Introduction to

Bayesian Neural Networks

Greg Johnson & Jason Wang

Math 538, Spring 2017

Contents

[I. Introduction 2](#_Toc482629540)

[Why Use a Neural Network? 2](#_Toc482629541)

[Why Use a Bayesian Neural Network? 2](#_Toc482629542)

[II. Neural Network Architecture 3](#_Toc482629543)

[One-Layer Neural Network 3](#_Toc482629544)

[Two-Layer Neural Network 4](#_Toc482629545)

[Tuning the Architecture 5](#_Toc482629546)

[III. Probability Specification 5](#_Toc482629547)

[Specifying the Prior Distributions 5](#_Toc482629548)

[Specifying the Likelihood 6](#_Toc482629549)

[Analytic Form of the Posterior and Posterior Predictive 6](#_Toc482629550)

[IV. Sampling the Posterior 7](#_Toc482629551)

[Gaussian Approximation 7](#_Toc482629552)

[Hamiltonian Monte Carlo & NUTS 7](#_Toc482629553)

[Variational Inference & ADVI 9](#_Toc482629554)

[V. Modeling Spiral Data 10](#_Toc482629555)

[Generalized Linear Model 10](#_Toc482629556)

[Neural Network 11](#_Toc482629557)

[Bayesian Neural Network 12](#_Toc482629558)

[VI. References 14](#_Toc482629559)

# I. Introduction

## Why Use a Neural Network?

Neural networks are extremely powerful statistical models with the flexibility to capture highly nonlinear real-world relationship. In a sense, they are the diametric opposite of linear models. Where linear models underfit, trading low bias for high variance, neural networks have the opposite tendency to overfit data (though they have safeguards built into their structure). It has been proven that if a neural network is complex enough, it may approximate *any* real-valued function to an arbitrary degree of accuracy (Goodfellow & Bengio, 2017).

Besides their flexibility, neural networks are known for being data-driven. Most statistical learners do not take raw data as input – they instead take features that have been carefully crafted by an expert in the problem domain. Neural networks remove this extra step of “human intervention” and instead learn the relevant features in a supervised manner. This is readily seen in image processing settings where a linear regression would produce useless results when given raw pixel data – the complex nonlinear relationships between pixels that make up the features necessary for accurate classification cannot be capture by a linear model. A neural network on the other hand will be able to identify and utilize these features (Goodfellow & Bengio, 2017).

## Why Use a Bayesian Neural Network?

Like any other model, neural networks must be completely specified before they can be trained. This includes things like the number of layers, the size of hidden layers, the amount of regularization, activation functions used, etc. The architecture of a neural network determines how well it performs for a given problem. Theoretically you want the optimal configuration of your neural network for your particular problem. This is not unlike the problem of finding the optimal budget parameter for a lasso model – something that can be easily estimated through cross-validation. Unfortunately, neural networks are too computationally costly to cross-validate across a grid of a single tuning parameter, much less ten or more. Another shortcoming is that neural networks give you point estimates as output. The model is too complex to produce prediction intervals whether through theory or bootstrapping.

Enter Bayesian neural networks. Priors can be placed on all of the parameters in the neural network, allowing the joint prior and the likelihood to combine and yield a joint posterior for the parameters. Estimates based on the posterior distributions can be used to find the optimal tuning parameters for the neural network. Further, any single vector of inputs will no longer yield a point estimate but a full posterior predictive distribution from which we may derive any number of distributional summaries (Wiecki, 2017).

In addition, Bayesian neural networks can incorporate prior knowledge about parameters (if previous architectures have been shown to be effective in this problem domain) and supplement small sample size (otherwise the likelihood will overpower most priors). Perhaps the biggest forte of the Bayesian treatment is the ability to let the data determine the network architecture and the tuning parameters instead of having the analyst set them beforehand based on guesswork and heuristics (Wiecki, 2017).

# II. Neural Network Architecture

## One-Layer Neural Network

In statistical terms, a neural network is a multistage non-linear multivariate regression. A one layer neural network takes the form:

where the input vector is fed into a multivariate regression (i.e. left-multiplied by the matrix of parameters ) to produce a vector of outputs . Then a nonlinear function (called an activation function in the neural network literature) is applied piece-wise to the vector ; is how the neural network function gets the “wiggle” that allows it to approximate nonlinear functions. The final result is the neural network prediction . Notice that an ordinary linear regression is the special case where is the identity function and logistic regression is the special case where is the softmax (also known as the inverse-logit) function.

## Two-Layer Neural Network

This process may be applied in multiple stages (or layers) as in the two-layer neural network:

where the input vector is put through one layer ( and ), yielding a vector of hidden values that we never see (and which make up the so-called hidden layer) then through the second ( and ) before yielding the final output. This format generalizes to an arbitrary number of layers. Figure 1 depicts a two-layer network as a directed acyclical graph.

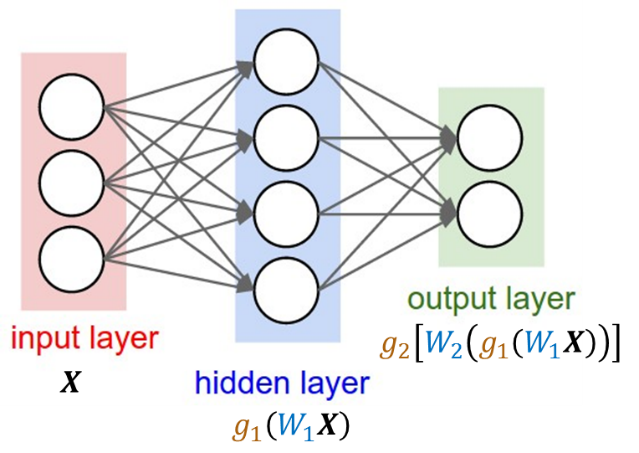


Figure . A Two-Layer Neural Network.

More than two layers designates a “deep” neural network that performs deep learning. Notice the sheer amount of processing taking place – the input vector is put through alternating nonlinear functions and linear combinations. The concatenation of these operations is how the overall neural network function has the flexibility to approximate any real-valued function.

## Tuning the Architecture

The number of layers and the size of the layers (number of hidden values) are pre-specified in frequentist treatments. It is possible to impose priors on both of these tuning parameters but we will only consider the latter for this paper.

Activation functions are normally chosen to be one of three functions: the hyperbolic tangent function, the sigmoid function, or the most common ReLu function – an identity function that is thresholded at zero. They are usually pre-specified and uniform within a layer (and often across). This leaves the matrices as the target of our estimation procedure. Ultimately our Bayesian treatment will involve placing priors on these parameter matrices and then investigating the posterior that results from incorporating information from the data (through the likelihood).

# III. Probability Specification

## Specifying the Prior Distributions

Because neural networks often involve many, many parameters, there usually isn’t any sophisticated background or expert information for each individual prior. Background knowledge on general architecture may be incorporated into the priors by specifying different priors by weight-type, essentially forming groups of weights. For example, you may wish to make the variance of the first-layer weights wider than that of the second-layer. Priors may also be specified hierarchically like assigning an inverse-gamma distribution to the variance of the weight prior; the hyperparameters of the inverse-gamma may also be given their own non-informative hyperpriors. Usually, the weights (if you vectorize all of the matrices and concatenate them) can be given a zero-mean multivariate Gaussian; if you don’t want to assign different variances to different groups of weight, you can simply use an identity covariance.

As a general illustration, a simple specification for a -layer neural network would be the zero-mean Gaussian prior:

where is the hyperparameter that corresponds to the variance of the weights.

## Specifying the Likelihood

The individual contribution of each data point to the likelihood is generally taken to be Gaussian. For our example, it takes the form:

where is our -layer neural network function and is another hyperparameter.

## Analytic Form of the Posterior and Posterior Predictive

In our example, the posterior takes the (somewhat) nice analytic form:

Note that the complexity of the neural network function in makes the analytic form of the posterior difficult to work with. Theoretically the posterior predictive requires marginalizing this posterior over the whole distribution network weights but in practice we use a simulation approach. Every draw from the posterior produces a vector of weights that determines a particular neural network architecture. We can then make a prediction using that particular architecture which serves as a posterior predictive distribution draw.

# *IV. Sampling the Posterior*

Traditional MCMC methods fail terribly when it comes to neural networks because of the sheer size of the model. A random walk based method cannot traverse the massive parameter space in an efficient manner. There are three common, alternative approaches to investigating the neural network posterior.

## Gaussian Approximation

The first method is to approximate the posterior distribution with a multivariate Gaussian distribution. The mean is taken to be the traditional point estimate of the neural network weights as obtained through back-propagation (i.e. gradient descent mixed with the chain rule). The covariance matrix is taken to be an estimate of the local curvature of the posterior around the mean. This approach is overly simplistic and has since been replaced with much more accurate and sophisticated techniques.

## Hamiltonian Monte Carlo & NUTS

Random walk MCMC methods fail in high dimensions because of their inefficiency and unfortunately reparameterization and efficient jumping rules don’t help. One solution is to speed up the algorithm’s traversal of the parameter space. MCMC was originally developed as a stochastic approach to modelling molecular states; an alternative, deterministic approach called Hamiltonian dynamics modelled molecular states with Newtonian mechanics. The combination of Hamiltonian dynamics and MCMC yields Hamiltonian (or Hybrid) Monte Carlo. In this synthesis, for every parameter of interest (in this case network weight) , we introduce an auxiliary momentum variable . It’s auxiliary because it’s probability density is totally independent of so we can factor it out when considering the joint posterior. What it does do is determine the movement of the algorithm as it traverses the parameter space.

The position and momentum of the algorithm is based on preservation of total energy as it explores the parameter space. The potential energy is proportional to the negative log posterior density of and the kinetic energy is proportional to the negative log density of . The total energy, otherwise known as the Hamiltonian, is the sum of these energies. As an intuitive example, imagine that the algorithm is climbing upwards towards a local maximum of the posterior density; as it climbs it loses potential energy and builds kinetic energy (because the energies are inversely proportional to the log-posterior densities). Once it reaches the maximum it will start to spend kinetic energy, perhaps rolling out of the local maximum towards a higher local maximum or the global maximum. In general, the algorithm will have positive momentum that allows it to traverse the many dimensions of the posterior very quickly.

The momentum variables are taken to be zero-mean multivariate Gaussian whose covariance matrix is referred to as the mass matrix. The mass matrix is either taken to be identity or proportional to the inverse covariance of the posterior of . Manipulating the mass matrix can only affect the efficiency of HMC.

For iterations the HMC algorithm takes the following steps:

1. Sample
2. Take **L** leapfrog steps (scaled by ) in parameter space (determined by the density and the distribution of ) to produce the proposal .
3. Update with the proposal with probability:

Note that the leapfrog steps are a discrete approximation of how the algorithm moves according to the total energy of the system. , the step size; , the number of leapfrog steps; and , the mass matrix are all tuning parameters of the HMC algorithm.

For difficult problems we want the HMC parameters to auto-tune to our posterior. and should adjust to the posterior curvature and should encourage the algorithm to traverse the entire posterior. The No U-Turn Sampler (NUTS) does just this. is auto-tuned so that the algorithm’s trajectory continues across the posterior until it’s finally forced to turn around (no premature U-turn); and are tuned during a warm-up phase.

## Variational Inference & ADVI

The variational inference approach (also called Variational Bayes) swaps the sampling problem of MCMC with an approximation/optimization problem. Variational Bayes attempts to directly approximate the posterior of with an approximating density that is indexed by the parameters . The algorithm is iterative, reducing the distance between the posterior and the approximating density by iteratively tweaking . Distance is measured as the Kullback-Leibler divergence:

The approximating density must be carefully chosen i.e. tailored to the specific posterior density. Expectation maximization is a special case of Variational Inference.

Automatic Differentiation Variational Inference (ADVI) is an automated form of variational inference. It automatically determines the approximating density as well as the algorithm for optimization; the only thing required for ADVI is a fully-specified Bayesian model and a dataset. ADVI takes a transformation-based approach where it first transforms latent variables to the real coordinate space (e.g. using the softmax function to transform a parameter confined to to the reals). Then it proposes a Gaussian on the transformed space (which has a corresponding non-Gaussian form on the original space), standardizes to zero-mean and identity covariance, and maximizes the evidence lower bound (ELBO; corresponds to Kullback-Leibler divergence). ADVI has been shown to be consistently faster than NUTS.

# V. Modeling Spiral Data

To illustrate the power and flexibility of the neural network approach we picked an artificial nonlinear dataset, specifically a 3-arm spiral dataset, pictured in Figure 2. Thus, we have two predictors and a categorical outcome of 3 classes. For comparison, we fit a generalized linear model, specifically a multinomial logistic regression; a basic neural network; and a Bayesian neural network.

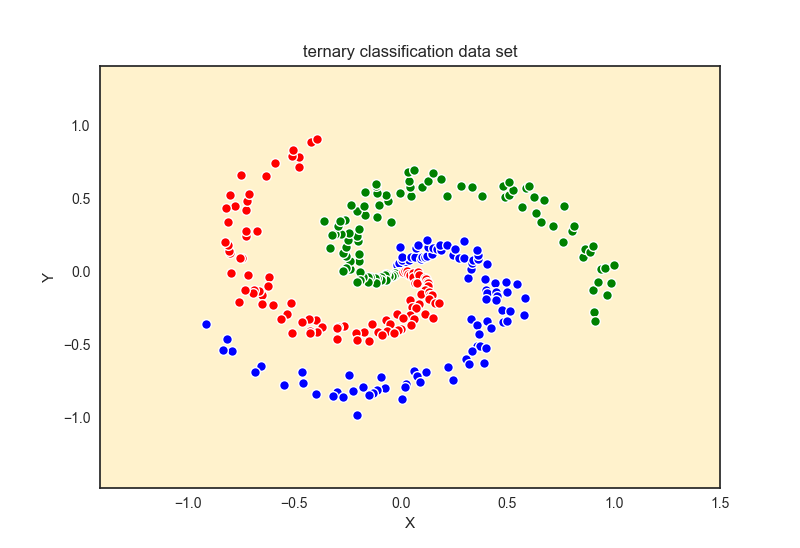


Figure . Three-Arm Spiral Data

## Generalized Linear Model

We fit a multinomial logistic regression to the data by minimizing the cross-entropy of the model classifications. The resultant group boundaries were, obviously, linear. Given the non-linear nature of the data, the GLM provided a very poor fit, as evidenced in Figure 3.

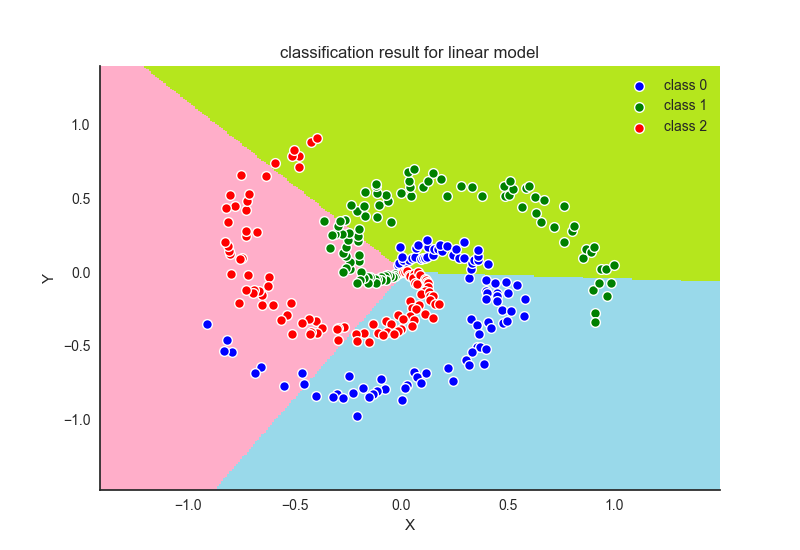


Figure . GLM Decision Boundaries

## Neural Network

Next we fit a neural network with three layers. The two hidden layers each consisted of 200 nodes. The ReLU activation function was used as well as a softmax classifier (essentially a logistic regression). This neural network can be thought of as two hidden layers of processing that then output derived features that can be easily classified by a logistic regression. The two inputs are essentially transformed into a new input space (of 200 dimensions) in which the 3 classes have linear classification boundaries. As can be seen in Figure 4, the neural network performs near-perfect classification and recovers the non-linear spiral classification boundaries. Unfortunately, we don’t get posterior probabilities, only point estimates.

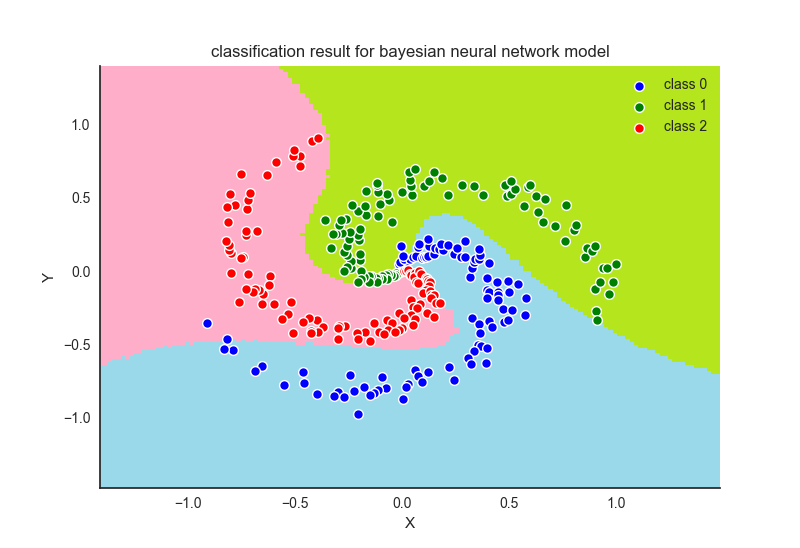


Figure . Neural Network Decision Boundaries

## Bayesian Neural Network

Our final model was a Bayesian neural network. The same architecture was used as in the previous model. A Hamiltonian Monte Carlo was attempted but the algorithm was extremely slow and failed to converge. Instead, ADVI was used to directly approximate the posterior of the network weights. Figure 5 shows how ADVI iteratively reduces the ELBO (a measure of distance between the posterior and the transformed Gaussian proposal of ADVI).

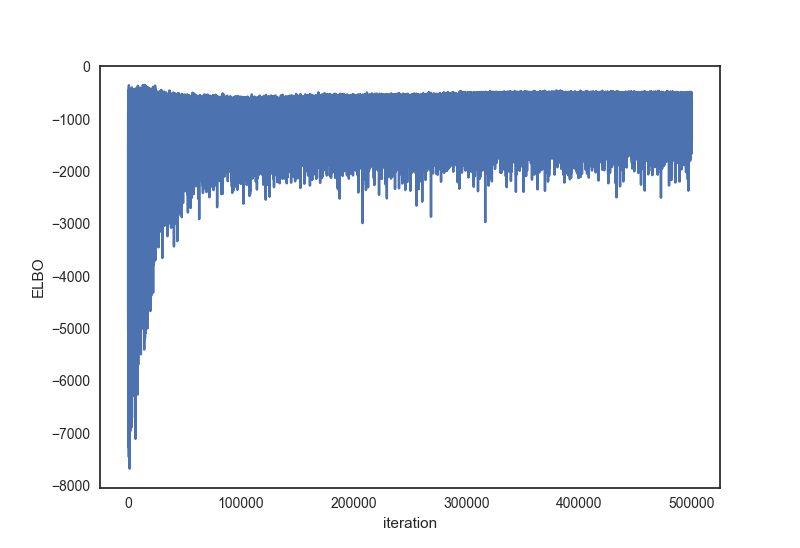


Figure . ADVI: Iterative Reduction in ELBO

Upon convergence, we simulated draws from the posterior predictive distribution of the neural network. The resultant uncertainty can be seen in Figure 6. Now in addition to the nice neural network decision boundaries we get an additional measure of uncertainty in prediction. In this case, the darker purple the region, the more uncertainty the Bayesian neural network has in predicting the class. Note that the asymmetry in uncertainty is mostly due to how close the blue and red spiral arms are (which is, in part, due to the random noise factor in how the data were generated). In the frequentist framework, a datum that fall in that purple region would get a hard classification as Class 1 or Class 2; in the Bayesian framework we can get a posterior probability of the three classes. Beautiful!

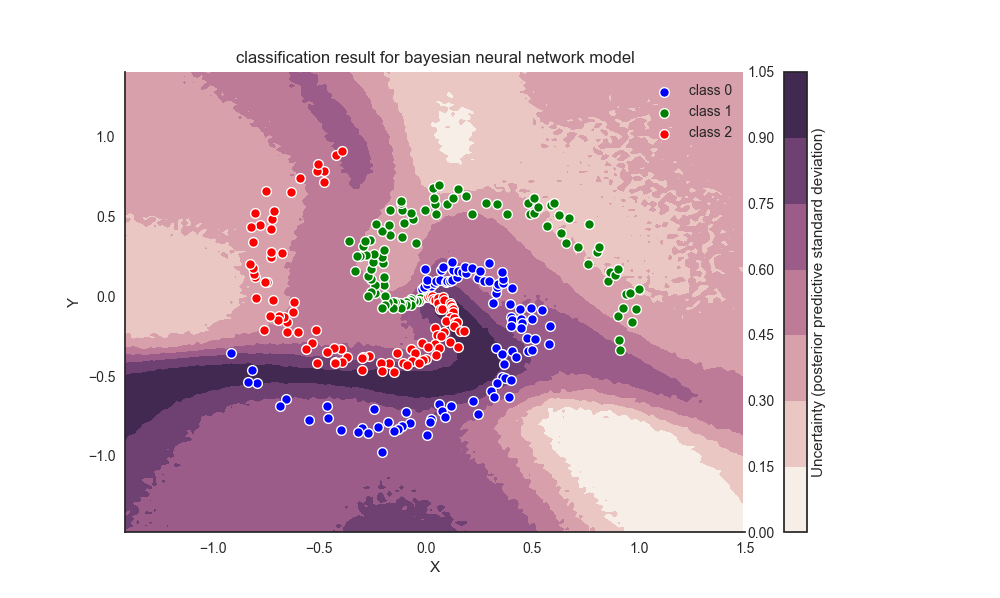


Figure . Bayesian Neural Network Decision Boundaries

# VI. References

Gelman, Andrew. *Bayesian Data Analysis*. CRC Press, 2013.

Goodfellow, Ian, and Yoshua Bengio. *Deep Learning*. MIT Press, 2017.

Wiecki, Thomas. “Bayesian Deep Learning*.”* Web blog post*. While My MCMC Gently Samples*. 1 Jun. 2016. Web. 19 Mar. 2017.

Rasmussen, Carl. “A Practical Monte Carlo Implementation of Bayesian Learning.” *NIPS'95 Proceedings of the 8th International Conference on Neural Information Processing Systems*, 1995, 598-604.

Neal, Radford. “MCMC using Hamiltonian Dynamics.” *Handbook of Markov Chain Monte Carlo*, edited by Gelman et al., Chapman & Hall, 2011, pp. 113-162.

Kucukelbir, Alp. “Automatic Variational Inference in STAN.” Cornell University Library, 2015.