Bayesian Optimisation using Neural Networks

by

Devang Agrawal (Queens')

Fourth-year undergraduate project in
Group F-, 2015/2016

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work.

Signed: _____ Date: _____

Summary

This project explored the use of neural networks for Bayesian Optimisation. No more than 100 words.

Contents

1 Introduction

Bayesian Optimisation is a very interesting problem. It has been used in .. . Neural networks can be used for dimensionality reduction and can be used for feature selection. It is not trivial how to build probabilistic models using Neural Networks. Several different options are available and two particular ones were explored in this report. Using a pragmatic approach .... . Alternatively a full Bayesian Neural network was used .... . These functions can be very useful for non-stationary functions.

2 Neural Networks

Neural networks represent multiple non-linear transforms of the input data[3] . Excellent at Feature Selection

3 Markov Chain Monte Carlo methods

Markov chain Monte Carlo(MCMC) methods are a class of algorithms for sampling from a desired probability distribution[1, 19]. A Markov chain is constructed with the desired distribution as its equilibrium distribution. These models are widely applied to Bayesian models in statistics. MCMC methods make no assumptions about the about the form of the underlying distribution. In theory they can take account of multiple modes and the possibility that the main contribution to the integral may come from areas not in the vicinity of any mode. In practise though it can under certain circumstances they can take a very long time to converge to the desired distribution. This is the main disadvantage of using MCMC methods.

In Bayesian learning we often encounter situations where we need to evaluate the expectation of a function $f(\theta)$ with respect to the posterior probability density of the model parameters. Writing the posterior as $Q(\theta)$, the expectation can be expressed as

$$E[f] \;=\; \int f(\theta)Q(\theta)d\theta \tag{3.1}$$

Such expectations can be estimated by the *Monte Carlo* method, using a sample of values from Q:

$$E[f] \;\approx\; \frac{1}{N}\sum_{t=1}^{N} f(\theta^{(t)}) \tag{3.2}$$

where $\theta^{(1)}, \ldots, \theta^{(N)}$ are generated by a process that results in each of them having the distribution defined by Q. In simple Monte Carlo methods, the $\theta^{(t)}$ are independent, however when Q is a complicated distribution , generating such independent values is

often infeasible. However it is often possible to generate a series of dependent values. The Monte Carlo integration formula of equation (3.2) still gives an unbiased estimate of $E[f]$ even when the $\theta^{(t)}$ are dependent as long as the independence is not too great[19]. The estimate will still converge to the true value as N increases.

Such a series of dependent values can be generated using a *Markov Chain* that has Q as its stationary distribution. The chain can be defined by giving an *initial distribution* for the first state of the chain, $\theta^{(1)}$, and a set of *transition probabilities* for a new state, $\theta^{(t+1)}$, to follow the current state, $\theta^{(t)}$. The probability densities for these transitions can be written as $T(\theta^{(t+1)}|\theta^{(t)})$. A *stationary (or invariant)* distribution, Q, is one that persists once it is established. This means that if $\theta^{(t)}$ has the distribution Q, then $\theta^{(t')}$ will have the same distribution for all $t' > t$. This invariance condition can be written as follows:

$$Q(\theta') = \int T(\theta'|\theta)Q(\theta)d\theta \tag{3.3}$$

The invariance with respect to Q is implied by the stronger *detailed balance* condition – that for all θ and θ':

$$T(\theta'|\theta)Q(\theta) = T(\theta|\theta')Q(\theta') \tag{3.4}$$

A chain satisfying detailed balance is said to be *reversible.*

An *ergodic* Markov chain has a unique invariant equilibrium distribution, to which it converges from any initial state. If we can find a Markov Chain that has Q as its equilibrium distribution, then we can find find expectations with respect to Q by using equation (3.2) . In this case $\theta^{(1)}, \ldots, \theta^{(N)}$ are the states of the chain, some of the early states can be discarded since they are not representative of the equilibrium distribution.

To use MCMC methods to estimate an expectation with respect to some distribution Q, we need to construct an ergodic Markov chain with Q as the equilibrium distribution. The chain should rapidly converge to this distribution and the states visited once equilibrium is reached should not be highly dependent.

3.1 The Metropolis algorithm

The Metropolis algorithm was introduced by Metropolis et.al in their classic paper in 1953[16] . In the Markov chain defined by the metropolis algorithm, a new state, $\theta^{(t+1)}$, is generated from the previous state, $\theta^{(t)}$, by first generating a *candidate state* using a specified *proposal distribution,* and then deciding whether or not to accept that state based on its probability density relative to the old state, with respect to the desired invariant distribution, Q. If accepted, the candidate state, becomes the next state of the Markov chain; if it is instead rejected, the new state remains the same as the old state.

In detail, the transition from $\theta^{(t)}$ to $\theta^{(t+1)}$ is defined as follows:

1. Generate a candidate state, θ^*, from a proposal distribution. The proposal distribution may depend on the current state, its density is by $S(\theta^* \,|\, \theta^{(t)})$. It is noted that the proposal distribution must be symmetrical, satisfying the condition $S(\theta' \,|\, \theta) = S(\theta \,|\, \theta')$.

2. If $Q(\theta^*) \geq Q(\theta^t)$, accept the candidate state; otherwise accept the candidate state with probability $Q(\theta')/Q(\theta)$. For $\theta' \neq \theta$, the overall transition probability is then given by:
$$T(\theta'|\theta) = S(\theta' \,|\, \theta) min(1, Q(\theta')/Q(\theta)) \tag{3.5}$$

3. If the candidate state is accepted, let $\theta^{t+1} = \theta^*$. However if it was not accepted, then set $\theta^{(t+1)} = \theta^{(t)}$.

Following the overall transition probability in equation (3.5), the detailed balance condition (3.4) can be verified as follows:

$$\begin{aligned} T(\theta'|\theta)Q(\theta) &= S(\theta' \,|\, \theta) min(1, Q(\theta')/Q(\theta))Q(\theta) \\ &= S(\theta' \,|\, \theta) min(Q(\theta), Q(\theta')) \\ &= S(\theta \,|\, \theta') min(Q(\theta'), Q(\theta)) \\ &= S(\theta \,|\, \theta') min(1, Q(\theta)/Q(\theta'))Q(\theta') \\ &= T(\theta|\theta')Q(\theta') \end{aligned}$$

Therefore the Metropolis updates leave Q invariant.

Many choices are available for the proposal distribution of the Metropolis algorithm. One simple and popular choice is a Gaussian distribution centred on $\theta^{(t)}$,with the variance chosen so that the probability of the candidate being accepted is reasonably high. Very low acceptance rates can be bad as they lead to successive samples being highly dependent. When sampling from a complex, high-dimensional distribution, the standard deviation of of the proposal distributions typically has to be very small to compared to the extent of Q. This is because large changes will almost certainly lead to regions of low probability. A large number of steps will be required to move to a distant point in the distribution. The problem is made worse by the fact that these movements will take the form of a random walk, rather than a systematic traversal.

Simple forms of the Metropolis algorithm can be very slow when applied to problems such as Bayesian learning for neural networks[19]. As will be seen in section (3.2) , this problem can be alleviated by using the hybrid Monte Carlo algorithm[8], in which candidate states are generated by a dynamical method which can avoid the random walk aspect of the exploration.

3.2 The hybrid Monte Carlo algorithm

The hybrid Monte Carlo algorithm was originally developed by Duane et.al. for application in quantum chromodynamics[8]. Radford Neal in his book on Bayesian Learning for Neural Networks[19]successfully applies this technique to Bayesian learning. The algorithm merges the Metropolis algorithm (reviewed in sec 3.1) with sampling techniques based on dynamical simulation. It generates a sample of points drawn from some specified distribution which can then be used to form Monte Carlo estimates for the expectations of various functions with respect to this distribution. For Bayesian learning, we often wish to sample from the posterior distribution given the training data, and are interested in estimating the expectations needed to make predictions for test cases.

The hybrid Monte Carlo algorithm is expressed in terms of sampling from the *canonical* (or *Boltzmann*) distribution for the state of a physical system, which is defined in terms of an energy function. The algorithm can be used to sample from any distribution for a set of real-valued variables for which the derivatives of the probability density can be computed. For describing the formulation of this algorithm it is convenient to retain the physical terminology even in non-physical contexts. The problem can then be described in terms of an energy function for a fictitious physical system.

Accordingly, suppose we wish to sample from some distribution for a "position" variable, $\mathbf{q}$, which has n real-valued components, q_i. When used for Bayesian neural networks in section (5) , $\mathbf{q}$ will be the set of $n$network parameters. The probability density for this variable under the canonical distribution is defined by :

$$P(\boldsymbol{q}) \propto \exp(-E(\mathbf{q})) \tag{3.6}$$

where $E(\mathbf{q})$ is the "potential energy" function. Any probability density which is not zero at any point can be put in this form, by simply defining $E(\mathbf{q}) = -\log P(\mathbf{q}) - \log Z$, for any convenient Z.

To allow the use of Hamiltonian dynamics, we can introduce a "momentum" variable, $\mathbf{p}$. The canonical distribution over the "phase space" of $\boldsymbol{q}$ and $\boldsymbol{p}$together is then defined to be

$$P(\mathbf{q}, \boldsymbol{p}) \propto \exp(-H(\mathbf{q}, \boldsymbol{p})) \tag{3.7}$$

where $H(\mathbf{q}, \boldsymbol{p}) = E(\mathbf{q}) + K(\mathbf{p})$ is the "Hamiltonian" function, which gives the total energy. $K(\mathbf{p})$ is the "kinetic energy" due to the momentum, for which the usual choice is :

$$K(\mathbf{p}) = \sum_{i=1}^{n} \frac{p_i^2}{2m_i} \tag{3.8}$$

The m_i are the "masses" associated with each component. It is possible to adjust the masses for each component to improve efficiency, however for the rest of the report and

project we take all of them to be one.

Since in the distribution of equation (3.7), $\boldsymbol{q}$ and $\boldsymbol{p}$ are independent, the marginal distribution of $\mathbf{q}$ is the same as that of equation (3.6), from which we intend to sample. We can proceed by defining a Markov chain that converges to the canonical distribution for $\boldsymbol{q}$ and $\boldsymbol{p}$. The values of $\boldsymbol{p}$ can then be simply ignored when estimating expectations of functions of $\boldsymbol{q}$.

The overall dynamics of the system with fixed total energy can be expressed by the following equations:

$$\frac{dq_i}{d\tau} = +\frac{\partial H}{\partial p_i} = \frac{p_i}{q_i} \tag{3.9}$$

$$dq_i = -\frac{\partial H}{\partial q_i} = -\frac{\partial E}{\partial q_i} \tag{3.10}$$

where τ, is the fictitious time in which the state evolves. To be able to run these dynamics, we must be able to compute the partial derivatives of E with respect to the the q_i. Radford Neal in his book[19] shows that the transformation preserves volume and is reversible. Therefore the transformation can be used as the transition operator for a Markov chain and will leave $P(\boldsymbol{q})$ invariant. Evolution under the Hamiltonian dynamics will not sample ergodically from $P(\boldsymbol{q}, \boldsymbol{p})$ because the value of the total energy H is constant. Regions with different values of H are never visited. To avoid this, HMC alternates between performing deterministic dynamical transitions and stochastic Gibbs sampling updates of the momentum. Since $\boldsymbol{q}$ and $\boldsymbol{p}$ are independent , $\boldsymbol{p}$ may be updated without reference to $\boldsymbol{q}$. For the kinetic energy function of equation (3.8), this is easily done, each of the p_i have independent Gaussian distributions which are trivial to sample from. These updates of $\boldsymbol{p}$ change the total energy H and hence allow the entire phase space to be explored.

The length in fictitious time of the trajectories is an adjustable parameter. It is generally better to use trajectories that result in large changes to $\boldsymbol{q}$ to avoid the random walk like effects that would result from randomizing the momentum after every short trajectory.

In practice, Hamiltonian dynamics can not be simulated exactly, but can only be approximated by some discretization using finite time steps. This will necessarily introduce numerical errors hence we need a scheme that minimizes the impact of these errors. In the *leapfrog* discretization, a single step finds the approximations to the position and momentum, $(\boldsymbol{q^*}, \boldsymbol{p^*})$ at time $\tau + \epsilon$ from $(\boldsymbol{q}, \boldsymbol{p})$ at time τ as follows:

$$p_i(\tau + \frac{\epsilon}{2}) \;=\; p_i(\tau) \;-\; \frac{\epsilon}{2}\frac{\partial E}{\partial q_i}(\mathbf{q}(\tau)) \tag{3.11}$$

$$q_i(\tau + \epsilon) \;=\; q_i(\tau) \;+\; \epsilon\frac{p_i(\tau + \frac{\epsilon}{2})}{mi} \tag{3.12}$$

$$p_i(\tau + \epsilon) \;=\; p_i(\tau + \frac{\epsilon}{2}) \;-\; \frac{\epsilon}{2}\frac{\partial E}{\partial q_i}(\mathbf{q}(\tau + \epsilon)) \tag{3.13}$$

The leapfrog step consists of a half-step for the p_i , followed by a full step for q_i, and another half-step for the p_i. To follow the dynamics for some period of time, $\Delta\tau$, a value of ϵ is chosen which is small enough to give an acceptable error, and equations (3.11) - (3.13) are applied for $L = \Delta\tau/\epsilon$ steps in order to reach the target time. The momentum variable is negated at this point to ensure that if we were to perform a dynamical transition starting at the end stage, we will get back to the initial stage.

In the *leapfrog* discretization scheme, the phase space volume remains preserved despite the discretization. The dynamics are also easily reversible. However, the value of H no longer remains exactly constant, this can introduce bias in the simulation.

HMC cancels these effects exactly by adding a Metropolis accept/reject stage after L leapfrog steps. If the original state is $(\boldsymbol{q}, \boldsymbol{p})$, and we get to the new state $(\boldsymbol{q^*}, \boldsymbol{p^*})$ after L leapfrog steps, the new state is then treated as a proposal state and is accepted with probability:

$$min(1, \frac{\exp(-H(\boldsymbol{q^*}, \boldsymbol{p^*}))}{\exp(-H(\boldsymbol{q}, \boldsymbol{p}))} \tag{3.14}$$

In detail , given the number of of leapfrog steps, L, a dynamical transition is performed as follows:

1. Start from the current state, $(\boldsymbol{q}, \boldsymbol{p}) = (\hat{\boldsymbol{q}}(0), \hat{\boldsymbol{p}}(0))$. Perform L leapfrog steps with a step-size of ϵ, resulting in the state $(\hat{\boldsymbol{q}}(\epsilon L), \hat{\boldsymbol{p}}(\epsilon L))$

2. Negate the momentum variables, to produce the proposal state $(\boldsymbol{q^*}, \boldsymbol{p^*}) = (\hat{\boldsymbol{q}}(\epsilon L), -\hat{\boldsymbol{p}}(\epsilon L))$.

3. Accept $(\boldsymbol{q^*}, \boldsymbol{p^*})$ as the new state with probability:

$$min(1, \frac{\exp(-H(\boldsymbol{q^*}, \boldsymbol{p^*}))}{\exp(-H(\boldsymbol{q}, \boldsymbol{p}))}$$

   otherwise let the new state be the same as the old.

It is important to maintain satisfactory acceptance rates by tuning the step-sizes ϵ and the number of leapfrog steps L. A simple adaptive version of HMC is used in this report as implemented in the code accompanying [20]. We track the average acceptance rate of the HMC move proposals, using an exponential moving average. If the average acceptance

larger than a target acceptance rate, we can increase the step-size ,ϵ, to improve the mixing rate of the Markov chain. If the acceptance rate is too low, the step-size can be decreased to improve the acceptance rate but yielding a more conservative mixing rate.

3.3 Intriguing properties of Neural Networks

Some interesting properties of neural networks have been recently investigated.

4 Bayesian Optimisation

Bayesian Optimisation is the model based optimisation of black box functions.

4.1 High Dimensional Bayesian Optimisation

An original goal of this project was to explore Bayesian Optimisation in high dimensions . This has not been thoroughly explored in the available literature. However two good papers have tried to tackle this problem.

4.1.1 Rembo

Developed by Ziyu wang ..

4.1.2 Add-GP-UCB

Was developed at CMU by ..

5 Bayesian Optimisation with Bayesian Neural Networks

5.1 Bayesian Neural Networks

Neural networks discussed in Section 2, used maximum likelihood to determine the network parameters. Regularized maximum likelihood can be interpreted as MAP (maximum a posteriori) approach in which the regularizer can be viewed as the logarithm of a prior parameter distribution. However, a full Bayesian treatment of neural networks, requires us to marginalize over the posterior distribution of parameters[5, 19]. In this section we discuss an implementation of a Bayesian neural network in which network parameters are updated using the hybrid Monte Carlo algorithm. The hyperparameters are updated separately by using Gibbs sampling.

Bayesian learning for neural networks is a difficult problem, due to the typically complex nature of the posterior distribution. The hybrid Monte Carlo(HMC) algorithm is

particularly suitable for this problem, due to its avoidance of random walk behaviour as discussed in section 3.2.

A neural network can be parametrized by its weights and biases, collectively denoted as $\boldsymbol{\theta}$, that define what network from input to output is denoted by the network. This function can then be written as $f(\mathbf{x}, \boldsymbol{\theta})$, where x is the input to the network. A prior can be defined on the network parameters, this prior can depend on some hyperparameters, γ. The prior density for the parameters can be written as $P(\theta \,|\, \gamma)$ and the prior density for the hyperparameters can be written as $P(\gamma)$. We have a training data-set given by $\mathbf{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$, consisting of independent pairs of input values, $x^{(i)}$, and target values, $y^{(i)}$. We aim to model the conditional distribution of target values given the input values. The required conditional probability density is given by $P(y \,|\, x, \theta, \gamma)$.

The ultimate objective is to predict the target value for new test cases, $y^{(n+1)}$, given the corresponding inputs, $x^{(n+1)}$,using the information in the training set. To make the prediction we require the posterior distribution for $\boldsymbol{\theta}$ and $\boldsymbol{\gamma}$, this is proportional to the product of the prior and the likelihood due to the training cases:

$$P(\boldsymbol{\theta}, \boldsymbol{\gamma} \,|\, \mathbf{D}) \;\propto\; P(\gamma) P(\theta|\gamma) \prod_{c=1}^{n} P(y^{(c)} \,|\, x^{(c)}, \boldsymbol{\theta}, \boldsymbol{\gamma}) \tag{5.1}$$

Predictions on new data-points can then be made by integration with respect to this posterior distribution. The prediction on new data-points is then given by:

$$P(y^{(n+1)}|x^{(n+1)}, \boldsymbol{D}) \;=\; \int P(y^{(n+1)}|x^{(n+1)}, \boldsymbol{\theta}, \boldsymbol{\gamma}) P(\boldsymbol{\theta}, \boldsymbol{\gamma} \,|\, \mathbf{D})\, d\theta\, d\gamma \tag{5.2}$$

For a regression model, the prediction that minizes the expected squared-error loss is the mean of the predictive distribution. It is given by:

$$\hat{y}^{(n+1)} \;=\; \int f(x^{(n+1)}, \boldsymbol{\theta}) P(\boldsymbol{\theta}, \boldsymbol{\gamma} \,|\, \mathbf{D})\, d\theta\, d\gamma \tag{5.3}$$

Since these integrals take the form of expectation of functions with respect to the posterior distribution, they can be tackled by using a Markov chain Monte Carlo approach. We can approximate the integral with the average value of the function over a sample of values from the posterior.

The hybrid Monte Carlo method discussed in section 3.2 is used to generate samples from the posterior. To apply this method, we first need to formulate the desired distribution in terms of a potential energy function. Since the objective is to sample from the posterior distribution for network parameters, the energy function will be a function of these parameters. The parameters $\boldsymbol{\theta}$, will play the role of the the "position" variables, $\boldsymbol{q}$, of an imaginary physical system. The hyperparameters remain fixed throughout one hybrid Monte Carlo update. Hence we can ignore all energy terms depending only on the hyperparameters. For the generic case described by equation (5.1) , the potential energy

can be calculated by taking the negative log of the prior and the likelihood due to the trainig cases, as follows:

$$E(\boldsymbol{\theta}) = F(\boldsymbol{\gamma}) - \log P(\theta|\gamma) - \sum_{c=1}^{n} P(y^{(c)} \mid x^{(c)}, \boldsymbol{\theta}, \boldsymbol{\gamma}) \tag{5.4}$$

where $F(\boldsymbol{\gamma})$ is any convenient function of the hyperparameters. The detailed form of the energy function will obviously depend on the network architecture, the prior, and the data model used. For the purpose of this project we put a Gaussian prior with mean zero and standard deviation σ_p on each of the network parameters. We also assume that the targets are a single real value which can be modelled with a Gaussian noise distribution with standard deviation σ_n. The hyperparameters are then $\boldsymbol{\gamma} = \{\tau_p\,,\, \tau_n\}$, where $\tau_p = \sigma_p^{-2}$ and $\tau_n = \sigma_n^{-2}$. The variances are expressed in terms of the associated preision for convenience. The resulting potential energy function is :

$$E(\boldsymbol{\theta}) \;=\; \tau_p \sum_{i=1}^{k} \frac{\theta_i^2}{2} + \tau \sum_{c=1}^{n} \frac{(y^{(c)} - f(x^{(c)}, \theta))^2}{2}$$

where k is the total number of parameters in the neural net i.e $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_k\}$, this includes both the weights and the biases associated with the network.

It can be noted that this energy function is similar to the error function which is minimized for training regularized conventional networks. However the objective here is not to minimise the total energy , but rather to sample from the canonical distribution represented by the energy.

We can proceed by introducing the momentum variable, $\boldsymbol{p}$. The "position" variable, $\boldsymbol{q}$ can be associated with the parameters $\boldsymbol{\theta}$ of the network. We then use the hybrid Monte Carlo algorithm to generate samples from the posterior of the parameters. These samples are then used to approximate the integral in equation (5.3) to make predictions on new data-points.

We plan to use the predictions from the neural network to perform Bayesian Optimisation using the GP-LCB acquisition function[6], which is given by:

$$LCB(x) \;=\; \mu(x) - K\sigma(x) \tag{5.5}$$

where $\mu(x)$ is the prediction at point x as given by equation (5.6) , and $\sigma(x)$ is the standard deviation at point x calculated in equation (5.7).

$$\mu(x) = \int f(x, \boldsymbol{\theta}) P(\boldsymbol{\theta}, \boldsymbol{\gamma} \mid \mathbf{D})\, d\theta\, d\gamma \tag{5.6}$$

$$\sigma(x) = \sqrt{\int (f(x, \boldsymbol{\theta}) - \mu(x))^2 P(\boldsymbol{\theta}, \boldsymbol{\gamma} \mid \mathbf{D})\, d\theta\, d\gamma} \tag{5.7}$$

With N , samples from the posterior distribution of the parameters,$\{\boldsymbol{\theta_1}, \ldots, \boldsymbol{\theta_N}\}$, we can approximate the required integrals by using MCMC methods to give:

$$\mu(x) = \frac{1}{N} \sum_{i=1}^{N} f(x, \boldsymbol{\theta_i}) \tag{5.8}$$

$$\sigma(x) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (f(x, \boldsymbol{\theta_i}) - \mu(x))^2} \tag{5.9}$$

5.2 Results

A neural network with 3 hidden layers each of width 50 units was used. The *tanh* non-linearity was used in the neural network. A one dimensional synthetic sinusoidal dataset given by :

$$f(x) = \sin(7x) + \cos(17x) \qquad x \in [-1, 1]$$

Some tests were run on the algorithm given above. We show a synthetic sinusoidal function and show the fit of a few samples from the posterior of the neural network. The samples were then averaged according to equations (5.8) - (5.9) to generate the overall fit of the neural network. The credible interval was taken to be two standard deviations on either side of the mean.

5.3 Using a hybrid model

A model where the last layer of hidden units are fed to the BLR to obtain

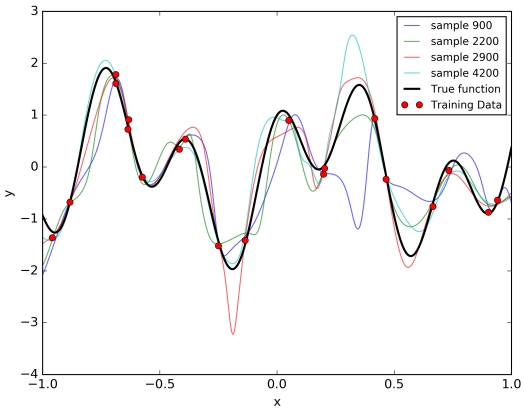

Figure 1: The fit of neural networks from four different samples from the posterior distribution of the network parameters

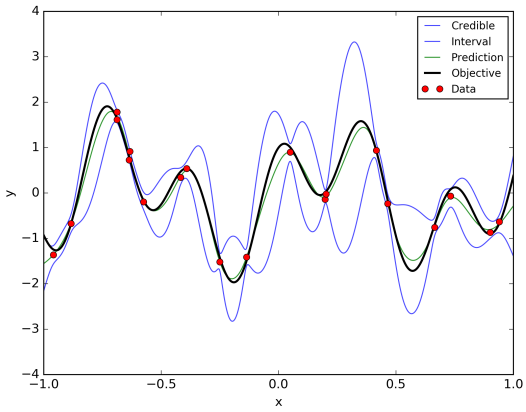

Figure 2: The overall fit of the neural network, averaged over a large number of samples from the neural network

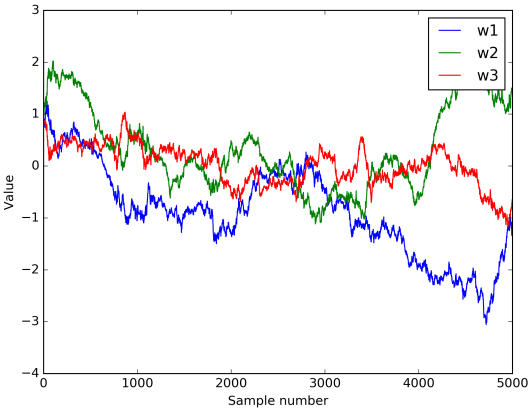

Figure 3: Trace plots of

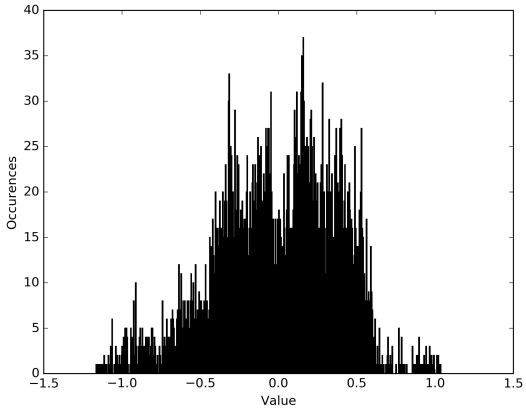

Figure 4: The overall fit of the neural network, averaged over a large number of samples from the neural network

5.3.1 Results

5.4 Using a full Bayesian neural network

5.4.1 Markov Chain Monte Carlo methods

5.4.2 Results

5.5 Results and Discussion

Maybe like this

6 Conclusions

The results are very promising

6.1 Potential applications/Future work

This system promises to provide better results for high-dimensional functions. Add Nilesh's paper to this

References

[1] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

[2] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[3] Yoshua Bengio, Ian J. Goodfellow, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2015.

[4] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[5] Christopher M Bishop. *Pattern recognition and machine learning*, volume 1. springer, 2006.

[6] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[7] Misha Denil, Loris Bazzani, Hugo Larochelle, and Nando de Freitas. Learning where to attend with deep architectures for image tracking. *Neural computation*, 24(8):2151–2184, 2012.

[8] Simon Duane, A.D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216 – 222, 1987.

[9] Robert F Dugan Jr, Rebecca M Dugan, and Corinna Trabucco. Healthy computer use for computer science. *Journal of Computing Sciences in Colleges*, 27(1):9–15, 2011.

[10] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[11] Geoffrey E Hinton and Ruslan R Salakhutdinov. Using deep belief nets to learn covariance kernels for gaussian processes. In *Advances in neural information processing systems*, pages 1249–1256, 2008.

[12] Kirthevasan Kandasamy, Jeff Schneider, and Barnabas Poczos. High dimensional bayesian optimisation and bandits via additive models. 03 2015.

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

[14] Daniel J Lizotte, Tao Wang, Michael H Bowling, and Dale Schuurmans. Automatic gait optimization with gaussian process regression. In *IJCAI*, volume 7, pages 944–949, 2007.

[15] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[16] 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.

[17] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129):2, 1978.

[18] Radford M Neal. Probabilistic inference using markov chain monte carlo methods. 1993.

[19] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 1996.

[20] Marc' Aurelio Ranzato, Alex Krizhevsky, and Geoffrey E. Hinton. Factored 3-way restricted boltzmann machines for modeling natural images, 2010.

[21] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[22] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable bayesian optimization using deep neural networks. 02 2015.

[23] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, and Nando de Freitas. Bayesian optimization in a billion dimensions via random embeddings. *arXiv preprint arXiv:1301.1942*, 2013.

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

A Risk Assessment Retrospective

The risk assessment submitted at the start of the project identified frequent computer use as the main potential hazard. This assessment was accurate and no other hazards were encountered during the execution of the project. To avoid repetitive stress injury from excessive computer use, frequent breaks were taken by the author. Several exercises and best practices[9] were employed to reduce the risk further. If the project is to be carried out again, the risk assessment will remain the same.