Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning

Alpha Barry
Ecole Polytechnique
alpha-saliou.barry@polytechnique.edu

Mathieu Rita
Ecole Polytechnique
mathieu.rita@polytechnique.edu

Quentin Le Lidec
Ecole Polytechnique
quentin.le-lidec@polytechnique.edu

Abstract

Deep Learning allows to make predictions in a frequentist and very computationally efficient way. However it does not allow to measure uncertainty in general. On the opposite, Bayesian paradigm appears to be natural when it comes to evaluating confidence intervals but at the cost of a complex computation. Our work is based on Gal and Ghahramani [2016] and addresses the issue of evaluating uncertainty while preserving the efficiency of Deep Learning. By investigating an approximate variational inference of deep Gaussian Processes, we reveal the equivalence between this problem and the training of Neural Networks with dropout. This link allows to deduce an estimation of the variance of the predictions from the Neural Networks, while preserving a computational complexity of the same order of magnitude. In this report, we first detailed the equivalence of the problems exhibited in Gal and Ghahramani [2016], we reproduced the experimental results and finally went through some additional testing.

1 Deep Learning and Dropout

In this first part, we will describe the optimization problem solved when training a Neural Network and how it can be formulated when using dropout.

1.1 Risk minimization

From now on, we note $(W_i)_{i\in\{1,2\}}$, b and σ respectively the weights, the bias and the activation function of the Neural Network. Given an entry x, we note y the corresponding value and $\hat{y}$ the prediction from the Neural Network, which has the following expression :

$$\hat{y} = W_2\sigma(W_1x + b)$$

The goal of learning is to find the parameters W_i and b minimizing the Bayesian risk :

$$\mathrm{E}(\frac{||y - \hat{y}||^2}{2})$$

However, we don't know the the distribution of (X, Y) and the expectation cannot be computed. Instead, we minimize the empirical risk:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{||y_i - \hat{y}_i||^2}{2}$$

Preprint. Under review.

where (x_i, y_i) is the available data.
In order to avoid overfitting, we will penalize large parameters by adding their square norm to the objective function which has the following expression :

$$\frac{1}{N}\sum_{i=1}^{N}\frac{||y_i-\hat{y}_i||^2}{2}+\lambda\left(||W_1||^2+||W_2||^2+||b||^2\right) \tag{1}$$

1.2 Dropout: a regularization technique

Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons. It consists in , at training time, for each hidden layer, for each training sample and for each iteration to randomly put to zero a random fraction p of neurons. By noting $z_i^n \in \mathrm{R}^{\mathrm{h_i}}$ with h_i the width of the i^{th} layer, $z_i^n \sim$ Bernouilli(p_i) where $1-p_i$ is the fraction of neurons to dropout at the i^{th} layer, then the new prediction made by the Neural Network is :

$$\hat{y}_n = W_2\text{diag}(z_2^n)\sigma(W_1\text{diag}(z_2^n)x+b)$$

2 Dropout under the Bayesian paradigm

The objective of this section is to show that the computation of the posterior of a Gaussian process with the right Kernel and using variational inference is equivalent to the optimization problem of training of a Neural Network with dropout and regularization.

2.1 Generative model : Gaussian process

We assume that the data is composed of $(X_i, Y_i)_{i\in[1,N]}$ with the observed inputs $X \in \mathbf{R}^{NQ}$ and the observed ouputs $Y \in \mathbf{R}^{ND}$. We consider a Gaussian process as generative model with the following covariance function :

$$K(\mathbf{x},\mathbf{y}) = \int p(\mathbf{w})p(b)\sigma(\mathbf{w}^T\mathbf{x}+b)\sigma(\mathbf{w}^T\mathbf{y}+b)d\mathbf{w}d\mathbf{b}$$

The prior $p(\mathbf{w})$ is a standard multivariate distribution of dimension Q and $p(b)$ correspond to some distribution. With samples $(\mathbf{w_k})_{k\in[1,K]}$ of law $p(\mathbf{w})$ and $(b_k)_{k\in[1,K]}$ of law $p(b)$ we can approximate the kernel with Monte Carlo

$$\hat{K}(\mathbf{x},\mathbf{y}) = \frac{1}{K}\sum_{k=1}^{K}\sigma(\mathbf{w_k}^T\mathbf{x}+b_k)\sigma(\mathbf{w_k}^T\mathbf{y}+b_k)$$

We can use this approximation in the following generative model :

$$\begin{gathered}
\mathbf{w}_k \sim p(\mathbf{w}), b_k \sim p(b) \\
\mathbf{W}_1 = [\mathbf{w}_k]_{k=1}^K \in \mathbf{R}^{KQ}, \mathbf{b} = [b_k]_{k=1}^K \\
\hat{K}(\mathbf{x},\mathbf{y}) = \frac{1}{K}\sum_{k=1}^{K}\sigma(\mathbf{w_k}^T\mathbf{x}+b_k)\sigma(\mathbf{w_k}^T\mathbf{y}+b_k) \\
\mathbf{F}|\mathbf{X},\mathbf{W}_1,\mathbf{b} \sim N(0,\hat{K}(X,X)) \\
\mathbf{Y}|\mathbf{F} = N(\mathbf{F},\tau^{-1}I_N)
\end{gathered}$$

with $\hat{K}(X,X) = (\hat{K}(X_i,X_j))_{i\in[1,N],j\in[1,N]}$

2.2 Re-parametrization of the process

We denote by $\phi(x, \mathbf{W}_1, \mathbf{b}) \in \mathbf{R}^K$ the vector of features corresponding to the entry x and by $\Phi \in \mathbf{R}^{NK}$ the complete feature vector for all the data.

$$\phi(x, \mathbf{W}_1, \mathbf{b}) = \frac{1}{\sqrt{K}}\sigma(\mathbf{W}_1 x + \mathbf{b})$$
$$\mathbf{\Phi} = [\phi(X_i, \mathbf{W}_1, \mathbf{b})]_{i=1}^N$$

With $\mathbf{\Phi}$ we can rewrite the law of Y conditioned on $\mathbf{X}, \mathbf{W}_1, \mathbf{b}$

$$\mathbf{Y}|\mathbf{X}, \mathbf{W}_1, \mathbf{b} = \mathcal{N}(0, \mathbf{\Phi}\mathbf{\Phi}^T + \tau^{-1} I_N)$$

Bishop [2006] shows that we can introduce weigths $\mathbf{w}_d \in \mathbf{R}^K$ for each ouput d (each column of Y) and rewrite the previous part of the generative model as :

$$\mathbf{w}_d \sim \mathcal{N}(0, I_K)$$
$$\mathbf{W}_2 = [\mathbf{w}_d]_{d=1}^D \in \mathbf{R}^{DK}$$
$$\mathbf{Y}|\mathbf{X}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b} = \mathcal{N}(\mathbf{\Phi}\mathbf{W}_2^T, \tau^{-1} I_N)$$

The two generative models are equivalent. The previous generative model is obtained by adding new weights: one weight vector of dimension K (which multiply the feature vector ϕ), for each output (columns of Y). Knowing the weights $\mathbf{W}_1$, $\mathbf{W}_2$ and the bias $\mathbf{b}$, $\mathbf{Y}$ can be obtained by adding noise (of variance τ^{-1}) on the output of a neural network with one hidden layer (the vector of features ϕ) and d outputs. The resulting generative process is :

$$\mathbf{w}_k \sim p(\mathbf{w}), b_k \sim p(b), \mathbf{w}_d \sim N(0, I_K)$$
$$\mathbf{W}_1 = [\mathbf{w}_k]_{k=1}^K \in \mathbf{R}^{KQ}, \mathbf{b} = [b_k]_{k=1}^K, \mathbf{W}_2 = [\mathbf{w}_d]_{d=1}^D \in \mathbf{R}^{DK}$$
$$\mathbf{Y}|\mathbf{X}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b} = \mathcal{N}\left(\left[\frac{1}{\sqrt{K}}\mathbf{W}_2\sigma(\mathbf{W}_1 X_i + \mathbf{b})\right]_{i=1}^N, \tau^{-1} I_N\right)$$

It corresponds to the generative process of a Deep GP (Damianou and Lawrence [2012])

2.3 Approximation of the posterior with variational inference

The posterior of the previous generative model is in general intractable. We will then use variational inference in order to approximate the posterior. We want to minimize the KL divergence between the approximate posterior q and the real posterior $P(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}|(\mathbf{X}, \mathbf{Y}))$. For q we use the following distribution over matrices and a Gaussian distribution for the bias $\mathbf{b}$:

$$\mathbf{W}_1 = \mathbf{M}_1 \text{diag}([z_{1,j}]_{j=1}^Q)$$
$$\mathbf{W}_2 = \mathbf{M}_2 \text{diag}([z_{2,j}]_{j=1}^K)$$
$$z_{1,j} \sim \text{Bernouilli}(p_1)$$
$$z_{2,j} \sim \text{Bernouilli}(p_2)$$
$$\mathbf{b} \sim \mathcal{N}(\mathbf{m}, \sigma^2 I_K)$$

with

$$\mathbf{M}_1 \in \mathbf{R}^{KQ}, \mathbf{M}_2 \in \mathbf{R}^{DK}, \mathbf{m} \in R^K$$

as variational parameters. The columns of the matrix $\mathbf{M}_1$ and $\mathbf{M}_2$ are set randomly to 0. The units corresponding of the columns set to 0 are not taken into account during the computation. p_1 is then the parameter for the first layer and p_2 the parameter for the hidden layer. Minimizing the KL divergence between q and $P(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}|(\mathbf{X}, \mathbf{Y}))$ is equivalent to maximizing the ELBO :

$$L = \int q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}) \log(P(\mathbf{Y}|\mathbf{X}, \mathbf{W}_1, \mathbf{W_2}, \mathbf{b})) - \text{KL}(q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b})||p(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}))$$

The first term can be rewritten as the following sum :

$$\sum_{i=1}^{N} \int q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}) \log(P(\mathbf{Y}_i|\mathbf{X}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}))$$

We can approximate the integral with Monte Carlo with one sample $(\hat{\mathbf{W}}_1^i, \hat{\mathbf{W}}_2^i, \hat{\mathbf{b}}^i)$ of law q for each term of the sum. The sum is then approximated by :

$$\sum_{i=1}^{N} \log(P(\mathbf{Y}_i|\mathbf{X}_i, \hat{\mathbf{W}}_1^i, \hat{\mathbf{W}}_2^i, \hat{\mathbf{b}}^i))$$

The law of $\mathbf{Y}_i$ conditioned on $\mathbf{X}_i, \hat{\mathbf{W}}_1, \hat{\mathbf{W}}_2, \hat{\mathbf{b}}$ is a gaussian of mean $\hat{Y}_i = \frac{1}{\sqrt{K}} \hat{\mathbf{W}}_2 \sigma(\hat{\mathbf{W}}_1 \mathbf{X}_i + \hat{\mathbf{b}})$ and covariance matrix $\tau^{-1} I_D$ so we have

$$\begin{aligned}
\log(P(\mathbf{Y}_i|\mathbf{X}_i, \hat{\mathbf{W}}_1^i, \hat{\mathbf{W}}_2^i, \hat{\mathbf{b}}^i)) &= \sum_{d=1}^{D} [\log(\mathcal{N}(\mathbf{Y}_{id}; \hat{Y}_{id}, \tau^{-1})] \\
&= -\frac{D}{2} \log(2\pi) + \frac{D}{2} \log(\tau) - \frac{\tau}{2} \sum_{d=1}^{D} \left(\mathbf{Y}_{id} - \hat{Y}_{id} \right)^2 \\
&= -\frac{\tau}{2} ||\mathbf{Y}_i - \hat{Y}_i||_{\mathbf{R}^D}^2 + C
\end{aligned}$$

with C a constant.

The ELBO become :

$$-\frac{\tau}{2} \sum_{i=1}^{N} ||\mathbf{Y}_i - \hat{Y}_i||_{\mathbf{R}^D}^2 - \text{KL}(q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b})||p(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}))$$

The second term of the ELBO is the KL divergence between q and p. In order to approximate it, we will estimate it with a different distribution for q and different priors.

As priors, we take $p(\mathbf{w}) = \mathcal{N}(0, l^{-2} I_K)$ for the columns of $\mathbf{W}_1$ and $p(\mathbf{b}) = \mathcal{N}(0, l^{-2} I_K)$ for the bias. For the variational distribution q we choose the following distributions on the columns of $\mathbf{W}_1$ and the columns of $\mathbf{W}_2$ and we do not change the distribution of $\mathbf{b}$:

$$\begin{aligned}
q(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}) &= q(\mathbf{W}_1) q(\mathbf{W}_2) q(\mathbf{b}) \\
q(\mathbf{W}_1) &= \prod_{q=1}^{Q} q(\mathbf{w}_q) \\
q(\mathbf{w}_q) &= p_1 N(\mathbf{m}_q, \sigma^2 I_K) + (1 - p_1) N(0, \sigma^2 I_K) \\
q(\mathbf{W}_2) &= \prod_{k=1}^{K} q(\mathbf{w}_k) \\
q(\mathbf{w}_k) &= p_2 N(\mathbf{m}_k, \sigma^2 I_D) + (1 - p_2) N(0, \sigma^2 I_D) \\
q(\mathbf{b}) &= \mathcal{N}(\mathbf{m}, \sigma^2 I_K)
\end{aligned}$$

The means $\mathbf{m}_q$ correspond to the columns of $\mathbf{M}_1$ and the means $\mathbf{m}_k$ correspond to the columns of $\mathbf{M}_2$. The interest of taking these forms for the prior and the variational distribution is that we can now estimate the KL term (ignoring the σ term because it is a constant) with (see the appendix of Gal and Ghahramani [2016] for details) :

$$\sum_{i=1}^{2}\left(\frac{p_i l^2}{2}||\mathbf{M}_i||_2^2\right) + \frac{l^2}{2}||\mathbf{m}||_2^2$$

To come back to the original setting, we just have to take $\sigma = 0$. Moreover, when σ tends towards 0, the previous distribution on the columns of $\mathbf{W}_1$ tends towards a Bernoulli distribution for which each column $\mathbf{w}_q$ equals to $\mathbf{m}_q$ with probability p_1 and 0 with probability $1 - p_1$. It is the same for $\mathbf{W}_2$ and p_2. It corresponds exactly to the original setting on q.

The previous approximation is still valid when we take σ close to 0 (because we ignored the σ term) and the distribution of $\mathbf{b}$ becomes a Dirac distribution on $\mathbf{m}$ and $\mathbf{b}$ become constant. As we have seen before, the mixture of Gaussians used previously for the computing of the KL divergence tends towards the previous Bernoulli distribution used for q when σ tends towards 0. Maximizing the Evidence Lower Bound is then equivalent to the following problem (when σ tends towards 0):

$$\min \frac{1}{2N}\sum_{i=1}^{N}||\mathbf{Y}_i - \hat{Y}_i||_{\mathbf{R}^D}^2 + \sum_{i=1}^{2}\left(\frac{p_i l^2}{2N\tau}||\mathbf{M}_i||_2^2\right) + \frac{l^2}{2N\tau}||\mathbf{m}||_2^2$$

with

$$\hat{Y}_i = \frac{1}{\sqrt{K}}\hat{\mathbf{W}}_2\sigma(\hat{\mathbf{W}}_1\mathbf{X}_i + \mathbf{m})$$

With the parameters p_1 and p_2 both equal to p and a choice of $\tau = \frac{pl^2}{2N\lambda}$ we finally have exactly the objective of the first part (1).

We should notice that the equivalence between the problem of training a Neural Network with dropout and the computation of the posterior of a Deep Gaussian Process is guaranteed only in the case were the loss function of the first problem contains an L2 penalty term. Thus, a priori, there is no evidence to think that this equivalence is still valid in a more general way, without the penalty term. In practice, this can be determinant because the L2 penalty is rarely used as regularization technique in Deep Learning, which limits the scope of the result previously demonstrated.

3 Dropout to measure uncertainty

In this section, we will investigate how the Bayesian point of view on Dropout, detailed previously, allows to deduce a measure of uncertainty on predictions made by a Neural Network using only the forward passes of the input.

3.1 Predicting with the Monte Carlo Dropout estimate

After the variational inference detailed previously, we have an approximation of the posterior distribution of the weights of the Neural Network $q(w)$ where $w = \{W_i\}_i$, and thus, we can deduce the posterior distribution of the output y conditioned on the input x:

$$q(y|x) = \int p(y|x,w)q(w)\mathrm{d}w$$

Here, the randomness on w comes from the fact that Dropout randomly puts to 0 a fraction $1 - p_i$ of the lines of the weight matrix W_i and this is why the output y resulting from a forward passes in the Neural Network is also a random variable. Then, when it comes to making a prediction, we can take the expected value of y:

$$\hat{y} = \mathrm{E}_{q(y|x)}(y)$$

and from the previous part we have $q(y|x,w) = p(y|x,w)q(w)\mathrm{d}w$, $p(y|x,w) = \mathcal{N}(y|\hat{y}(x,w), \tau\mathrm{Id})$, and we showed that $w \approx (W_i\mathrm{diag}(z_i))$ and so we have $q(w) \approx \mathrm{Bernouilli}(z_1)\mathrm{Bernouilli}(z_2)$ and thus :

$$\begin{aligned}
\hat{y} &= \mathrm{E}_{q(y|x)}(y) \\
&= \int y q(y|x)\mathrm{d}y \\
&= \int y\mathcal{N}(y|\hat{y}(x,w), \tau\mathrm{Id})\mathrm{Bern}(z_1)\mathrm{Bern}(z_2)\mathrm{d}z_1\mathrm{d}z_2\mathrm{d}y \\
&= \int \left(\int y\mathcal{N}(y|\hat{y}(x,w), \tau\mathrm{Id})\mathrm{d}y \right) \mathrm{Bern}(z_1)\mathrm{Bern}(z_2)\mathrm{d}z_1\mathrm{d}z_2 \\
&= \int \hat{y}(x, W_1, W_2)\mathrm{Bern}(z_1)\mathrm{Bern}(z_2)\mathrm{d}z_1\mathrm{d}z_2 \\
&\approx \frac{1}{T}\sum_{i=1}^{T} \hat{y}(x, W_1^t, W_2^t)
\end{aligned}$$

where the expected value has been replaced by a Monte Carlo approximation. This estimated will be called the MC Dropout estimate.

Practically, the MC Dropout estimate consists in making several forward passes through the network with dropout and averaging the outputs to make a prediction. As explained previously, this prediction is a random variable due to the randomness of Dropout, and, depending on the application, it can be determinant to be able to estimate its uncertainty. However, MC Dropout is not the classical way to make predictions after training a Neural Network with Dropout. Indeed, the standard prediction consists in making only one forward pass without dropping any weight of the network and by multiplying W_i by p_i. As shown by Srivastava et al. in Srivastava et al. [2014], he standard prediction can be seen as an approximate of MC Dropout and requires less computation.

3.2 Measuring the uncertainty of the prediction

As we explained before, the MC Dropout estimate is a random variable and it can be interesting to have a measure of its uncertainty. A natural way of measuring it and proposed in Gal and Ghahramani [2016] is to evaluate its variance :

$$\mathrm{Var}_{q(y|x)}(\hat{y}) = \mathrm{E}_{q(y|x)}(\hat{y}^T\hat{y}) - \mathrm{E}_{q(y|x)}(\hat{y})^T\mathrm{E}_{q(y|x)}(\hat{y})$$

In our case, we supposed that $q(y|x,w) = p(y|x,w)q(w)\mathrm{d}w$, $p(y|x,w) = \mathcal{N}(y|\hat{y}(x,w), \tau\mathrm{Id})$, and we showed that $w \approx (W_i\mathrm{diag}(z_i))$ and so we have $q(w) \approx \mathrm{Bern}(z_1)\mathrm{Bern}(z_2)$. Thus, we can express the second raw moment:

$$\begin{aligned}
\mathrm{E}_{q(y|x)}(\hat{y}^T\hat{y}) &= \int \left(\int \hat{y}^T\hat{y}p(\hat{y}|x,w)\mathrm{d}\hat{y} \right) q(w)\mathrm{d}w \\
&= \int \left(\mathrm{Cov}_{p(\hat{y}|x,w)}(\hat{y}) + \mathrm{E}_{p(\hat{y}|x,w)}(\hat{y})^T\mathrm{E}_{p(\hat{y}|x,w)}(\hat{y}) \right) q(w)\mathrm{d}w \\
&= \int \left(\frac{1}{\tau}\mathrm{Id} + \hat{y}(x, W_1, W_2)^T\hat{y}(x, W_1, W_2) \right) \mathrm{Bern}(z_1)\mathrm{Bern}(z_2)\mathrm{d}z_1\mathrm{d}z_2 \\
&\approx \frac{1}{\tau}\mathrm{Id} + \frac{1}{T}\sum_{i=1}^{T} \hat{y}(x, W_1^t, W_2^t)^T\hat{y}(x, W_1^t, W_2^t)
\end{aligned}$$

Thus, we finally have an expression of the variance of $\hat{y}$:

$$\mathrm{Var}_{q(y|x)}(\hat{y}) = \frac{1}{\tau}\mathrm{Id} + \frac{1}{T}\sum_{t=1}^{T} \hat{y}(x, W_1^t, W_2^t)^T\hat{y}(x, W_1^t, W_2^t) - \mathrm{E}_{q(y|x)}(\hat{y})^T\mathrm{E}_{q(y|x)}(\hat{y})$$

where $\mathrm{E}_{q(y|x)}(\hat{y})$ can be approximated by the MC Dropout estimate. We can notice this expression corresponds to the empirical variance of T forward passes with an additionnal term $\frac{1}{\tau}\mathrm{Id}$. Moreover,

the equivalence of problems shown in the second part revealed that we have to take:

$$\tau = \frac{pl^2}{2N\lambda}$$

It is important to note that the estimation of the variance only requires the results of T forward passes through the Neural Network to be computed. Thus, there is no need of extra computation to obtain a measure of uncertainty when using the MC Dropout estimate to make a prediction. However, it still requires T forward passes while the standard prediction from a network trained with dropout requires only one forward pass. Thus, the computation of variance proposed in Gal and Ghahramani [2016] would require a bit of extra computation compared to when using the standard prediction. We tested this last point on a classification task on MNSIT and an implementation on Pytorch. It revealed that a forward pass through a network with Dropout is even slower than a forward pass with all the weights being active in the same network. Thus, we notice that estimating the variance as suggested in the article comes at a price of a slightly less efficient computation.

Moreover, estimating the variance of the prediction made by the estimator allows to compare the relative uncertainty of the different predictions made by this estimator. However, it does not correspond to real confidence intervals, which are more interesting in practice. Indeed, in order to have a confidence interval in the Bayesian paradigm, the distribution of $p(y|x, \mathbf{X}, \mathbf{Y})$ must be known. However, without approximation, this distribution cannot be computed. In Gal and Ghahramani [2016] this distribution is approximated by a Gaussian distribution and then two standard deviations are used to construct confidence intervals.

4 Experiments

Now let's analyze the properties of the uncertainty estimates of the Dropout network on simple cases. Our aim is to see how the Dropout NN allows us to build uncertainty intervals. We assess two types of tasks: classification and regression.

4.1 Implementation details

To perform the experiments, we built a small `Pytorch` pipeline. We restricted our study to the class of networks with linear layers (each of them separated by dropout layers). For the regression task, we worked with 5 datasets that are used in the reference paper (bostonHousing, energy, naval-propulsion-plant, power-plant, wine-quality-red). For the classification task, we used the classical MNIST dataset (contrarily to the paper, we still used Linear layers with MNIST. We flattened the data images in order to get vectors). Then, all the experiments are presented in the notebook provided in the Github repository.

Methodology: In the following sections, we are going to present the results obtained with $\sim$ 20 different network settings. During the training, we used the negative log-likelihood for the classification loss, the Mean-Square Error for the regression loss and Adam optimizer for all the networks. A weight decay is also added to the loss for a regularization purpose (L_2 penalization). Before analyzing the effects of the problem parameters (τ, λ...), we tried different values of learning rate, number of epochs and network architecture to be sure to reach the convergence and get a correct accuracy (step not shown in the notebook). Then, we assigned a triplets (`layer_sizes`, `l_rate`, `n_epochs`) for each dataset.

4.2 Results on the classification task

First let's analyze the results we got on the classification task (Mnist dataset). After training, our network outputs a probability distribution over the 10 classes. In classical prediction tasks, we do not care about the values output by the network and merely take the class that get the highest score. Nevertheless, in the case of the Dropout NN (with active dropout layers during the testing step), there is a random effect introduced by the dropout layers. Therefore, each of the score assigned to a class is a random variable for which we can evaluate the empirical moments. The aim of this experiment is to see whether we can observe and quantify the uncertainty brought by the Dropout layers.

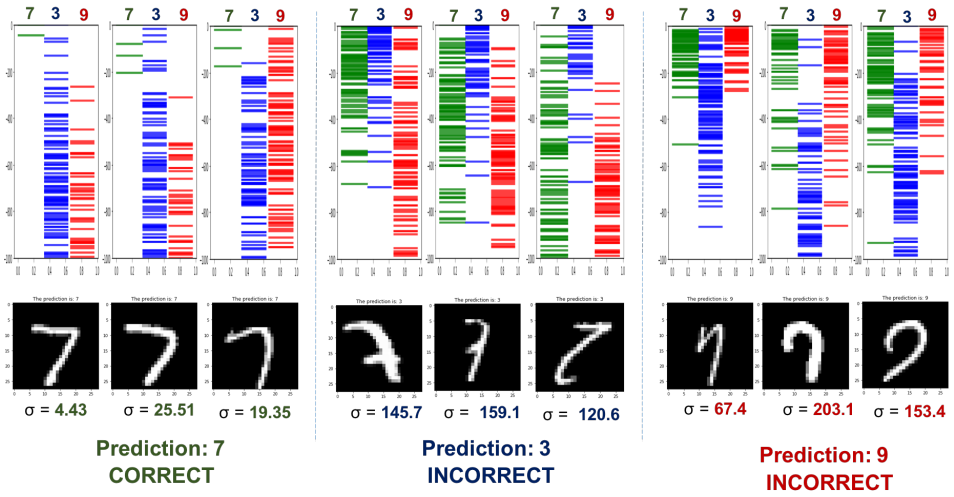

Figure 1: Results after $T = 100$ passes of the scores obtained by the classes 7 (green), 3 (blue) and 9 (red) on 9 images labelled 7. Each line corresponds to a score. The three first images are data well assigned to 7 by the network ; The three following images are data well assigned to 3 by the network ; The three first images are data well assigned to 9 by the network

To do so, we analyzed the predictions of the images that correspond to the number 7. We first looked at the most common mistakes made by the NN and saw that the NN mostly confound the class 7 with the classes 3 and 9. Then, among all the images labelled 7, we chose 3 images that were correctly predicted by the NN, 3 images that were assigned to the class 3 and 3 images that were assigned to the class 9. Then we fed the Dropout NN with the 9 images and ran $T = 100$ forward passes. In the end, for each of the 9 images, we stored the results of these 3 classes after the 100 passes. The purpose of the experiment is to see whether we can suspect an error from the network by computing kind of confidence intervals.

On figure 1, We plotted the scores (the higher is the score, the more likely a class is chosen) obtained by each of the 3 classes for the 9 images (each line correspond to a score). The three first images correspond to the correct predictions (7 assigned to 7) whereas the 6 following cases are wrong predictions (3 images assigned to 3 and 3 images assigned to 9). Under the 9 images, we put the empirical variance of the class that has been predicted by the classifier.

As we can see, when the prediction is correct (3 first cases), the score obtained by the right class (green lines) is always very high and the variance of the values obtained is small (<30). On the contrary, when the network is predicting the wrong class (6 other cases), we see that the standard deviation of the predicted class is very high and therefore we cannot be confident with the result. Moreover, we see that the "confidence interval" of this class overlaps the "confidence intervals" of the other classes.

In conclusion, for a classification task, we see that using the dropout Network allows us to get some information about the confidence of the network. For a wrong prediction, even though the score of the predicted class is high, by doing several forward passes, we see that we can estimate a kind of confidence interval.

4.3 Results on the regression task

After this experiment, we analyzed the results obtained on regression tasks. Such as for the classification task, the Dropout network allows us to evaluate the uncertainty by performing several forward passes with our data. Instead of drawing and computing kind of confidence intervals, we tried in these experiments to evaluate the predictive performance of the network, in order to comment the

results provided by the paper. The predictive performance can be evaluated with the RMSE and the predictive log-likelihood.

Following the Bayesian interpretation of dropout, we first need to define a prior length-scale l, and find an optimal model precision parameter τ in order to evaluate the predictive log-likelihood. As it is done in the paper, we fixed l to 0.01. Then, we performed a grid search over the `dropout_rate` and τ in order to find the best model. Once the grid search is performed, we select the model that is getting the best predictive log-likelihood (ie. the highest).

We store in table 1 the results we got on 5 different datasets. This table also displays the results presented in the paper and those obtain by other papers:

RMSE

Dataset	Dropout (our imp.)	Dropout (paper)	VI	PBP
BostonHousing	**2.85**	2.97	4.32	3.01
energy	2.56	**1.66**	2.65	1.80
naval-propulsion	**0.01**	**0.01**	**0.01**	**0.01**
power-plant	4.40	**4.02**	4.33	4.12
wine-quality-rd	0.63	**0.62**	0.65	0.64

Predictive log-likelihood

Dataset	Dropout (our imp.)	Dropout (paper)	VI	PBP
BostonHousing	**-2.49**	**-2.46**	-2.90	-2.57
energy	-2.39	**-1.99**	-2.39	-2.39
naval-propulsion	**-1.72**	3.80	3.73	3.73
power-plant	-3.63	**-2.80**	-2.89	-2.84
wine-quality-rd	-1.76	**-0.93**	-0.98	-0.97

Table 1: Test performance for our implementation of the dropout NN, the implementation of the paper, the variational inference method (VI, Graves [2011]), the Probabilistic back-propagation method (PBP, Hernandez-Lobato and P. [2015]).

Here are some comments about the results we got:

- The results of our implementation have a similar magnitude to those obtained by the paper. Therefore, we see that the results are reproducible and do not depend on the implementation (the paper experimented Theano and Tensorflow implementations of the Dropout NN).
- Nevertheless, for almost all the datasets, we did not reach as good results as those presented in the paper (ie. a small RMSE and a high predictive LL). This can be explained by the fact that we performed the grid search on a limited number of parameters. Moreover, we did not use exactly the same architecture and learning parameters (architecture, learning rate, ...). Moreover , the author suggests that because of the small size of the data, the splitting of the dataset may be important and different splitting could lead to non-comparable results. In the paper, they average their results on 20 splits while here we tested only one random splitting. Therefore, our work lacks a reproducible error analyzes. Nevertheless, our purpose was mainly to verify that the Dropout Network can lead to predictive performance comparable to state-of-the-art methods. For the few experiences we made, it seemed to be the case.

5 Conclusion

In their article, Gal and Ghahramani [2016], showed that the problem solved during the training of a Neural Network of arbitrary depth with dropout and with an L_2 penalty on weights is equivalent to an approximate computation of the posterior of a Deep Gaussian Process using variational inference.

From this Bayesian point-of-view, it is possible to build an estimator,the MC Dropout estimate, that allows to make predictions but also to compute the variance of this estimator. Compared to classical Bayesian techniques that allow to obtain confidence intervals, the computation of the variance of the prediction gives a quick estimation of the uncertainty without requiring a significant additional computation.

References

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Andreas C. Damianou and Neil D. Lawrence. Deep gaussian processes, 2012.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. pages 1050–1059, 2016.

Graves. A. practical variational inference for neural networks. *NIPS*, 2011.

J M Hernandez-Lobato and Adams R P. Probabilistic backpropagation for scalable learning of bayesian neural networks. *ICML*, 15, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.