COM3023, Continuous Assessment

180707

March 24, 2024

Abstract

This paper aims to investigate the implementation of Bayesian Neural Networks, in particular their use in being used to assess the uncertainty within Neural Network model parameters. It makes use of approximation methods such as Markov Chain Monte Carlo and Variational Inference in order to implement this Bayesian approach. Through the use of the Breast Cancer Wisconsin dataset, we shall see how such methods are implemented and what benefits they may hold. We shall also compare the two methods in order to provide an overview of the key differences and capabilities they hold.

I certify that all material in this document which is not my own work has been identified.

1 Introduction

1.1 Problem Outline

This paper seeks to investigate Bayesian Neural Networks (BNNs), with a focus on how they use Bayesian techniques to manage the uncertainties in the parameters of Artificial Neural Networks. It explores the use of two popular posterior approximation techniques, Markov Chain Monte Carlo (MCMC) and Variational Inference (VI). We explore these approaches through the use of the popular Breast Cancer Wisconsin dataset [20] in order to gain insight into the real-world applicability of such methods.

1.2 Neural Networks

Over the last decade, renewed interest has been placed on Neural Networks (NN), which has led to a number of breakthroughs in the field of Machine Learning (ML). NN models trace their origin back to the Perceptron architecture proposed by McCulloch and Pitts in 1943 [12]. It was originally designed to be a binary classifier composed of a single layer, which can be used to model linear relationships. It used parameters, also referred to commonly as weights, to classify an input based on its features. The first actual implementation of the model was made in 1957 by Rosenblatt [17] for image classification.

The modern day NN is in most cases synonymous with a Multi-Layered Perceptron (MLP). This architecture, as its name suggest, is composed of multiple stacked Perceptrons which allow for more complicated relationships to be learned between the input and output features. Non-linear activation functions, such as the sigmoid and ReLU functions [19], allowed for the scope of relationships being modelled to be expanded to non-linear relationships.

Furthermore, the implementation of NNs in real world settings was limited until the late 90s by three factors, a lack of computing power, a lack of available data and a lack of efficient algorithms. The development of graphics processing units (GPUs) revolutionised the computing power space with its multiple core architectures, which were capable of rapid parallel matrix multiplication, the underlying calculations necessary in NNs. This allows for efficient calculations in MLPs, which can have tens to hundreds of different layers and nodes.

The backpropagation algorithm, whose use in the space of NNs was popularised by [18] helped make training more efficient and have deeper architectures. It allows for errors to be fed backwards through the algorithm so that weights are adjusted based on what the model has learned. Meanwhile the technological revolutions of the 90s and early 2000s brought about the large increases in data availability needed in order to train models. NNs have therefore grown significantly in popularity in recent decades and their application scope has been constantly expanding.

1.3 Bayesian Approach

While traditional approaches to ML such as NNs have shown their effectiveness, they have also shown certain downsides. One of these is that NNs are usually based on frequentist statistics which means they are "prone to overfitting, dimming their generalization capabilities and performance on unseen data" [11]. Traditional NNs are also modelled as black boxes with their inner workings being difficult to understand. They also do not allow us to easily visualise the uncertainty present in the different parameter weightings. This has led to their real world im-

plementations to be limited down to use cases where uncertainty knowledge is not needed.

It has, therefore, limited their implementation in areas such as healthcare where uncertainty in predictions can have devastating consequences if not taken into account. There has even been a push by bodies such as the Europe Union to mandate that predictions made by AI algorithms, especially those deployed in high risk environments, to have their inner workings be easily explainable so that predictions made can be interpreted by non-experts [22]. This leads us to the incorporation of Bayesian methods into NNs.

Bayes formula is given by:

$$P(\theta|y,x) = \frac{P(y|\theta,x) \cdot P(\theta|x)}{P(y|x)}$$

Where $P(y|\theta,x)$ is the likelihood, $P(\theta|x)$ is the prior and $P(y|x)$ is the evidence (normalisation factor).

Bayesian methods provide a good alternative to frequentist methods. Instead of point estimate they provide estimated distributions for the parameter being estimated. This allows the uncertainty in a model's weights and predictions to be quantified and visualised. They also reduce the likelihood of overfitting, as well as allowing the integration of prior knowledge into the estimation. The challenge in the case of Bayesian methods is the calculation of the posterior distribution based on the prior distribution, the likelihood and the normalising constant. This often proves challenging as the normalising constant is often intractable or unknown. As a result, Bayesian approaches are usually more computationally expensive to run and have to rely on specific methodologies to approximate the posterior.

This project seeks to apply such Bayesian methods to a NN model in order to get more information on the choice of model weightings, as well as the uncertainty in the predictions made by the model. In a first part we investigate two methods which are used to approximate the posterior in Bayesian inference, MCMC and VI. We will then look into the design of the experiment conducted. In a third part we show the results obtained from applying our methods on the Breast Cancer Wisconsin dataset. Finally, we look at the differences produced by using MCMC and VI. Overall, this project will provide a good overview of the implementation of Bayesian methods in NNs.

2 Background of Chosen Techniques

2.1 Markov Chain Monte Carlo

Different methods attempt to fill the gap caused by the intractable evidence value in the calculation of the posterior. One such methods is MCMC, a method which allows sampling from the true posterior distribution through simulation. First introduced under the form of what is now known at the Metropolis-Hasting Algorithm in 1953 [13]; it does this by constructing a Markov chain which has the true posterior as its stationary distribution. This allows for samples to be drawn using Monte Carlo simulation techniques [14] in place of the intractable distribution. This method assumes convergence of the chain to the true posterior as the number of samples drawn increases. Its reliance on sampling means that we can use the samples drawn in order to quantify the uncertainty in our model parameters.

The benefit of MCMC is that it is sampling from the estimate of the true distribution, so as more samples are drawn the accuracy should increase. However, it can very computationally

expensive to use MCMC, especially in cases where the data is highly dimensional. Without proper convergence of the Markov chain to the true posterior, the results obtained can also be unreliable. Overall, however, MCMC provides a good method to estimate the posterior distribution and therefore can play a key role in BNNs.

2.2 Variational Inference

An alternative to MCMC is VI which seeks to draw approximate samples from the posterior distribution through it being "approximated by the so-called variational distribution" [11]. The variational distribution is most often chosen as a probability density function with well-known characteristics such as a Normal or Beta distribution. To estimate the posterior distribution, we turn the estimation problem into an optimisation problem. We therefore try and find the parameters for the variational distribution which will allow it to be as similar as possible to the true posterior distribution. To do this we can minimise the KL divergence between the variational distribution and the posterior, but the equivalent maximisation of the Evidence Lower BOund (ELBO) is often preferred.

Additionally, VI has the benefit that it is sampling from a simpler distribution than the true posterior which allows for more computational efficiency. This is especially useful in cases of high dimensional data where methods such as MCMC struggle. However, this may also lead to it having less reliable estimates than those produced by methods such as MCMC which are sampling from the true posterior. In conclusion, VI presents a method for the estimating the posterior which is efficient and powerful, especially for high-dimensional data.

3 Experimental Design

3.1 Data and Preprocessing

The Breast Cancer Wisconsin dataset is composed of data points which describe the characteristics of the cell nuclei of different tumour images [20]. It is composed of 569 instances with 30 features including radius, texture and smoothness. Its output variable is a classification of a tumour as benign or malignant. Modelling the uncertainty in our model when classifying cancer tumours may be particularly useful in order to know how the model may arrive at a certain classification, especially given the high impact of the classification on a patient's life. Knowledge of the uncertainty may boost confidence in the implementation of such systems in a real world setting. Uncertainty in both the model parameters and the estimates can therefore be of interest. Summary statistics for a selection of the most important features can be found in Table 1.

3.2 Packages Used

The BNN's implementation will rely on Pyro [3], a Python package developed by researchers at Uber. It was designed to enable the efficient creation of advanced probabilistic models, including Bayesian methods such as MCMC and VI. It is based on PyTorch [15][16], one of the most popular packages for implementing NNs. While PyTorch allows the use of GPUs to train models, Pyro currently has tensor transfer issues when using BNNs. This project will therefore use a CPU to train the models. This will likely have an impact on the training times of the models, but the general comparisons should remain the same when comparing MCMC and VI.

In order to train our models we will use the No-U-Turn Sampler (NUTS) [6] for our MCMC implementation, which is type of Hamiltonian Monte Carlo algorithm, an MCMC method based on Hamiltonian Dynamics. These methods help to adapt traditional MCMC to higher dimension space and improve the convergence. Our VI method will rely on Stochastic Variational Inference (SVI) [5], a method which adds stochastic optimisation to the Variational Inference optimisation process. It relies on adjusting the VI distribution to a random chosen subsample at each step which allows it to scale better to larger data and speed up the optimisation process. Both these methods represent the state-of-the-art methods for implementing BNNs using Pyro.

Table 1: Data Summary Statistics

Feature	Count	Mean	Std	Min	25%	50%	75%	Max
Mean Radius	569.000	14.127	3.524	6.981	11.700	13.370	15.780	28.110
Mean Texture	569.000	19.290	4.301	9.710	16.170	18.840	21.800	39.280
Mean Perimeter	569.000	91.969	24.299	43.790	75.170	86.240	104.100	188.500
Mean Area	569.000	654.889	351.914	143.500	420.300	551.100	782.700	2501.000
Mean Smoothness	569.000	0.096	0.014	0.053	0.086	0.096	0.105	0.163
Mean Compactness	569.000	0.104	0.053	0.019	0.065	0.093	0.130	0.345
Mean Concavity	569.000	0.089	0.080	0.000	0.030	0.062	0.131	0.427
Mean Concave Points	569.000	0.049	0.039	0.000	0.020	0.034	0.074	0.201
Mean Symmetry	569.000	0.181	0.027	0.106	0.162	0.179	0.196	0.304
Mean Fractal Dimension	569.000	0.063	0.007	0.050	0.058	0.062	0.066	0.097
Target	569.000	0.627	0.484	0.000	0.000	1.000	1.000	1.000

3.3 Model Architecture

The BNN model architecture used in this project is illustrated in Figure 1. It consists of four layers including two hidden layers. Each hidden layer contains fifteen nodes. We set out priors to be Normal distributions centred on zero with a standard deviation of one. This provides an unbiased, although uninformative, estimator of the model parameters. It serves rather as a regularisation tool to stop parameters being set to large values and causing overfitting. We also use a ReLU activation function [1], which allows the model to learn non-linear relationship, along with batch normalisation [7] for each input into the hidden layers. The model presented remains a fairly simple architecture. In a real-world implementation the model would likely be a lot more complex, including more layers, more nodes and case specific priors. However, for the scope of this project a simpler architecture is selected in order to reduce the computational resources required.

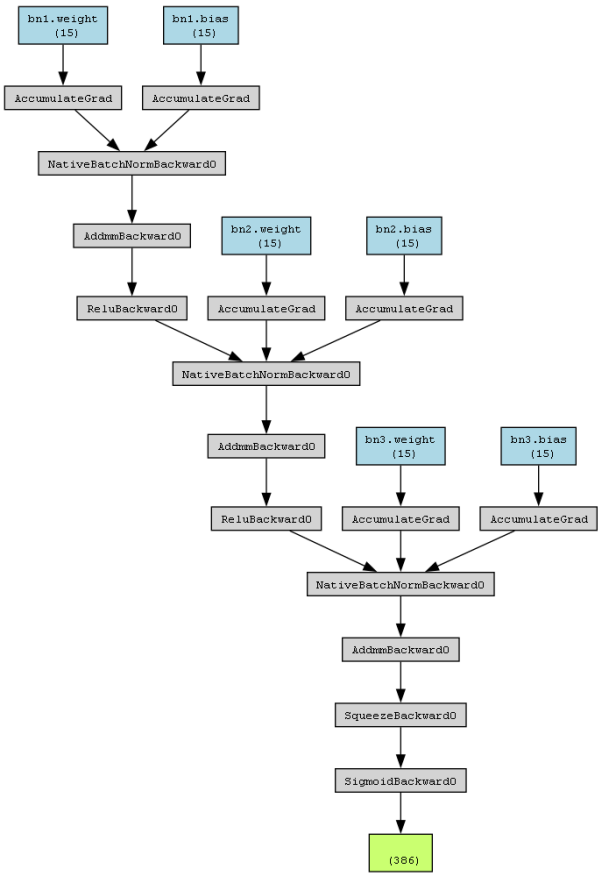


Figure 1: Model Architecture of Bayesian Neural Network

3.4 Training and Evaluation

Our data is split into training, validation and testing sets. The training set, as its name suggests, is used to train the model on labelled examples. The validation set is used to fine-tune the hyperparameters; this is especially useful for our VI implementation as it allows us to monitor the evolution of the loss on unseen data for different number of epochs. The testing set is used at the end to assess the overall performance of our models on unseen data points.

Our MCMC method is run using 100 warm up steps, 2000 samples and two chains. Warm-up steps allow the NUTS algorithm to draw a chosen number of initial samples, this allows the chain to transition to its steady state before sampling begins. These initial warm-up samples are discarded so that only samples drawn from the stationary distribution are used. Using multiple chains allows for more accurate convergence and space exploration. In order to verify good convergence, it is important to verify that the Gelman-Rubin statistic, or R-hat, has a value as close as possible to 1 [4][21]. This ensures that the chains have correctly explored the parameter space and converged to the same distribution for each chain ensuring that our weights are being effectively estimated. We also want to verify that our trace plot shows a so-called "hairy caterpillar" look for all the parameters, this ensure effective space exploration.

Our VI implementation uses an AdamW optimiser [10] and Automatic Differentiation Variational Inference (ADVI) [2][9]. The AdamW optimiser improves on the Adam optimiser [8], fixing problems with weight decay in the Adam optimiser where large weights were not regularised as well as smaller weights. It does this by adjusting the regularization based on the size

of the weight. The ADVI algorithm defines the distribution we think should be used to approximate our distribution. For simplicity we use a standard Normal distribution. We use 20000 epochs and a learning rate of 0.005 to train our model and use 10000 samples drawn from the distribution for evaluation. This ensures a good balance between exploration and exploitation.

We will compare the performance of both methods using three criteria: accuracy, computational efficiency and applicability to real-world data. It is important to evaluate all three of these criteria in order to give an accurate comparison of the performance of both models.

4 Results

4.1 Performance Evaluation

The MCMC method tends to have produced good convergence. The weights for each layer can be seen in Figure 3, they seem to have all converged towards standard normal distributions as specified in our prior. All of the R-hat scores have converged to 1 while the chain acceptance probability has stayed around 0.6 as specified in [6] when using the NUTS kernel. The model provides good accuracy at 93% and good F1-score at 94%. These are calculated as the average of the metric scores over the samples drawn.

The VI method tends to produce particularly good results, with the loss on the training set converging to around 0.150 while the loss on the testing data converges to around 0.4, as seen in Figure 2. This translates to good performance with an accuracy consistently around 95% and an F1-score around 96% when drawing 10000 samples. The choice of the learning rate and number of epochs are interdependent, with lower learning rate necessitating more training epochs in order to ensure convergence. Using higher learning rates speeds the training up but may lead to sub-optimal results, while too low of a learning rate may lead to a convergence which is too slow.

While MCMC tends to model the weights as normal distributions for all the weights, the VI method produces normally distributed weights only for the first 3 layers. The trace of the MCMC shows that the method has had good convergence overall. Overall, both methods seem to perform very well on this task with MCMC having more robust weight distributions than VI.

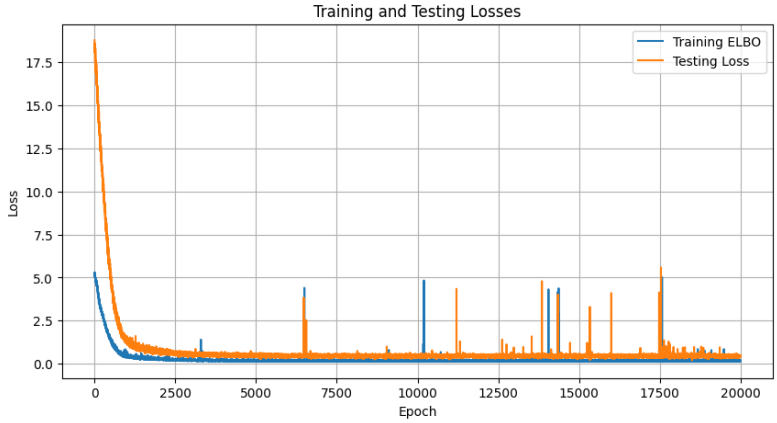
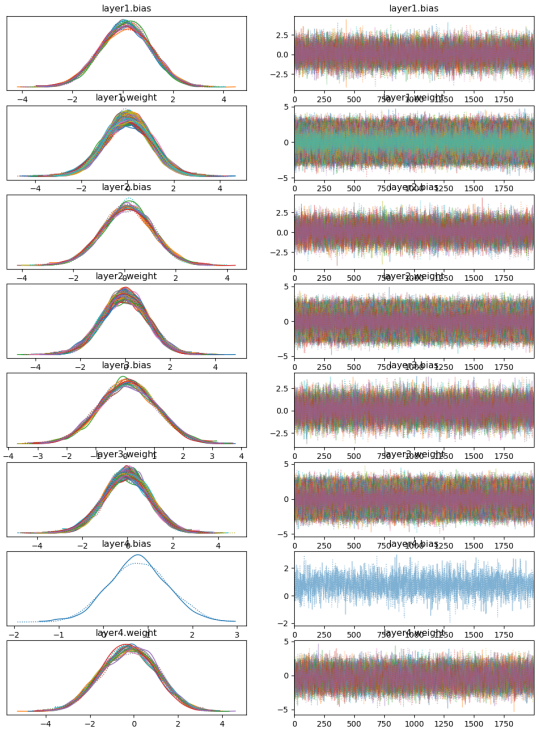
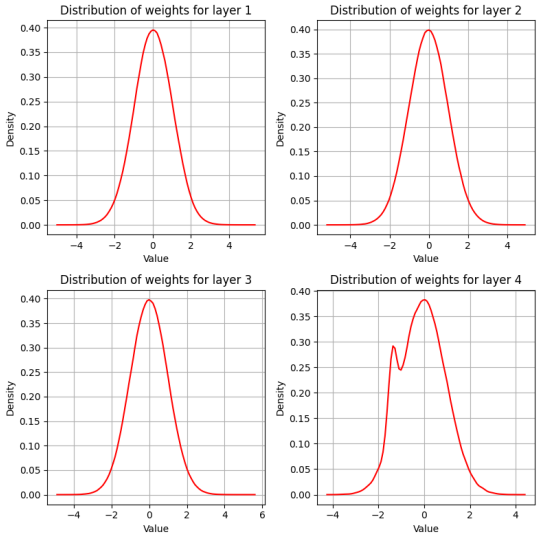


Figure 2: SVI Loss Per Epoch



(a) MCMC Weight Distribution and Trace



(b) SVI Weight Distribution

Figure 3: Comparison of Distribution of Weights for each Layers

5 Efficiency Comparison Between MCMC and VI

The MCMC method training is considerably slower than the VI method. It takes around 2 hours to run the MCMC variation while taking around 15 minutes to run the VI method on a CPU. As shown above the VI method also produces better accuracy and F1-score. Overall, both methods tend to produce predictions centred on the same mean with the VI method having slightly tighter tails, as shown in Figure 4, meaning it is slightly more confident than the MCMC method.

In Figure 5 we can see that the MCMC method has more data points which it is not too sure how to classify, while VI only has one point which is classifies equally unsure for each binary target class. The prediction uncertainties also tend to be a lot tighter for the VI implementation than the MCMC implementation. We can therefore say that in this respect VI tends to perform better than MCMC. Overall VI appears to have produced more accurate and certain predictions.

5.1 Applicability to Real-World Scenarios

As both models perform very well, the final point to take into consideration is the suitability of the methods to being implemented in a real-world setting. In the case of our data this would be a medical setting. This means that to deploy a BNN to diagnosis tumour cells we would want our method to be as accurate and interpretable as possible. While VI produces the best accuracy it does have the downside that its distributions draws are not made from the true posterior while the MCMC draws are. This means that there is a possibility that our predictions do not generalise well to new data points. We could expect that given the opportunity to run for longer, the MCMC's performance may be further increased in order to catch-up with the VI implementation. However, in this case, VI seems to have produced better predictions on our

testing data and so comes out on top.

With regards to uncertainty measurements, both methods allow us to draw multiple output samples for a given input which allow us to measure the uncertainty in the predictions. Both methods also have the benefit that the distributions of the models weights can be explored and analysed. VI here to has produced more certain results giving it an edge over MCMC.

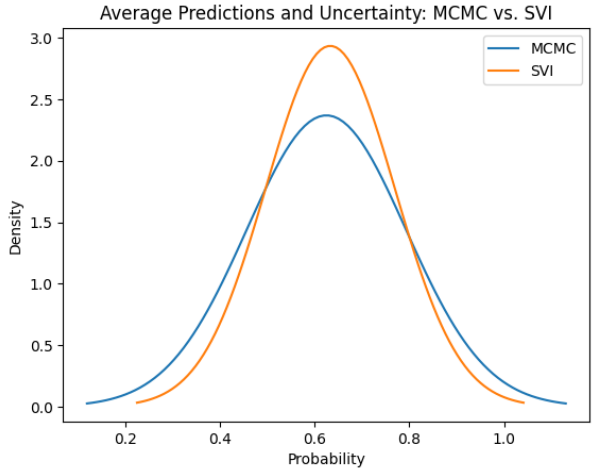
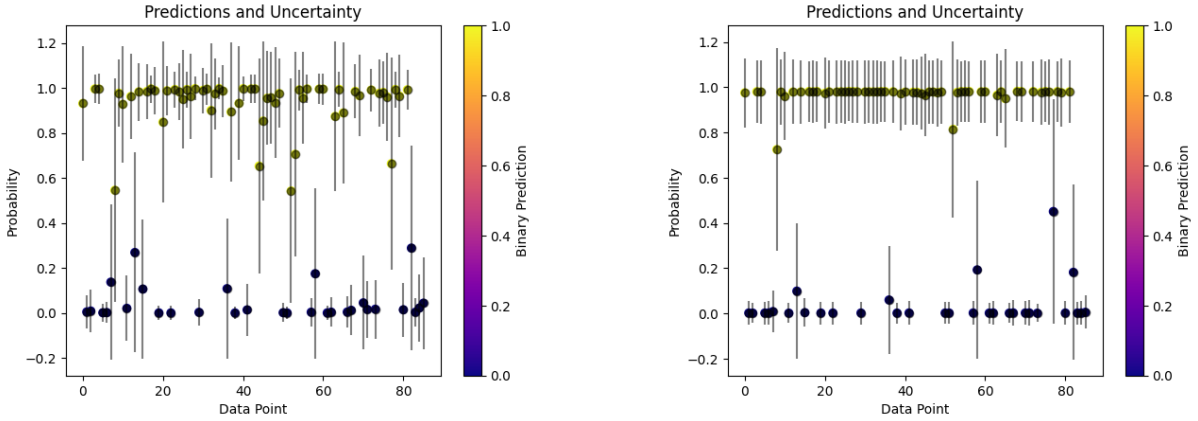


Figure 4: Comparison of Spread of Average Predictions for MCMC vs SVI



(a) MCMC Predictions and Uncertainties

(b) SVI Predictions and Uncertainties

Figure 5: Comparison of Model Predictions for Each Method

6 Conclusion

In conclusion, both MCMC and VI provide good ways to implement Bayesian methods into Neural Networks and assess the uncertainty in model weights and predictions. BNNs therefore provide an effective way to implement Bayesian analysis into NNs, improving the suitability of such models to real-world scenarios. It is likely such methods may become more popular over the coming years as AI is implemented in more sectors where uncertainty evaluation is important.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] Guillaume Baudart and Louis Mandel. Automatic guide generation for stan via numpyro. *arXiv preprint arXiv:2110.11790*, 2021.

[3] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *Journal of machine learning research*, 20(28):1–6, 2019.

[4] Andrew Gelman and Donald B Rubin. Inference from iterative simulation using multiple sequences. *Statistical science*, 7(4):457–472, 1992.

[5] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 2013.

[6] Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *Journal of machine learning research*, 18(14):1–45, 2017.

[10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[11] Martin Magris and Alexandros Iosifidis. Bayesian learning for neural networks: an algorithmic survey. *Artificial Intelligence Review*, pages 1–51, 2023.

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

[13] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[14] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[15] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[17] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[18] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[19] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[20] W Nick Street, William H Wolberg, and Olvi L Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *Biomedical image processing and biomedical visualization*, volume 1905, pages 861–870. SPIE, 1993.

[21] Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner. Rank-normalization, folding, and localization: An improved r for assessing convergence of mcmc (with discussion). *Bayesian analysis*, 16(2):667–718, 2021.

[22] Wojciech Wiewiórowski, Supervisory Opinions, Personal Data Breach, Joint Opinions, DPO Corner, and Press Kit. European data protection supervisor. *URL: https://edps. europa. eu/about-edps/members-mission/supervisors/wojciechwiewi% C3% B3rowski_en (last accessed: 11.05. 2021).*