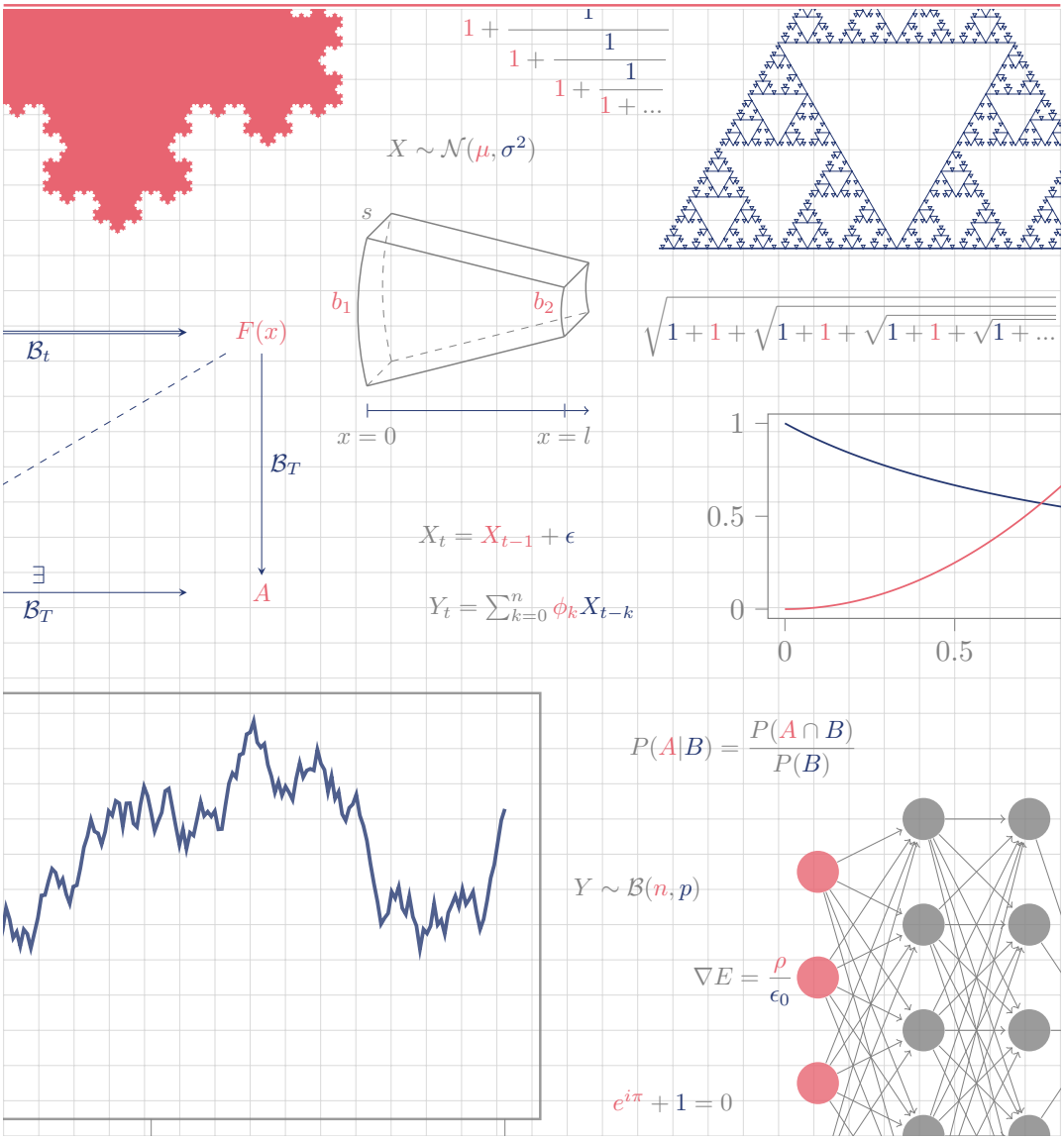
DataGenius

LITERATURE REVIEW OF BAYESIAN DEEP LEARNING

Robin CAMARASA

August 14, 2023



Contents

Deep Learning

1.1 Artificial Neural Networks

1.1.1 Introduction

Context : Deep Learning is a part of Machine Learning which is a field of Artificial Intelligence. Machine Learning consists in a study of algorithms that achieve a task without any explicit instruction. As part of Machine Learning, a Deep Learning model performs this task using Artificial Neural Networks.

Evolution : Artificial Neural Networks were intuited in 1943 by Warren McCulloch and Walter Pitts and popularised by Yann Lecun[10] thanks to their applications in computer vision. Nowadays, from start-up to GAFA, many companies invest resources in this innovative field.

Supervised/Unsupervised : To train a Machine Learning model data are required. A supervised learning model have the raw data and and the expected answer of the model on those raw data whereas an unsupervised learning algorithm only have raw data.

1.1.2 Basic Neural Network

Vanilla Neural Network : A neural network is ensemble of neurons organised in layers and linked with weights. The number of layers and units per layers are hyperparameters of the model. Please find an example of a simple neural network in figure 1.1 page 2.

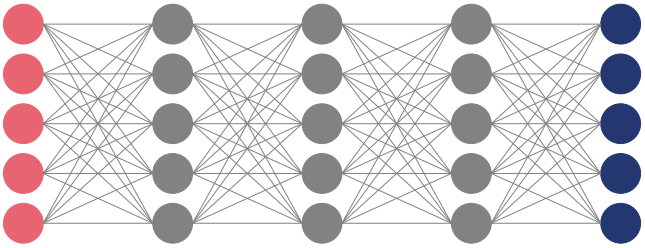


Figure 1.1: Vanilla Neural Network

Layer : As one can see in figure 1.1 page 2, the network is organised in layers. The input layer (pink neurons) holds the normalised value of the raw data. The output layer (blue neurons) holds the value of the prediction made by the network.

Neuron : A neuron is composed of an aggregation function (usely scalar product) and an activation function (ReLu, Tanh, sigmoid ...). The aggregation function of a neural of the i^{th} layer is linked with weights to the output of the neurons of the $i-1^{th}$ layer. Neurons of the first layer does not have aggregation function. The activation function rescale the output and gives non-linearity property to the model. Please find an example of a simple neuron in figure 1.2 page 3.

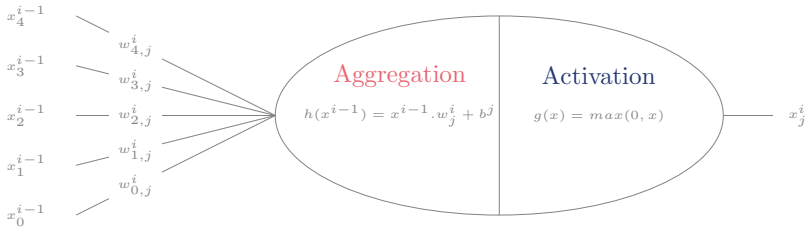


Figure 1.2: Simple neuron

1.1.3 Propagations

Forward propagation : To obtain the output layer with the input, a composition of aggregation and activation functions have to be computed. Please find this composition of functions at the equation 1.1 page 3. The forward propagation is the decomposition of the equation 1.1 page 3 with the equation 1.2 page 3.

$$x^n = g_n \circ h_n \circ ... \circ g_1 \circ h_1(x^0) \tag{1.1}$$

where x^n is the output of the network, x^0 the input of the network, g_i i^{th} layer activation function and h_i i^{th} layer aggregation function.

$$x^j = g_j \circ h_j(x^{j-1}) \tag{1.2}$$

where x^j is the values hold by the j^{th} layer, g_j j^{th} layer activation function and h_j j^{th} layer aggregation function.

Backpropagation : Backpropagation consists in a gradient descent adapted to Neural Networks, to obtain the weights that give the best predictions. The gradient descent is computed on the loss function, a function to minimise that takes the prediction and the ground truth as input (example: mean squared error). Weight values are updated with the equation 1.3 page 3. Backpropagation principle is illustrated in the figure 1.3 page 4. In practice, data scientists train their networks with algorithms based on backpropagation such as Adadelta[16].

$$w_{i,j}^k = w_{i,j}^k - \lambda \frac{\partial L(y,t)}{\partial w_{i,j}^k} \tag{1.3}$$

where λ is the learning rate, $w_{i,j}^k$ the weight linking the i^{th} neuron of the $(k-1)^{th}$ layer with j^{th} neuron of the k^{th} layer, L the loss function, t the prediction of the network and y the ground truth.

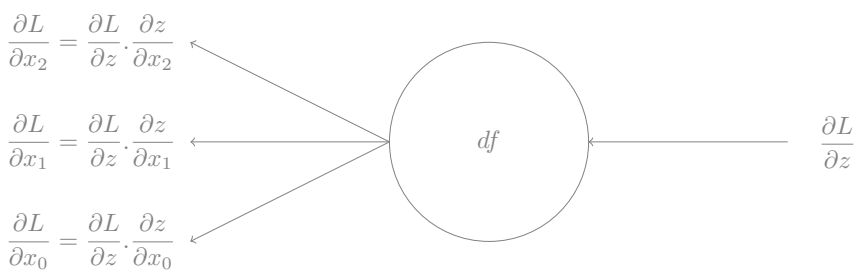


Figure 1.3: Backpropagation principle

1.1.4 Other Supervised Learning Network

CNN : CNN stands for Convolutional Neural Network. This kind of network is obtained using convolution product as aggregation function. This kind of networks is very efficient in image analysis tasks.

RNN : RNN stands for Recurrent Neural Network. When data display a recurrence in their structure (example: Time Series), using Recurrent Neural Networks is a wise choice. The two main RNN architectures are Gated Recurrent Units[5] (GRU) and Long Short Term Memory[7] (LSTM).

1.2 Standard Practices

1.2.1 Model Training and Selection

Splitting data : To train a model, one splits the dataset in 3 subset, the training set, the validation set and the test set. The training set is data used to compute forward and backward propagation, therefor those data are the only one used to update weights, they represent around 70% of the dataset. The validation set is data used to monitor training and prevent overfitting, they represent around 15% of the dataset. The test set allows to tune hyperparameters and select the best model, they represent around 15% of the dataset.

During training : In practice, training data are sent by batch into the network and weights are updated. When all the train set has been sent in the network an epoch is finished. At epoch end, metrics such as mean squared error or accuracy are computed on the training set and on the validation set. Please find in figure 1.4 page 5, in blue the theoretical curve of the mean squared error on the training set and in pink the theoretical curve of the mean squared error on the validation set with a model that overfits after 350 epochs.

Model selection : Once you have trained different models, you can test their performances comparing metrics on the test set. The best models can be combined by averaging their output in case of regression or choosing the majority class in case of classification.

1.2.2 Deep Learning in Practice

Hardware : The most important component of a computer to train a neural network is the GPU. Two options are possible :

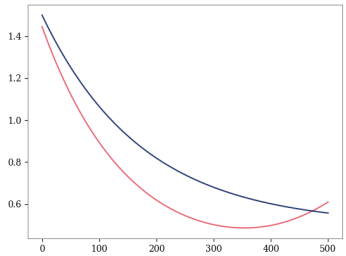


Figure 1.4: Example of metric curves

- Train a network on your own computer. This option requires a powerful GPU (at least an NVIDIA GTX 1060 6Go of RAM)
- Train your network on a GPU cluster. That option is not free and an internet connection is required.

Languages : The main programming language used for Deep Learning is Python because of the multiplicity of frameworks that exist. It is also possible to train basic neural network in R and Java but those options are marginal. Finally, C++ allows you to create Neural Networks for onboard devices.

Framework : The most common framework and the one used for that project is keras[4] with Tensorflow[1] as backend. Tensorflow[1] can also be used without keras.

1.3 Limits

1.3.1 Technical Issues

Complexity : Compared to other Machine Learning models, Artificial Neural Networks are hard and long to train. Furthermore, to train a neural network one needs to tune much more hyperparameters than with other Machine Learning models such as Random Forest.

Lack of theory : Even though Neural Networks were coined in 1943 by Warren McCulloch, their implementation are quite recent. Therefor Deep Learning is mostly an empirical field of Artificial Intelligence. This empiricism is highlighted when one has to tune the hyperparameters of its network.

1.3.2 Reluctance of the Industrial World

Black Box : Deep Learning gives outstanding results in many tasks like Image Analysis, Natural Language Processing, Time Series Predictions ... Those results are hardly explicable because unlike Linear Regression or Decision Trees, the prediction process is too complex to be computed by a human being. Therefor Neural Networks are like black boxes.

Uncertainties : Unlike a Linear Regression, a standard Neural Network do not give uncertainties values with its prediction. And this point is going to be discussed in the second chapter of this literature review.

Bayesian Deep Learning

2.1 Bayesian Neural Networks

2.1.1 Introduction

Bayes formula : The Bayesian adjective derives its name from Thomas Bayes that discovers the Bayes Theorem, which is the equation 2.1 page 6. Whereas that discovery is due to Bayes, Pierre-Simon Laplace is the one that understood in 1774 the implications of the formula in statistics and probabilities.

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)} \tag{2.1}$$

where A and B are two random variables.

Application to Deep Learning : The main difference between Bayesian Deep Learning and standard Deep Learning is that weights are not scalar but gaussian distributions. During training, prosterior distribution of weights is computed with the prior distribution of weights with the Bayes Theorem[3]. The advantage of Bayesian Deep Learning over standard Deep Learning is the uncertainty computation that is required in some projects like the Google Car.

2.1.2 Mathematical Notions

Notations : The dataset $D = \{(x_n, y_n)\}_{n=1}^{N}$ is composed of the input data x_n and the target data y_n. $\{t_n\}_{n=1}^{N}$ are the output data. The Neural Network has L layers with V_l units per layers. $W = \{W_l\}_{l=1}^{L}$ is a collection of weights matrix.

Bayesian training : Considering the prior distribution of weights, one can obtain the posterior distribution of weights with the Bayes rule[3], as shown at the equation 2.2 page 6. In most cases, $P(W|x, y, D')$ and $P(y|x, D')$ are intractable

$$P(W|x, y, D') = \frac{P(y|x, W)P(W|D')}{P(y|x, D')} \tag{2.2}$$

where $P(y|x, D')$ is a normalisation constant, D' the already observed data of the dataset, $P(y|x, W, D')$ the likelihood function, $P(y|x, W, D')$ the prior distribution of weights and $P(W|x, y, D')$ the posterior distribution of weights.

Bayesian prediction : Prediction can be done integrating over the space of weights[3] like at the equation 2.3 page 7. This is called marginalisation and consist in computing a weighted average on the weight space.

$$p(t|x, D) = \int p(t|x, W)p(W|D)dW \tag{2.3}$$

where $p(t|x, D)$ is the predictive distribution, $p(W|D)$ the distribution of the weights and $p(t|x, W)$ the output distribution considering a specific distribution of weights.

2.2 Uncertainties

2.2.1 Aleatoric Uncertainties

Definition : As explained by Kendal in an article[9] in 2017, aleatoric uncertainties are inherent to the data. Therefor a noisy dataset will produce huge aleatoric uncertainties. In the classification example of the figure 2.1 page 7, these uncertainties mainly lie in the junction between classes. These uncertainties can be classified as heteroscedastic and homoscedastic uncertainties. They are respectively the variable and the constant parts of aleatoric uncertainties, this difference intervenes with data that are not equally noisy.

Reduction method : In regression cases, a modification of the loss allows you to reduce the heteroscedastic giving more importance to the unnoisy input data. Please find this modification of the loss function at the equation 2.4 page 7.

$$L_{alea}(x_i, y_i, t_i) = \frac{L(y_i, t_i)}{2\sigma(x_i)^2} + \frac{1}{2}log(\sigma(x_i))^2 \tag{2.4}$$

where $L(y_i, t_i)$ is the standard loss and σ the noise of the data.

2.2.2 Epistemic Uncertainties

Epistemistic uncertainties correspond to the uncertainties inherent to the model. Those uncertainties decreases while training. For instance in the figure 2.1 page 7 most of the epistemic uncertainties are localised in the class of the sidewalks. Those uncertainties detection will be discussed in the next section.

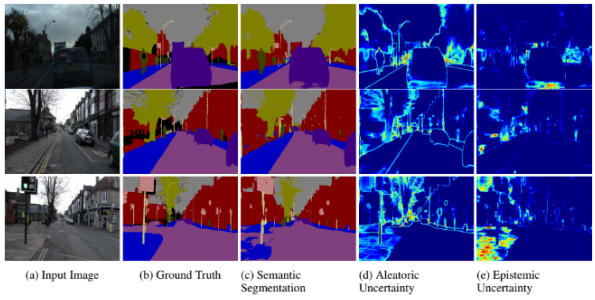


Figure 2.1: Illustration of the differences between the uncertainties extrated from an article of Dr Kendall[9]

2.3 Main Probabilistic Techniques

2.3.1 Error Bars

Intuition : A classic misconception is to consider the softmax output as the model confidence as explained by Yarin Gal [6]. Chritopher Bishop proposed a solution to tackle this issue in an article where he exposes the main Bayesian Methods adapted to Deep Learning[3]. The idea is to make modifications on the backpropagation algorithm to make it returns the result with uncertainties.

Concept : The idea behind adding error bars is to make a second order Taylor expansion of the loss function around the best weights as described at the equation 2.5 page 8. By a modification of the backward propagation exposed by Bishop [2], it is possible to compute the exact value of the Hessian matrix. With the Gaussian approximation justified by Walker[15] and the equation 2.5 page 8, one can deduce an approximation of the equation 2.3 page 7. Please find this approximation at the equation 2.6 page 8. Once you have the value of σ_t^2 you can compute error bars.

$$L(w) = L(w_{best}) + \frac{1}{2}(w - w_{best})^T H_L(w_{best})(w - w_{best}) \tag{2.5}$$

where L is the loss function, w_{best} the best weights obtained after training and $H_L(w_{best})$ the Hessian matrix of the loss function computed for w_{best}.

$$p(t|x, D) = \frac{1}{\sqrt{2\pi\sigma_t^2}} e^{-\frac{(t - y_{best})^2}{2\sigma_t^2}} \quad with\ \sigma_t^2 = \frac{1}{\beta} + \nabla_w y_{w=wbest}^T H_L(w_{best}) \nabla_w y_{w=wbest} \tag{2.6}$$

where β depends in the aleatoric uncertainties, $\nabla_w y_{w=wbest}$ is the gradient of the output computed for w_{best}

Pros and cons : This technique offers a first approach on how one can compute error bars and also offers an approximation of the intractable equation 2.3 page 7. The main issue their is the computation complexity due to Hessian calculus.

2.3.2 Hyperparameter Tuning

Intuition : Bayesian Deep Learning introduce new hyperparameters in the Neural Network, β defined at the equation 2.6 page 8 and an hyperparmeter α that will be defined in the next paragraph. One of the great strength of Bayesian Deep Learning is to compute those hyperparameters while training. This computation technique is exposed in an article written by Hernandez[8].

Concept : Let's define β properly, A Bayesian Neural Network is a function approximation technique that follows the equation 2.7 page 8. Now, we introduce a precision parameter α for weights, in order that the prior distribution of weights follows : $w_{i,j,k} \sim \mathcal{N}(0, \alpha^{-1})$. Those hyperparmeters tuning is performed converting the equation 2.2 page 6 into the equation 2.8 page 8.

$$y_n = f(x_n, W) + \epsilon_n \ with\ \epsilon_n \sim \mathcal{N}(O, \beta^{-1}) \tag{2.7}$$

where f is the Neural Network approximation function

$$P(W|x, y, \alpha, \beta, D') = \frac{P(y|x, W, \beta)P(W|\alpha, D')P(\alpha)P(\beta)}{P(y|x, D')} \tag{2.8}$$

Pros and cons : This technique increases the computation complexity but gives a better understanding and description of the aleatoric and epistemistic uncertainties. It also adds more approximation to the weights and output distributions.

2.3.3 Probabilistic Backpropagation

Intuition : The probabilistic backpropagation is based on an article of Hernandez[8]. The main idea proposed in that article is that weights are gaussians and that can be update using a variant of the backpropagation algortihm.

Formal definition : Let's define Z the normalisation component of the equation 2.2 page 6. Using an approximation developped by Dr Minka in 2001[12], weights can be updated as shown at the equations 2.9 and 9.

$$\begin{cases} m^{new} = m + v\dfrac{\partial log(Z)}{\partial m} \\ v^{new} = v - v^2\left(\left(\dfrac{\partial log(Z)}{\partial m}\right)^2 - \left(\dfrac{\partial log(Z)}{\partial v}\right)\right) \end{cases} \tag{2.9}$$

where $w \sim \mathcal{N}(m, v)$

Pros and cons : This method proposed by Dr Hernandez is in practice quicker than the one exposed by Bishop and it also does not need the computation of any Hessian matrix which is a huge complexity gain. On the other hand, Yarin Gal highlights in its article[6] that weights storage is twice bigger which is an issue for onboard systems.

2.3.4 Monte-Carlo Dropout

Intuition : In 2016 Yarin Gal proposed a ground breaking technique to compute a Bayesian approximation using standard Deep Learning Networks [6]. This method is based on Dropout layers, which consist in randomly set neurons value to zero with a probability that depends on the layer. The figure 2.2 page 9 illustrates that principle.

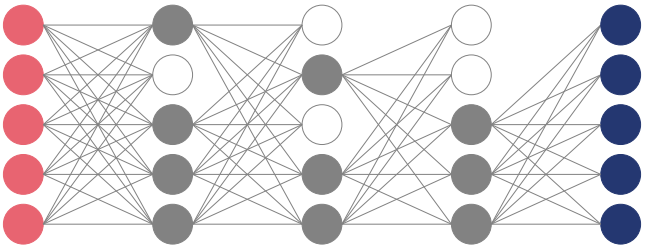


Figure 2.2: Dropout example with a dropping rate of 30%

Concept : Yarin Gal defines the distribution of its weights with the equation 2.10 page 10. The loss defined in the article corresponds to a cross entropy loss with weight decay. Dr Gal shows the equivalence between a Gaussian process and a Network based on Dropout Layers and cross entropy loss described in this paragraph. Therefor to compute an estimation of the equation 2.3 page 7, one can use the Monte-Carlo approximation on the network we have defined. You will find an approximation of the mean and the variance of the output respectively at the equation 2.11 page 10 and the equation 2.12 page 10.

$$W_l = M_l.diag((z_{l,i})_{i=1}^{V_l}) \; with \; z_{l,i} \sim \mathcal{B}(0, p_l) \tag{2.10}$$

where M_l is a matrix containing the weights value of l^{th} layer before dropout.

$$\mathbf{E}(p(t|x, D)) = \frac{1}{T}\sum_{i=1}^{T} t^i(x, W_i) \tag{2.11}$$

$$\mathbf{Var}(p(t|x, D)) = \frac{1}{\beta}.I_d + \frac{1}{T}\sum_{i=1}^{T} t^i(x, W_i)^T t^i(x, W_i) + \mathbf{E}(p(t|x, D))^T \mathbf{E}(p(t|x, D)) \tag{2.12}$$

where $t^i(x, W_i)$ is the result of the i^{th} forward propagation in the network with x as input

Pros and cons : The method described can be computed quickly, and is easy to implement because you don't need a probabilistic framework like PyMC[13] or Edward[14]. The main drawback is how recent this article is. So far, few research teams and companies have implemented it yet.

2.4 Conclusion

Overview : This literature review proposed an overview of Bayesian Deep Learning structured around an explanation of Deep Learning, an exposition of the main probabilistic models applied to Neural Networks and their applications for uncertainties computation.

To go further : This literature review will be followed by an implementation of the main Bayesian techniques. Those implementations will be tested on the MNIST dataset[11], that will lead to the redaction of a scientific article and a poster.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Chris Bishop. Exact calculation of the hessian matrix for the multilayer perceptron, 1992.

[3] Christopher M Bishop. Bayesian methods for neural networks. *Neural Computing Research Group Report NCRG/95/009, Department of Computer Science and Applied Mathematics, Aston University, Birmingham B4 7ET, UK*, 1995.

[4] François Chollet et al. Keras, 2015.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[6] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[7] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[8] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

[9] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017.

[10] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[11] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998. *URL http://yann. lecun. com/exdb/mnist*, 10:34, 1998.

[12] Thomas Peter Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

[13] Anand Patil, David Huard, and Christopher J Fonnesbeck. Pymc: Bayesian stochastic modelling in python. *Journal of statistical software*, 35(4):1, 2010.

[14] Dustin Tran, Alp Kucukelbir, Adji B Dieng, Maja Rudolph, Dawen Liang, and David M Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

[15] AM Walker. On the asymptotic behaviour of posterior distributions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 31(1):80–88, 1969.

[16] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Uncertainty Quantification Applied to Monte-Carlo Dropout Neural Networks, a Bayesian Approximation

Robin Camarasa[1, 2] and Samy Melaine[2]

[1]Mines de Saint-Etienne, Saint-Etienne, France
[2]Data Genius, Villeurbanne, France

August 14, 2023

Abstract

Deep Learning has become a game changer technology in all the field of industry. Indeed, from predictive maintenance to image analysis, A.I. systems outperform humans in a lot of tasks. However Neural Networks are seen by most of the industrial companies as an over-confident black-box. Therefore, the main purpose of this article is to tackle this over-confidence using a ground-breaking technique to compute epistemic uncertainties.

1 Keywords

Data Sciences, Artificial Intelligence, Machine Learning, Bayesian, Deep Learning, Epistemic Uncertainties

2 Introduction

Neural Networks and Deep Learning are statistical learning techniques born in the 40s with the work of Warren McCulloch and Walter Pitts [McCulloch and Pitts, 1943]. However, at that time, this revolutionary idea did not have industrial applications because of the power of computation in the 40s. With the work of Yann Lecun [LeCun et al., 1998] and the evolution of the hardware devices, especially the GPU, over the last two decades, McCulloch and Pitt's theory has become a reality.

Those outstanding techniques outperform other statistical learning models like Random Forest, Logistic Regression ... but a major disadvantage slows down their expansion. Deep Learning acts as a black box which induces reticence from the industrial word toward those methods. Two main strategies are proposed to counter the lack of confidence one have in Neural Networks. Some researchers work on the interpretability of Networks [Lipton, 2016] while others propose to add uncertainties to Networks [Gal and Ghahramani, 2016]. The second approach will be developed in this article.

The main goal of this paper is to identify if an input of the test set is part of a sparse space of the input training set using epistemic uncertainties. Our approach was to qualify different types of uncertainties, train a Vanilla Neural Network over the MNIST dataset[LeCun et al., 1998] and then apply Dr Gal's framework[Gal and Ghahramani, 2016].

3 Materials and Methods

3.1 Epistemic and Aleatoric Uncertainties

The two main sources of uncertainties are aleatoric uncertainties and epistemic uncertainties :

- As explained by Kendal in an article[Kendall and Gal, 2017] in 2017, aleatoric uncertainties are inherent to the

data. Therefore a noisy dataset will produce huge aleatoric uncertainties. Those uncertainties can be classified as heteroscedastic or homoscedastic. They are respectively the variable and the constant parts of aleatoric uncertainties, this difference intervenes with data that are not equally noisy.

- Epistemic uncertainties correspond to the uncertainties inherent to the model. Those uncertainties decreases while training.

Our research will be focused on epistemic uncertainties.

3.2 Dataset and tools

Figure 1: Sample of the MNIST dataset

The experiments were launched on the famous, 28×28 greyscaled images of numbers dataset, the MNIST[LeCun et al., 1998]. You will find an element of this dataset at figure 1 page 2. It contains 60 000 annotated images.

Training was performed on an Ubuntu 18.04 distribution of Python 3.6 with the packages Keras 2.2.0 [Chollet et al., 2015], Tensorflow-gpu 1.9.0[Abadi et al., 2016]. About the hardware, the computer was equipped with an NVIDIA GeForce GTX 1060 6Go of RAM and an Intel Core i7-7700.

3.3 Vanilla Network

In order to have the more usual Neural Network, mean squared error loss and stochastic gradient descent optimiser (default Keras[Chollet et al., 2015] learning rate) were used. The structure is a combination of fully-connected layers (figure 2 page 2). The number of neurons per layer and number of layers are the hyperparmeters of the network.

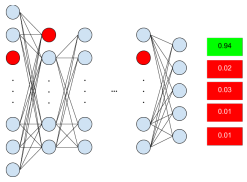


Figure 2: Vanilla network structure

The training contained 250 epochs with a batch size of 50 images. Overfitting has been prevented splitting data in 3 samples, training (70%), validation (15%) and test set (15%).

Once the hyperparameters correctly fine-tuned the same structure was applied on a 5-classes-output network.

3.4 Bayesian Deep Learning

The dataset $D = \{(x_n, y_n)\}_{n=1}^{N}$ is composed of the input data x_n and the target data y_n. $\{t_n\}_{n=1}^{N}$ are the output data. The Neural Network has L layers with V_l units per layers. $W = \{W_l\}_{l=1}^{L}$ is a collection of weights matrix.

Considering the prior distribution of weights, one can obtain the posterior distribution of weights with the Bayes rule[Bishop, 1995], as shown at the equation 1 page 2. In most cases, $P(W|x, y, D')$ and $P(y|x, D')$ are intractable.

$$P(W|x, y, D') = \frac{P(y|x, W)P(W|D')}{P(y|x, D')} \qquad (1)$$

where $P(y|x, D')$ is a normalisation constant, D' the already observed data of the dataset, $P(y|x, W)$ the likelihood function, $P(W|D')$ the prior distribution of weights and $P(W|x, y, D')$ the posterior distribution of weights.

Prediction can be done integrating over the space of weights[Bishop, 1995] like at the equation 2 page 2. This is called marginalisation and consist in computing a weighted average over the space of weights.

$$p(t|x, D) = \int p(t|x, W)p(W|D)dW \qquad (2)$$

where $p(t|x, D)$ is the predictive distribution, $p(W|D)$ the distribution of the weights and $p(t|x, W)$ the output distribution considering a specific distribution of weights.

3.5 MC Dropout - a Bayesian Approximation

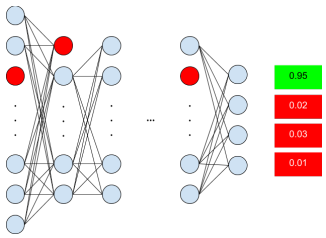


Figure 3: Monte-Carlo Dropout Neural Network

In 2016 Yarin Gal proposed a ground breaking technique to compute a Bayesian approximation using standard Deep Learning Networks [Gal and Ghahramani, 2016]. This method is based on Dropout Layers, which consist in randomly set neurons value to zero with a probability that depends on the layer. The figure 3 page 3 illustrates that principle.

Yarin Gal defines the distribution of its weights with the equation 3 page 3. The loss defined in Gal's article corresponds to a cross entropy loss with weight decay. Dr Gal shows the equivalence between a Gaussian process and a network based on Dropout Layers and cross entropy loss described in this paragraph. Therefore, to compute an estimation of the equation 2 page 2, one can use the Monte-Carlo approximation on the network we have defined. You will find an approximation of the mean and the variance of the output respectively at the equation 4 page 3 and the equation 5 page 3.

$$W_l = M_l.diag((z_{l,i})_{i=1}^{V_l}) \; with \; z_{l,i} \sim \mathcal{B}(0, p_l) \quad (3)$$

where M_l is a matrix that contains the weights values of l^{th} layer before dropout.

$$\mathbf{E}(p(t|x, D)) = \frac{1}{T}\sum_{i=1}^{T} t^i(x, W_i) \quad (4)$$

$$\mathbf{Var}(p(t|x, D)) = \frac{1}{\beta}.I_d + \frac{1}{T}\sum_{i=1}^{T} t^i(x, W_i)^T t^i(x, W_i) + \mathbf{E}(p(t|x, D))^T \mathbf{E}(p(t|x, D)) \quad (5)$$

where $t^i(x, W_i)$ is the result of the i^{th} forward propagation in the network with x as input

By some manipulation of the equation 6 page 3 exposed in the article of Dr Kwon [Kwon et al., 2018], one can obtain the equation 6 page 3. From this equation, one can extract the epistemic contribution to the covariance matrix.

$$\mathbf{Var}(p(t|x, D)) = \underbrace{\frac{1}{T}\sum_{t=1}^{T} diag(t^i(x, W_i)) - t^i(x, W_i)^{\otimes 2}}_{epistemic} + \underbrace{\frac{1}{T}\sum_{t=1}^{T}(t^i(x, W_i) - \bar{t})^{\otimes 2}}_{aleatoric} \quad (6)$$

where $u^{\otimes 2} = uu^T$, $\bar{t}$ is the empiric mean and $diag(u)$ is a diagonal matrix with the element of the vector u.

Like the Vanilla Neural Network, the MC Dropout Neural Network, illustrated at figure 3 page 3, have the number of layers and the number of neurons per layer as hyperparameter. But new hyperparameters had to be added to the model, the dropping rate of each layer.

The epistemic metric that will be discussed in the results' section is the average of the diagonal elements of the epistemic matrix in equation 6 page 3. The value of this metric is high when the input data does not fit what the model learnt during training and low when it fits.

4 Results

4.1 Vanilla Neural Network

To obtain a fine-tuned network we used a one-at-the-time optimisation algorithm as illustrated at

Table 1: Accuracy mean values computed over the test set

	5 classes	**10 classes**
Accuracy	0.97	0.94

figure 4 page 4, the optimised criterion was the accuracy of the model over the test set. The best network obtained with this algorithm is a 4 inner layers with 25 units per layer Neural Network.

Once the 10-classes-output network and the 5-classes-output network were fine-tuned, we computed the accuracy on the test set. You will find the result of those computations at table 1 page 4. One can notice that the accuracy is better for the 5-classes-output network, this is due to the fact that it is easier to learn to a model to recognise number from 0 to 5 rather than to recognise numbers from 0 to 9.

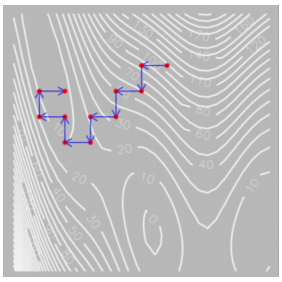


Figure 4: One-at-the-time optimisation algorithm visualised on a minimisation problem of a function $f : \mathbf{R}^2 \to \mathbf{R}$

4.2 MC Dropout - a Bayesian Approximation

Table 2: Epistemic mean values computed over the test set

	0-4 classes	**5-9 classes**
Epistemic	3.61×10^{-3}	2.22×10^{-2}

Another one-at-the-time algorithm was used to obtain an approximation of the optimal dropping rates. The maximised criterion with this algorithm is the difference between the mean of the epistemic metric computed over the unlearnt classes elements of the test set and the mean of the epistemic metric computed over the learnt classes elements of the test set. The dropping rates that maximise this criterion were 0.1.

Once the network fine-tuned, we computed the epistemic metric mean over the elements of the unlearnt classes of the test set and over the elements of learnt classes of the test set. Those results are referenced in the table 2 page 4.

5 Discussion

The results of the table 2 page 4 show that it is possible to differentiate an element of a class that was learnt during training from an element of a class that was not. From that perspective, that result demonstrate the efficiency of the technique developed in that paper.

However, the one-at-the-time optimisation algorithm used to fine-tuned the dropping rates hyperparameters introduce a bias in the analyse because the test set is used to fine-tune the hyperparameters and test the efficiency of the network. A better method would have been to split the test set but that required to use more data for testing and less for training.

The technique developed here could have a huge impact in systems that need to counter one pixel attack or where human life are at risk like autonomous cars. Furthermore, this technique can be applied to networks that have already been trained.

6 Conclusion

This article gave a framework to compute epistemic uncertainties and applied this framework to the MNIST dataset[LeCun et al., 1998]. That technique is part of an overall reflection about how to make neural networks safer from a human point of view.

Many interesting projects could follow this work. On the one hand, this technique could be compared this to classic Bayesian approaches[Hernández-Lobato and Adams, 2015], on the other hand, aleatoric uncertainties has not be treated in this article because the data were equally noisy, one could also work on this aspect adding some blur in the input data and quantify the impact on aleatoric uncertainties.

References

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

[Bishop, 1995] Bishop, C. M. (1995). Bayesian methods for neural networks. *Neural Computing Research Group Report NCRG/95/009, Department of Computer Science and Applied Mathematics, Aston University, Birmingham B4 7ET, UK*.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras.

[Gal and Ghahramani, 2016] Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

[Hernández-Lobato and Adams, 2015] Hernández-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869.

[Kendall and Gal, 2017] Kendall, A. and Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584.

[Kwon et al., 2018] Kwon, Y., Won, J.-H., Kim, B. J., and Paik, M. C. (2018). Uncertainty quantification using bayesian neural networks in classification: Application to ischemic stroke lesion segmentation.

[LeCun et al., 1998] LeCun, Y., Cortes, C., and Burges, C. J. (1998). The mnist database of handwritten digits, 1998. *URL http://yann. lecun. com/exdb/mnist*, 10:34.

[Lipton, 2016] Lipton, Z. C. (2016). The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.