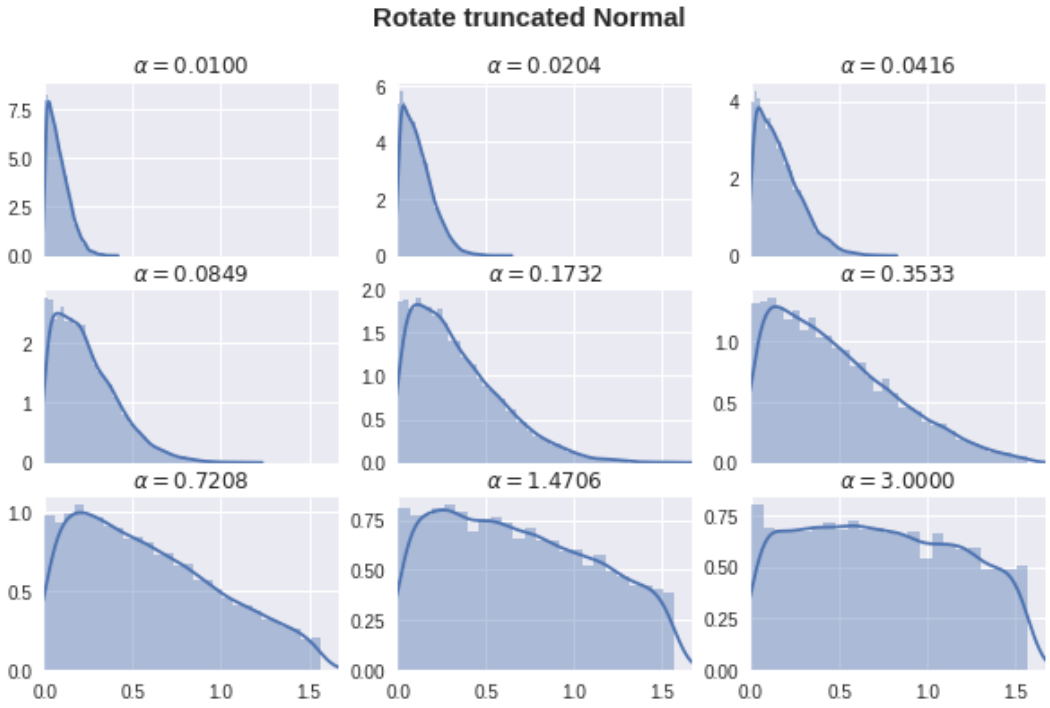

Figure 14: The distribution of angle between v_{norm} and $\hat{v}_{norm}$ for rotation-in-a-plane direction noise with truncated Normal distribution. Each picture is obtained with 10000 samples of $\hat{v}_{norm}$ in $\mathbb{R}^{1000}$.