Bayesian Neural Networks

Ilya Zharikov, Roman Isachenko, Artem Bochkarev

Abstract

The recent advances in deep learning allow us to solve really hard real word problems with high accuracy. What these methods lack is interpretability of the model and uncertainty guarantees for the predictions. Bayesian framework allows us to solve both these two issues, and with recent development in variational inference technique it is possible to take the best of two worlds and implement bayesian neural network. Our project is dedicated to testing different priors for neurons weights and exploring advantages and disadvantages of bayesian approach in deep learning.

Introduction

Deep learning is one of the most promising and quickly advancing areas of machine learning research [1]. Deep models proved to be superior to other algorithms in many fields such as image classification [2], speech recognition [3], unsupervised learning [4] and reinforcement learning [5]. Still, these models are mostly used as some black box in order to solve complicated problems. We often want to know not only the prediction of the network on given input, but we would like to understand the uncertainty [6] of the model on this object. It is also desirable to be able to do transfer learning [7], i.e. be able to obtain correct results on new types of data, using previous model as prior knowledge.

On the other side of the spectrum there is bayesian approach [8]. The main advantage of this framework is that we operate with parameters distribution, instead of their point estimation. This allows us to calculate uncertainties in predictions and representations. Another advantage of the bayesian approach is that we are free to select prior distribution for the parameters of our model [9], which may include real knowledge or some non-informative assumption. Finally, we may

want to build hierarchical models because data is too complex and good priors are hard to guess.

With recent advances in computational technology and GPU it is very compelling to try mixing deep learning with bayesian framework. This would allow us to achieve the same high performance of neural network, as well give us more freedom with different prior selection and interpretability of the results using uncertainty prediction. One of the obstacles for such approach is that full bayesian inference is very memory expensive, given huge parameters number (we need to store whole distribution instead of one point). What is more important, calculation of the model evidence is intractable for high-dimensional data. MCMC sampling methods [10] like Metropolis-Hastings or Gibbs are too computationally expensive to perform on big models.

In our project we decided to implement alternative way for approximation of posterior probability, namely ADVI [11]. The main idea of the method is that we constitute true posterior with some distribution from known parametric family and then we use stochastic optimization procedure to minimize their difference.

We conducted computational experiments on several datasets. We used PyMC3 [12] and Lasagne [13] for probabilistic and deep learning frameworks respectively. We used synthetic datasets as well as MNIST hand-written digits dataset. For these datasets we constructed bayesian neural network models, optimized them with ADVI, played with different priors and drew conclusions from model uncertainty.

Problem Statement

Bayesian approach

Suppose that $\mathbf{X}$ is the observed data which comes from the unknown distribution $p(\mathbf{X})$. This distribution defines our model and is called model evidence. We assume that our model also contains latent variables $\boldsymbol{\theta}$. The likelihood function is given by conditional probability distribution $p(\mathbf{X}|\boldsymbol{\theta})$. If we know the prior distribution $p(\boldsymbol{\theta})$ over hidden variables, we could use Bayes' theorem to derive the

posterior distribution $p(\boldsymbol{\theta}|\mathbf{X})$

$$p(\boldsymbol{\theta}|\mathbf{X}) = \frac{p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{X})}, \tag{1}$$

where the evidence function $p(\mathbf{X})$ could be obtained by integration over hidden variables

$$p(\mathbf{X}) = \int p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})p(\boldsymbol{\theta}).$$

The posterior distribution shows the transformed initial prior knowledge of $\boldsymbol{\theta}$ given the observed data $\mathbf{X}$.

The main goal of bayesian inference to compute the posterior distribution. In some cases we could derive the posterior in the closed form formula. However, it is impossible for complex models where the latent variables lies in high-dimensional space. The problem is the denominator of (1), since it is obtained by integrating over all possible values of $\boldsymbol{\theta}$.

Suppose that we have the posterior distribution $p(\boldsymbol{\theta}|\mathbf{X})$, then we could use the usual Maximum A Posteriori (MAP) approach to obtain point estimation of the parameters

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} \left[\ln p(\boldsymbol{\theta}|\mathbf{X})\right] = \arg\max_{\boldsymbol{\theta}} \left[\ln p(\mathbf{X}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta})\right].$$

In case of flat prior distribution $p(\boldsymbol{\theta}) = \text{const}$, this estimation coincides with the Maximum Likelihood Estimation (MLE) approach

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \left[\ln p(\mathbf{X}|\boldsymbol{\theta})\right].$$

Hierarchical modelling

There is one natural addition to previously described bayesian approach, which we will use in our project extensively, namely hierarchical modelling. If we look at Bayes theorem, we realize that the prior selection might be very important for model performance. This issue is more critical if we have little data or very complex model.

Hierarchical modelling solve this problem by allowing us to learn priors from the data itself. We consider prior distribution $p(\boldsymbol{\theta})$ of parameter $\boldsymbol{\theta}$ to be dependent

on some additional hyperparameter α. Following full bayesian approach, we need to introduce hyperprior $p(\alpha)$. Expression for posterior distribution becomes

$$p(\boldsymbol{\theta}, \alpha|\mathbf{X}) = \frac{p(\mathbf{X}|\boldsymbol{\theta}, \alpha)p(\boldsymbol{\theta}|\alpha)p(\alpha)}{p(\mathbf{X})}$$

Now we don't have to guess prior distribution, we can learn it from data directly.

Variational inference

One of the widely used approach to get posterior distribution is sampling methods such as Metropolis-Hastings, Gibbs, NUTS algorithms. The general idea behind sampling methods is to obtain procedure for generating samples from the true posterior distribution. Most of these methods are based on Monte Carlo Markov Chains (MCMC) procedures. The main disadvantage of such methods is that they are very slow for high-dimensional data.

In order to deal with high-dimensional data we could use analytical deterministic approximation of the posterior. Variational inference approach determines the function which is the closest to the desired posterior distribution. The distance is measured by Kullback-Leibler divergence

$$\mathrm{KL}(q||p) = -\int q(y) \ln \left[\frac{p(y)}{q(y)}\right] dy.$$

This distance function is non-negative and equals to zero iff $q \equiv p$

$$\mathrm{KL}(q||p) \geqslant 0; \quad \mathrm{KL}(q||p) = 0 \Leftrightarrow q \equiv p.$$

But the problem is that we don't know the true posterior and can't minimize the Kullback-Leibler divergence explicitly. Variational inference approach solves the

equivalent task which follows from the equation:

$$\ln p(\mathbf{X}) = \int q(\boldsymbol{\theta}) \ln p(\mathbf{X}) d\boldsymbol{\theta} = \int q(\boldsymbol{\theta}) \ln \left[\frac{p(\mathbf{X}, \boldsymbol{\theta})}{p(\boldsymbol{\theta}|\mathbf{X})} \right] d\boldsymbol{\theta} =$$
$$= \int q(\boldsymbol{\theta}) \ln \left[\frac{q(\boldsymbol{\theta}) p(\mathbf{X}, \boldsymbol{\theta})}{p(\boldsymbol{\theta}|\mathbf{X}) q(\boldsymbol{\theta})} \right] d\boldsymbol{\theta} = -\int q(\boldsymbol{\theta}) \ln \left[\frac{p(\boldsymbol{\theta}|\mathbf{X})}{q(\boldsymbol{\theta})} \right] d\boldsymbol{\theta} +$$
$$+ \int q(\boldsymbol{\theta}) \ln \left[\frac{p(\mathbf{X}, \boldsymbol{\theta})}{q(\boldsymbol{\theta})} \right] d\boldsymbol{\theta} = \mathrm{KL}(q||p) + \mathrm{ELBO}(q).$$

Here, the log model evidence function is decomposed into the Kullback-Leibler divergence and the Empirical Lower BOund (ELBO) term. Since the left hand side is independent of q function, the minimization of $\mathrm{KL}(q||p)$ is equivalent to the maximization of $\mathrm{ELBO}(q)$ which we can easily compute.

$$q^* = \arg\min_q \mathrm{KL}(q||p) = \arg\max_q \mathrm{ELBO}(q). \tag{2}$$

This lower bound can be expressed in the following way

$$\mathrm{ELBO}(q) = \int q(\boldsymbol{\theta}) \ln \left[\frac{p(\mathbf{X}, \boldsymbol{\theta})}{q(\boldsymbol{\theta})} \right] d\boldsymbol{\theta} =$$
$$= \int q(\boldsymbol{\theta}) \ln p(\mathbf{X}, \boldsymbol{\theta}) d\boldsymbol{\theta} - \int q(\boldsymbol{\theta}) \ln q(\boldsymbol{\theta}) d\boldsymbol{\theta} =$$
$$= \mathbb{E}_{q(\boldsymbol{\theta})} \left[\ln p(\mathbf{X}, \boldsymbol{\theta}) \right] - \mathbb{E}_{q(\boldsymbol{\theta})} \left[\ln q(\boldsymbol{\theta}) \right]. \tag{3}$$

The joint probability distribution $p(\mathbf{X}, \boldsymbol{\theta})$ is given by product of likelihood function $p(\mathbf{X})$ and prior distribution $p(\boldsymbol{\theta})$.

Automatic differentiation variational inference

In [11] the Automatic Differentiation Variational Inference (ADVI) method was proposed to solve the problem (2).

At the first stage ADVI method automatically transforms the original constrained variables $\boldsymbol{\theta}$ to the unconstrained real-valued variables $\boldsymbol{\zeta} = T(\boldsymbol{\theta})$. ADVI then defines the corresponding variational problem on the transformed variables, that is, to minimize $\mathrm{KL}\left(q(\boldsymbol{\zeta})||p(\boldsymbol{\zeta}|\mathbf{X})\right)$. With this transformation, all latent variables are defined on the same space. E. g. if some component θ of $\boldsymbol{\theta}$ are non-

negative ($\theta \in \mathbb{R}_+$), then one could use the logarithm transformation $\zeta = T(\theta) = \log(\theta) \in \mathbb{R}$.

The joint probability function for transformed latent variables equals

$$p(\mathbf{X}, \boldsymbol{\zeta}) = p(\mathbf{X}, T^{-1}(\boldsymbol{\zeta}))|\det J_{T^{-1}}(\boldsymbol{\zeta})|,$$

where $J_{T^{-1}}(\boldsymbol{\zeta})$ is the Jacobian of the inverse T transformation.

The lower bound (3) in real coordinate space is given by

$$\text{ELBO}(q(\boldsymbol{\zeta})) = \mathbb{E}_{q(\boldsymbol{\zeta})}\left[\ln p(\mathbf{X}, T^{-1}(\boldsymbol{\zeta})) + \ln|\det J_{T^{-1}}(\boldsymbol{\zeta})|\right] - \mathbb{E}_{q(\boldsymbol{\zeta})}\left[\ln q(\boldsymbol{\zeta})\right]. \quad (4)$$

The next stage of the ADVI algorithm includes stochastic optimization. Firstly, we assume that $q(\boldsymbol{\zeta})$ comes from fixed parametric family. It allows us to avoid variational optimization and replace it by parameter optimization. Usually ADVI uses $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ family with diagonal $\boldsymbol{\Sigma}$. The general positive-definite form of the covariance matrix $\boldsymbol{\Sigma}$ is also possible, but it leads to excessive computations. After this parametrization the problem (2) is equivalent to

$$\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^* = \arg\max_{\boldsymbol{\mu}, \boldsymbol{\Sigma}} \text{ELBO}(q).$$

The solution of this problem is obtained using stochastic optimization. However, we cannot directly use automatic differentiation on the ELBO. This is because the objective function is expectation over optimized parameters. To avoid this problem one could use elliptical standardization as reparametrization trick [14]. We convert the Gaussian variational approximation to the standard Gaussian $\boldsymbol{\eta} = S(\boldsymbol{\zeta})$

$$q(\boldsymbol{\eta}) = \mathcal{N}(\boldsymbol{\eta}|0, \mathbf{I}).$$

The standardization transforms the variational lower bound from(4) into

$$\text{ELBO}(q(\boldsymbol{\eta})) = \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta}|0,\mathbf{I})}\Big[\ln p\left(\mathbf{X}, T^{-1}(S^{-1}(\boldsymbol{\eta}))\right) + \\ + \ln|\det J_{T^{-1}}(S^{-1}(\boldsymbol{\eta}))|\Big] - \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta}|0,\mathbf{I})}\left[\ln q(\boldsymbol{\eta})\right].$$

Now the expectation is independent of the optimized parameters and we can easily compute gradients over $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

In the original paper a new adaptive step-size gradient sequence was proposed. This sequence is implemented in used frameworks. But it is of no interest for our research.

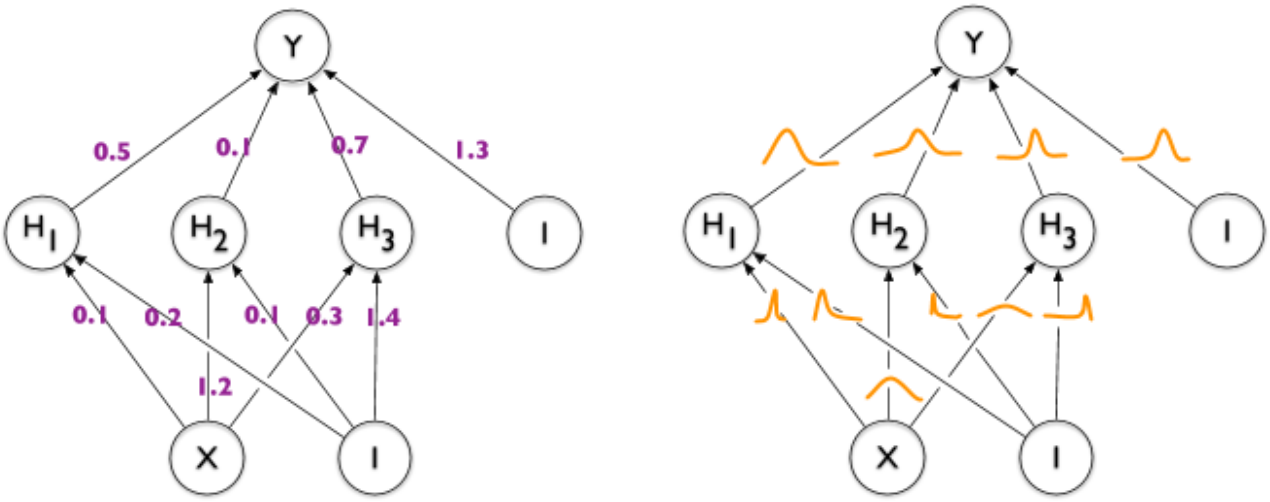
Bayesian neural networks

ADVI was implemented for deep learning models. In our project, neural network consists of several layers of neurons, outputs $\mathbf{x}_k$ of layer k depends on layer $k-1$ neurons $\mathbf{x}_{k-1}$ through some linear combination with coefficients $\mathbf{W}_{k-1}$, $\mathbf{b}_{k-1}$ and activation function g_{k-1}:

$$\mathbf{x}_k = g_{k-1}\left(\mathbf{W}_{k-1}\mathbf{x}_{k-1} + \mathbf{b}_{k-1}\right).$$

The usual deep learning approach is to use stochastic gradient descent and error backpropagation in order to fit the network parameters $\{\mathbf{W}_k, \mathbf{b}_k\}$, where k iterates over all network layers. The activation function such as *tanh*, *ReLU*, *softmax* allows to fit non-linear models.

We implemented bayesian approach, i.e. instead of point estimation, each parameter and each output in the network is given by probability distribution, the difference between two approaches can be seen on fig. 1. These probability distributions correspond to the priors of the bayesian model described above. The usual models tend to overfit in the case of high-dimensional parameter space. This is exactly the case with deep neural networks, where number of estimated parameters are several millions or more. There are some techniques to avoid this problem. The most widely used is regularization. Adding the term which restricts the model complexity allows to control parameters values. Introducing priors distributions is exactly the same procedure. The normal prior corresponds to $\ell 2$ regularization and the laplace prior — $\ell 1$ regularizer.



(a) Conventional neural network

(b) Bayesian neural network

Figure 1: Neural networks difference

Numerical experiments

Datasets

In our research we used synthetic data for binary classification problem in non-linearly separable case and real data such as MNIST[1] dataset for multiclass classification problem. One could find the detailed description of used data below.

Synthetic data

Synthetic data consists of two-dimensional binary classification datasets (moons and circles) which are challenging, since they are non-linearly separable.

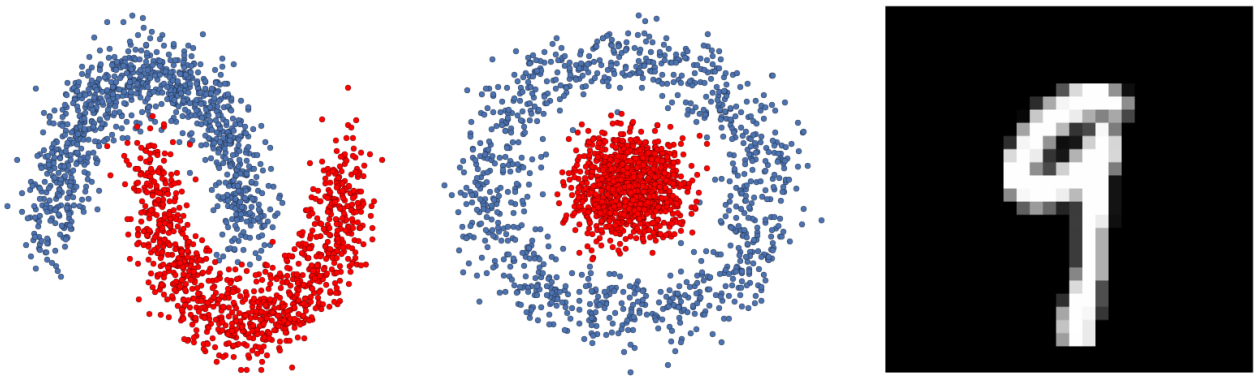
1. Moons: two interleaving half-circles (see fig.2 (a));
2. Circles: a large circle containing a smaller circle (see fig.2 (b)).

Real data

As for the real data MNIST dataset was selected. The data contains gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total (see fig.2 (c)). Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

[1] MNIST dataset — https://www.kaggle.com/c/digit-recognizer/data

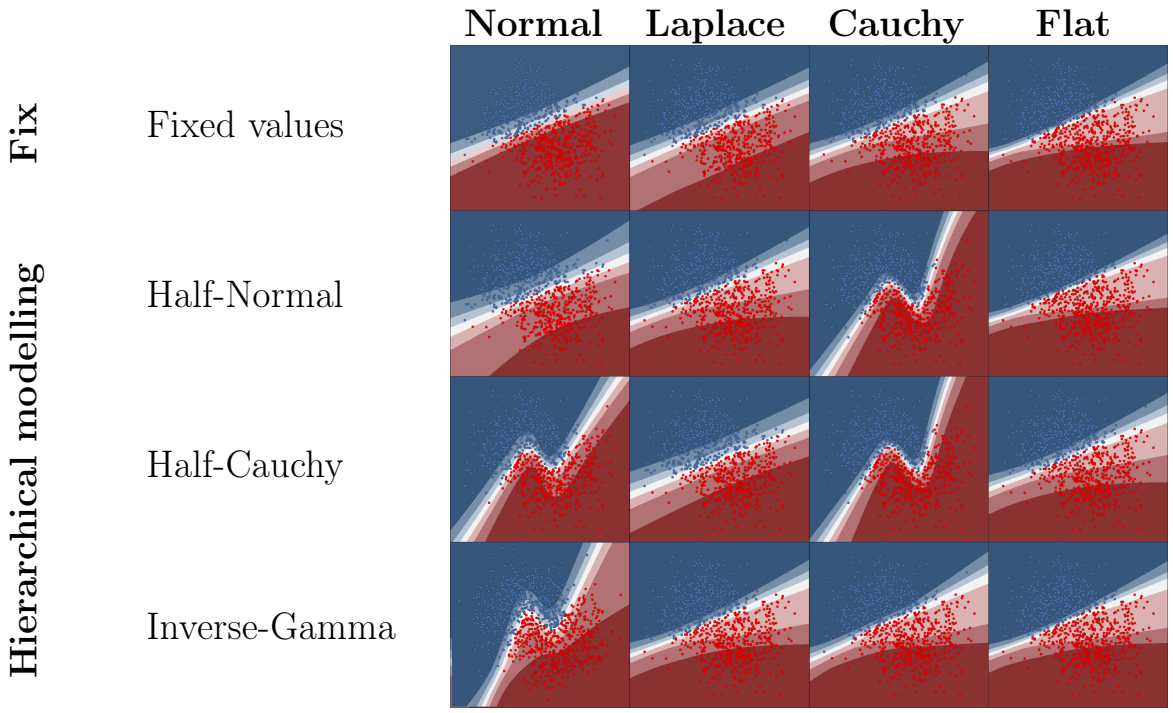


(a) Moons (b) Circles (c) MNIST

Figure 2: Synthetic and real data examples

Priors and hyperpriors influence

On the following figure one could see the design of the experiment for synthetic data.



Normal, laplace, cauchy and flat distributions were used as priors. We also tried hierarchical modelling approach. Fixed values of parameters, half-normal, half-cauchy and inverse-gamma distributions were used as hyperpriors for scaling parameters. Fixed values of hyperparameters correspond to the delta function hyperprior.

In this experiment we used the neural network with the following structure:

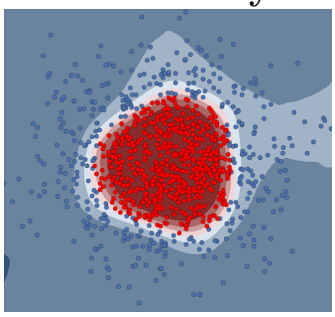
• Dense layer with 5 units and *tanh* activation;

- Dense layer with 5 units and *tanh* activation;
- Dense layer with 2 units and *softmax* as activation.

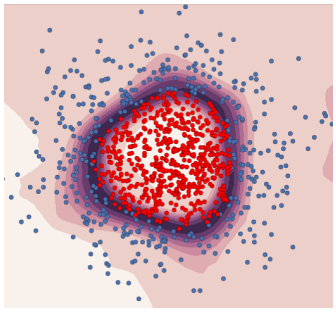
Different values of the noise parameter for data generation were tried. As expected it is hard to reconstruct the initial pattern with high noise parameter (the true class distribution could be seen in the fig.2). To detect the influence of priors and hyperpriors we decided to use the large noise parameter. In the picture below the best result for circles dataset is shown (700/300 samples for training/validation). Here the first plot illustrates the posterior probability for the first class. The second plot illustrates model uncertainty in predictions. The uncertainty is obtained as standard deviation of prediction over batch of samples from posterior distribution.

Prior: Cauchy
Hyperprior:
Inverse-Gamma
Accuracy: 0.735

Posterior Probability



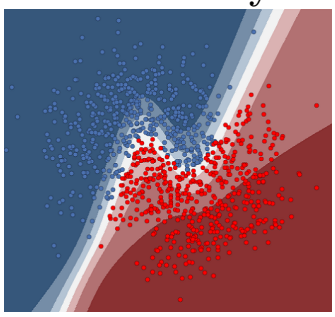
Uncertainty



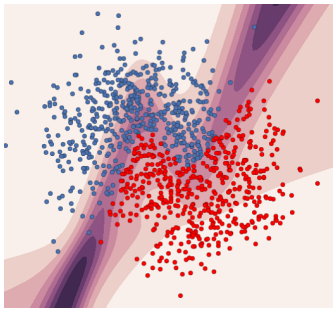
In the plots below one could see the same results for moons dataset. Both datasets shows that our model become uncertain about predictions near the true decision boundary.

Prior: Normal
Hyperprior:
Inverse-Gamma
Accuracy: 0.851

Posterior Probability

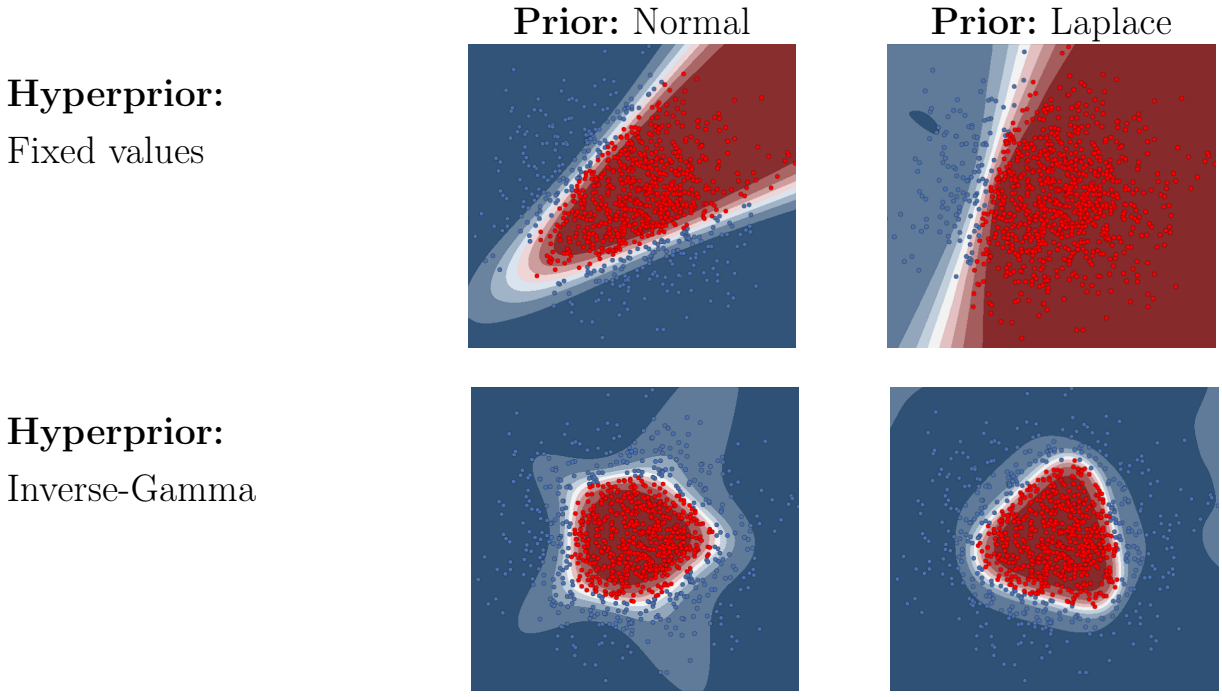


Uncertainty



Next experiment compares the results for fixed values of hyperparameters and hierarchical bayesian approach with introducing hyperpriors. The following pictures shows one of the example for circles dataset. The hyperpriors allows to

find the appropriate hyperparameters values for normal and laplace priors. The fixed values were tuned on the logarithmic grid.



Laplace prior is equivalent to $\ell 1$ regularazation which gives sparsity solution. Let us have a look at the distributions of the weights from one of the layers from the network. In the fig. 3 the posterior distributions of the last dense layer weights are illustrated. We have laplace prior on the left plot and cauchy on the right. All weights except two are distributed around zero for laplace prior. It leads to the sparse solution. In contrast, the cauchy prior gives the parameters without centering around zero.

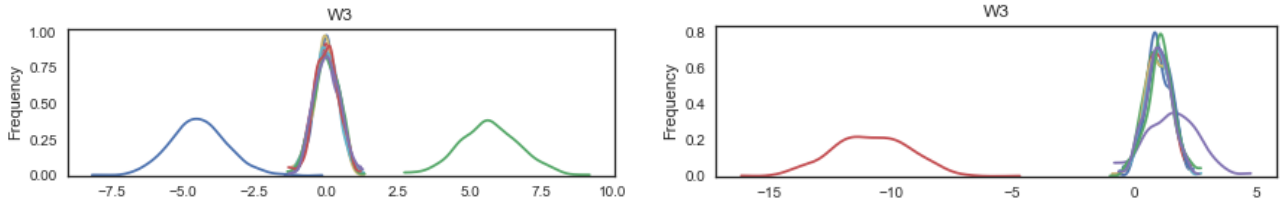


Figure 3: Estimated posterior distributions of the weights from the last layer for laplace prior (left), and cauchy prior (right)

To summarize all the results for synthetic datasets, we formulate several **conclusions:**

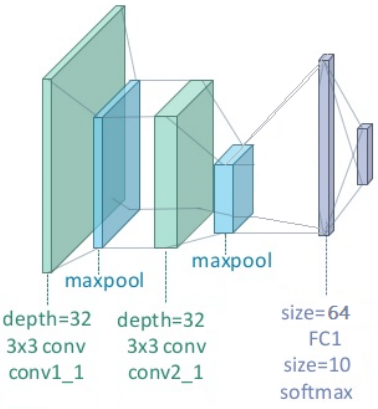
- variational inference allows to approximate posterior distribution;
- posterior distribution helps to make conclusions about uncertainties;

• hyperpriors are almost always give better results, especially on noisy data.

Uncertainty in predictions

In this experiment we used MNIST datasets (50000/10000 objects for training/validation). The main goal was to use ADVI method on real data with several classes and analyze the behaviour of the model. Here we used a network with the following structure:

- Convolution layer with 32 filters of the size 5×5 and *tanh* nonlinearity;
- Max pooling layer where pool size is equal to 2×2;
- Convolution layer with 32 filters of the size 5×5 and *tanh* nonlinearity;
- Max pooling layer where pool size is equal to 2×2;
- Dense layer with 64 units of the layer and *tanh* as activation function;
- Dense layer with 10 units of the layer and *softmax*.



Using the estimated posterior distributions of the considered neural network parameters we generated 100 samples from it and computed the output of the network. In the fig. 4 the first plot illustrates the variances of predictions over all samples. One could have these variances averaged for each class over misclassified objects (blue line), objects with correct prediction (green line) and all objects (red line). The variances for misclassified objects are much higher than for correct. Also one could see that for digits 2, 7, 8, 9 the results are more dispersed. The second plot shows the histogram of the expected error rate which expresses the fraction of samples for which the prediction is different from the mode of the posterior distribution.

The fig. 5 demonstrates the digits from validation sets for which the expected error rate equals zero, but the model makes mistake. It means that our model is overconfident about prediction, but it is not correct. There are exactly three of them.

The next picture (see fig. 5) shows three digits for which the expected error rate is the highest (about 0.7). For these objects our model is not confident at all. Also we show the true labels and model predictions for these examples.

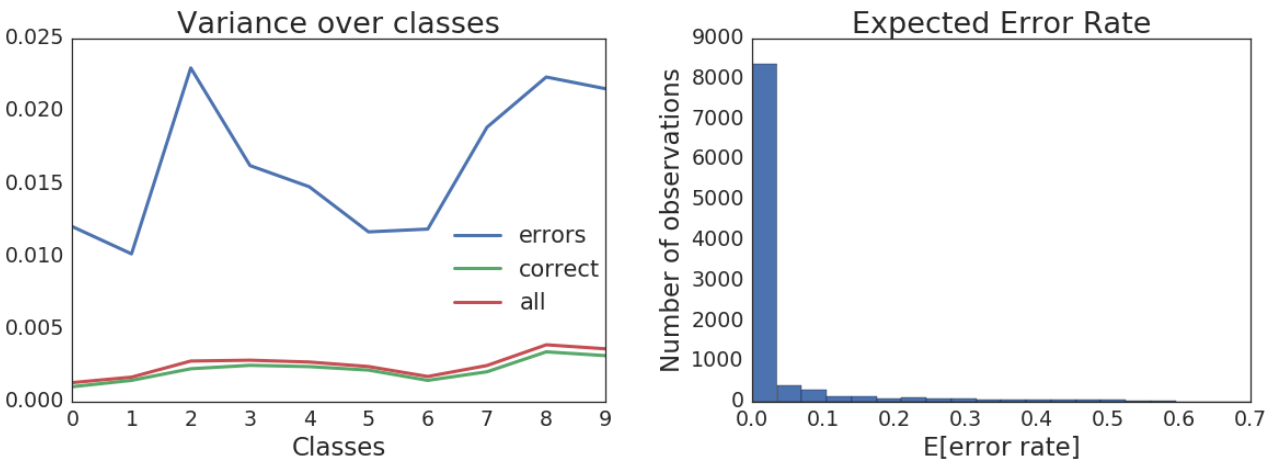


Figure 4: Variance over classes and expected error rate

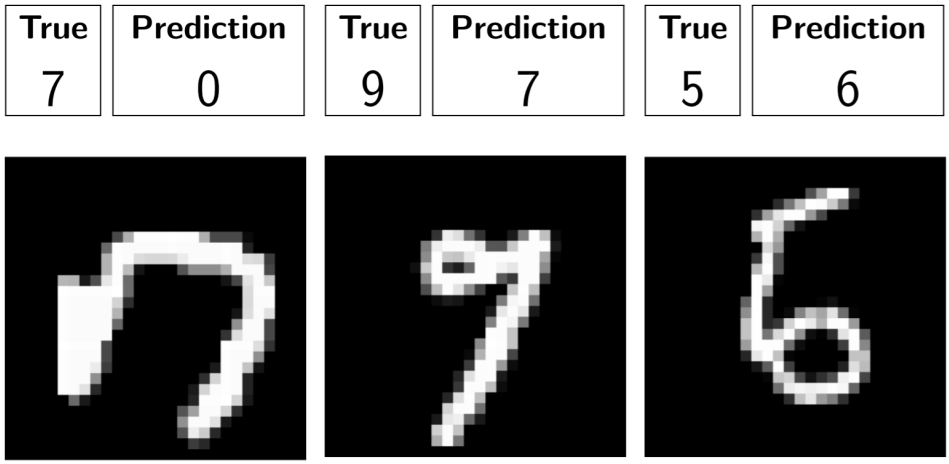


Figure 5: Misclassified pictures with zero expected error rate

The model accuracy for this dataset is 97.7%. We used the different setup for priors and hypepriors. But the final model examples shown above includes normal prior with fixed values of hyperparameters.

Transfer learning

The bayesian approach allows to use transfer learning approach to build a model on new object recognition data set using model, pretrained on another dataset. In order to implement this idea one could place informed priors centered around weights retrieved from this pretrained network. We tried this approach for two real datasets. The first one is CIFAR-10 dataset [2] which consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000

[2] CIFAR-10 dataset — https://www.cs.toronto.edu/ kriz/cifar.html

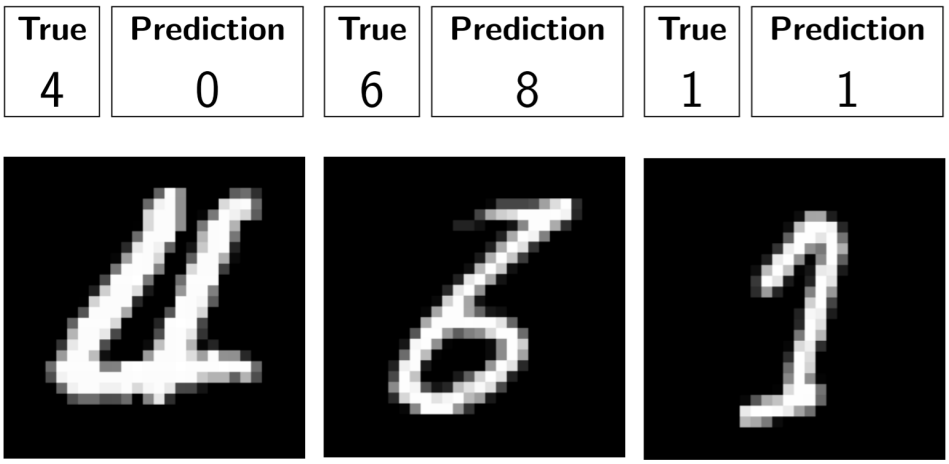


Figure 6: Pictures with the lowest confidence

training images and 10000 test images. The second one is Cats-vs-Dogs[3] which consists of 6000 colour images of cats and dogs with diferent sizes. There are 5000 training images and 1000 test images.

Firstly, we tried to train model with the network which were used for MNIST dataset. But we didn't get the good accuracy.

The pretrained model that we used in this experiment was VGG-19[4]. The architecture for VGG-19 model is given in fig. 7. We trained our models on AWS Amazon server[5] on g2.2xlarge. Unfortunately, our approach didn't have enough memory in this setup for both datasets. Then we tried to cut off last layers to reduce number of estimated parameters. When we got rid of all dense layers we managed to compile the model, but it would took days to fit the model. Since that, we didn't get the results for these datasets.

Conclusion

In this project we investigated bayesian approach for deep learning models. We read the latest papers on this issue and understood the theory behind them. We also wrote implementation of bayesian deep model on python and conducted experiments using both synthetic and real data. We played with different configurations and parameters of the model and we drew conclusions which illustrate advantages and disadvantages of presented methods.

[3] Cats-vs-Dogs dataset — https://www.kaggle.com/c/dogs-vs-cats
[4] VGG-19 model — http://www.robots.ox.ac.uk/∼vgg/research/very_deep/
[5] https://aws.amazon.com/

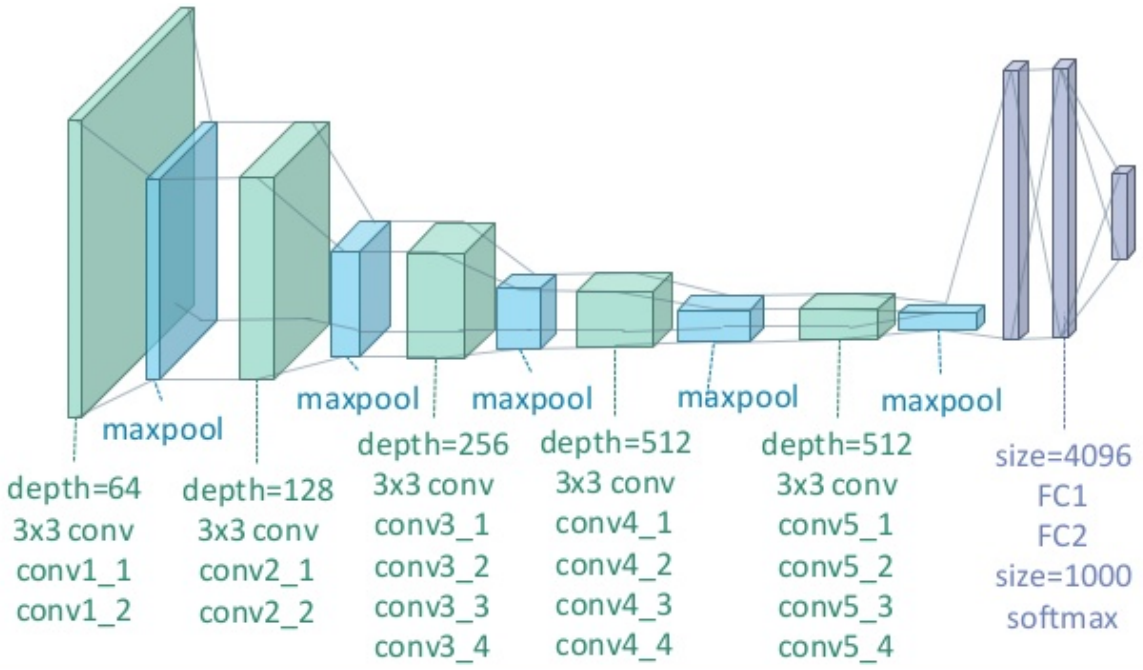


Figure 7: Architecture of VGG-19 neural network

Contributions

Ilya Zharikov programmed main framework for placing priors in the network, built neural networks architectures, implemented experiments on CIFAR-10 and cats-vs-dogs datasets.

Roman Isachenko extended the bayesian approach to hierarchical modelling, built VGG-19 network with transfer learning approach.

Artem Bochkarev conducted experiments and drew conclusions from synthetic data and MNIST dataset.

Everyone were equally involved in creating presentation, writing this report, as well as general project development.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

[4] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[6] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[7] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[8] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[9] William J Browne, David Draper, et al. A comparison of bayesian and likelihood-based methods for fitting multilevel models. *Bayesian analysis*, 1(3):473–514, 2006.

[10] Larry Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.

[11] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *arXiv preprint arXiv:1603.00788*, 2016.

[12] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55, 2016.

[13] Sander Dieleman, Jan Schluter, Colin Raffel, Eben Olson, et al. Lasagne: First release., August 2015.

[14] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.