AMSI VACATION RESEARCH SCHOLARSHIPS 2019–20

EXPLORE THE MATHEMATICAL SCIENCES THIS SUMMER

Design of Neural Networks Using Randomised Algorithms

Jonathan Wilton

Supervised by Professor Dirk Kroese

The University of Queensland

Vacation Research Scholarships are funded jointly by the Department of Education and the Australian Mathematical Sciences Institute.

Abstract

Machine Learning attributes a considerable amount of its success and popularity to the rise of Deep Learning — a framework that allows computers to make accurate predictions by learning sophisticated patterns in large datasets, while also taking full advantage of modern computational power. At the heart of Deep Learning is the *Neural Network* which describes classes of flexible parametric functions. Neural Networks can be combined to form an ensemble in order to learn patterns in data in a more robust manner — improving predictive performance on unseen data. The main downside of using an ensemble of Neural Networks is choosing its design for a given problem to ensure sufficient performance. We investigate Bayesian optimisation and its use in the design of Neural Network ensembles along with methods for making efficient implementations on a computer.

1 Introduction

Making predictions using large amounts of data is commonly referred to as Machine Learning. In Machine Learning the workflow is typically as follows: choose a model or approximating class of functions, collect appropriate data, choose the optimal member of the approximating class of functions and use it to make predictions on new data. Supervised Learning is a successful area in Machine Learning which is concerned with being able to make predictions y given some input features $\boldsymbol{x}$. For example, $\boldsymbol{x}$ could represent pixel values in an image and y could take on values 0 or 1 depending on whether the image is of a dog or a cat; or $\boldsymbol{x}$ could denote the elapsed time in an experiment and y the spread of a certain disease.

Neural Networks encode rich classes of nonlinear parametric functions that have seen increasing popularity in recent years due to the increase in available data and computational power. Neural Networks are able to learn a deep hierarchy of abstract features of data in order to make good predictions — this is where the term Deep Learning arises. In recent years it has become clear that both the sparsity and structure of Neural Networks are essential for efficient learning [1, 2]. One way of introducing sparsity into a neural network is to combine multiple Neural Networks into an ensemble which allows for greater generalisation ability to unseen data.

A drawback of adopting the use of ensembles of Neural Networks is the large amount of hyperparameters[1] associated with it; these hyperparameters are things such as the size of the networks and number of networks to include in the ensemble. The values of these hyperparameters are typically fixed before training of Machine Learning models and have a significant effect on the model's performance. Bayesian Optimisation [3] is a modern approach to global optimisation which exploits ideas from Bayesian statistics in order to make informed decisions about the direction to take the optimisation, this is particularly important when the time cost of testing a hyperparameter configuration is high. Details on how to effectively implement each step of Bayesian optimisation with Gaussian process surrogate models for the optimisation of the design of deep Neural Network ensembles on a computer are discussed in this report with a view to making the implementation efficient and

[1]Note that the term *hyperparameters* has a different definition in Machine Learning and in Bayesian statistics. In Machine Learning, hyperparameters refer to parameters whose value is fixed by a human and are not learnt during training. Hyperparameters in Bayesian statistics refer to parameters of a prior distribution. Hopefully the use of the word hyperparameter in this report will be clear from context.

easily generalisable to problems not considered in this project.

2 Statement of Authorship

Jonathan Wilton and Dirk Kroese devised a research topic for this project and had ongoing discussions regarding ideas and directions to take it. Jonathan Wilton devised the main methods of the project, wrote the Python code, conducted numerical experiments and drafted the report. Dirk Kroese supervised the project, provided frequent consultation sessions to aid in Jonathan's progress, suggest sources of information and proof-read drafts of the report. AMSI funded the project.

3 Background

3.1 Machine Learning Objective Functions

An attempt is made to use a notation system that is, in order of importance, simple, descriptive, consistent, and compatible with historical choices. See appendix A.1 for some notation conventions that will be used throughout the report.

The Machine Learning paradigm for making predictions starts with a *prediction function*[2] $g : \mathbb{R}^p \to \mathbb{R}$ which takes as input features $\boldsymbol{x}$ and outputs a guess $g(\boldsymbol{x})$ for y, sometimes denoted as $g(\boldsymbol{x}) = \widehat{y}$. The accuracy of a prediction $\widehat{y}$ with respect to a given response y is measured by a *loss function* $\text{Loss}(y, \widehat{y})$. When labels take on continuous values we call the problem *regression*, a common example of loss function here is the squared-error loss: $\text{Loss}(y, \widehat{y}) = (y - \widehat{y})^2$. When the labels take on discrete values we call the problem classification and the zero-one loss function $\text{Loss}(y, \widehat{y}) = \mathbb{1}\{y \neq \widehat{y}\}$ is often used. The expected loss, or *risk* for g is defined as

$$\ell(g) = \mathbb{E}\ \text{Loss}(Y, g(\boldsymbol{X})).$$

Here the expectation is taken over all possible input-output pairs[3] $(\boldsymbol{X}, Y)$. An important result shows how to find the function g^* that optimises the risk in the case of squared error Loss.

Theorem 3.1.1: For the squared error loss $\text{Loss}(y, \widehat{y}) = (y - \widehat{y})^2$, the solution to $g^* = \underset{g}{\text{argmin}}\ \mathbb{E}\ \text{Loss}(Y, g(\boldsymbol{X}))$ is given by $g^*(\boldsymbol{x}) = \mathbb{E}[Y|\boldsymbol{X} = \boldsymbol{x}]$.

Proof. See [4] for proof. □

Denote by $\mathcal{T} = \{(\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)\}$ the random sample of *training data* with n *examples* obtained by the data collection process with corresponding observation $\tau = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$. Since the risk $\ell(g)$ depends on the joint distribution of $\boldsymbol{X}$ and Y, which is typically unknown, it is common practice to approximate

[2] Prediction functions can also be used to predict multiple outputs at a time, resulting in functions of the form $g : \mathbb{R}^p \to \mathbb{R}^m$.
[3] It may be useful to interpret this as an expectation over all possible data that we could have seen from a data collection process.

the risk by the *training loss*

$$\ell_{\mathcal{T}}(g) = \frac{1}{n}\sum_{i=1}^{n} \text{Loss}(Y_i, g(\boldsymbol{X}_i)), \tag{1}$$

which estimates the risk (expected loss) of g without bias using the training set. The functional optimisation problem of minimising the training loss (as an approximation of the risk) with respect to g can be simplified by assuming g is a parametric function, resulting in a parametric optimisation problem. For example if $g(\boldsymbol{x}) = \boldsymbol{b}^{\mathsf{T}}\boldsymbol{x}$ is a linear function, then we call the function $g_{\mathcal{T}}$ in

$$g_{\mathcal{T}} = \underset{\boldsymbol{b}\in\mathbb{R}^p}{\text{argmin}}\ \ell_{\mathcal{T}}(g), \tag{2}$$

the *learner* of the training set $\mathcal{T}$. The quintessential problem in Machine Learning is the overfitting–underfitting tradeoff: observe that taking $g_{\mathcal{T}}$ to be the function defined by $g_{\mathcal{T}} : \boldsymbol{x}_i \longrightarrow y_i$ for all examples in the training set and undefined elsewhere will give zero training loss and hence will be a minimizer in (2). However, this particular function will have poor performance when making predictions on inputs not in the training set. In this case we say that the function $g_{\mathcal{T}}$ has *overfit* to the training set. The prediction accuracy of new pairs of data is measured by the *generalization risk* of a learner

$$\ell(g_\tau) = \mathbb{E}\ \text{Loss}(Y, g_\tau(\boldsymbol{X})).$$

For an observed training set τ. In the case of a random training set $\mathcal{T}$, the generalization risk is thus a random variable which depends on $\mathcal{T}$. One can take the expectation of such a random variable to give the *expected generalisation risk*

$$\mathbb{E}\ \ell(g_{\mathcal{T}}) = \mathbb{E}\ \text{Loss}(Y, g_{\mathcal{T}}(\boldsymbol{X})),$$

where the $(\boldsymbol{X}, Y)$ in the expectation above are distributed independently of $\mathcal{T}$. For an outcome τ of the training data, the generalisation risk can be estimated without bias by the *test loss*

$$\ell_{\mathcal{T}'}(g_\tau) = \frac{1}{n'}\sum_{i=1}^{n'} \text{Loss}(Y_i', g_\tau(\boldsymbol{X}_i')),$$

where $\mathcal{T}' = \{(\boldsymbol{X}_1', Y_1'), \ldots, (\boldsymbol{X}_{n'}', Y_{n'}')\}$ is called the *test sample* containing new examples that you want to test the learner on. The test sample $\mathcal{T}'$ has the same distribution as $\mathcal{T}$ but the two samples are independent. The goal in Machine Learning is typically to find a learner g_τ which generalises well to unseen data, that is, admits a small test loss $\ell_{\mathcal{T}'}(g_\tau)$.

3.2 Neural Networks

Neural Networks represent parametric functions whose output is the repeated function composition of simple component-wise non-linear functions. A type of parametric function with a Neural Network structure (See

Figure 1) is a *multilayer perceptron* (MLP):

$$g(\boldsymbol{x}) = \boldsymbol{h}_L(\boldsymbol{h}_{L-1}(\cdots(\boldsymbol{h}_1(\boldsymbol{x})))), \tag{3}$$

with $\boldsymbol{h}_l(\boldsymbol{x}) = \boldsymbol{S}_l(\mathbf{W}_l\boldsymbol{x} + \boldsymbol{b}_l)$, the component functions of $\boldsymbol{S}_l(\boldsymbol{z}) = (S(z_1), \ldots, S(z_d))$ are *activation functions* and $\mathbf{W}_l, \boldsymbol{b}_l$ are learnable parameters of the MLP called weight matrices and bias vectors respectively. Some common choices of activation functions are the logistic sigmoid function $S(x) = (1 + \exp(-x))^{-1}$ and more recently the *rectified linear unit* (ReLU) $S(x) = \max\{x, 0\}$. The non-linear vector-valued function $\boldsymbol{S}_L$ on the output layer h_L is chosen in accordance with the desired data type of the output of the neural network; the identity function $\boldsymbol{S}_L(\boldsymbol{x}) = \boldsymbol{x}$ is commonly used for outputs in regression problems[4], allowing each output value to take on values in $\mathbb{R}$.

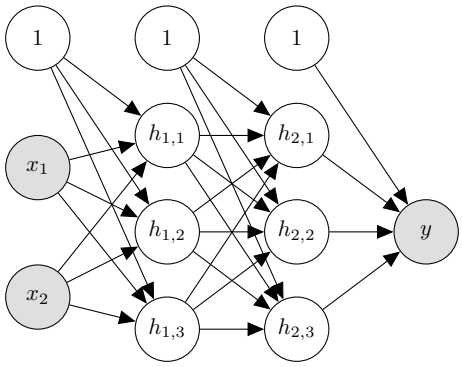

Figure 1: MLP with two dimensional input $\boldsymbol{x} = (x_1, x_2)^\mathsf{T}$, one dimensional output y, two hidden layers, each 3 units wide. The nodes with a 1 inside them will be referred to as *bias nodes*. The connections between nodes not involving the bias nodes represent elements in the *weight matrices* $\mathbf{W}_l$, and the connections involving bias nodes represent elements of the *bias vectors* $\boldsymbol{b}_l$. The $h_{i,j}$ represents the j-th node in hidden layer i and forms $\boldsymbol{h}_i(\boldsymbol{z}) = (h_{i,1}(\boldsymbol{z}), \ldots, h_{i,\dim(\boldsymbol{z})}(\boldsymbol{z}))$. We shaded the *input layer* neurons and *output layer* neurons to indicate that these are observed in the graphical model; that is, on a *forward pass* of the network, we input $\boldsymbol{x} = (x_1, x_2)^\mathsf{T}$ and try and match the output of the network with the label y corresponding to $\boldsymbol{x}$ which are both observed in training data.

MLPs with both logistic sigmoid and ReLU activations are universal approximators [5, 6]. In 2011 an influential paper [7] showed that Neural Networks with ReLU activations train more efficiently and effectively compared to networks with logistic sigmoid activations[5]. A universal approximation theorem for *deep but shallow* ReLU networks (the current state of the art designs) is stated in Theorem 3.2.1.

Theorem 3.2.1: *For any Lebesgue-integrable function $f : \mathbb{R}^p \to \mathbb{R}$ and any $\varepsilon > 0$, there exists a fully-connected ReLU network g with width $D \leqslant p + 4$, such that*

$$\int_{\mathbb{R}^n} |f(\boldsymbol{x}) - g(\boldsymbol{x})| \, \mathrm{d}\boldsymbol{x} < \varepsilon.$$

[4] A function called *softmax* is a common choice for classification problems, see [4] for more information.
[5] Consequently as of 2017, ReLU has taken over the most widely used activation function for Neural Networks [8, 9]

Proof. For a proof and further information see [10]. □

The result in Theorem 3.2.1 does not say how deep (how many hidden layers, L) are required in the network to get the desired level of accuracy ε. In practice it is common to take the width of the network past $p+4$ to compensate for not knowing how deep to go, and the results by doing this seem to work well.

Given a training set $\tau = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, an MLP g can be trained to learn the mapping $\boldsymbol{x} \to y$ (with $(\boldsymbol{x}, y)$ having the same distribution as the one which generated τ) in the hope that it can generalise well. This train process is formulated as an unconstrained parametric optimisation problem with the objective function being the training loss as in (1) and the variables that are being optimised over are the weight matrices and bias vectors of the MLP. Such an optimisation problem is typically solved using algorithms such as stochastic gradient descent, Adadelta [11] or Adam [12]. These methods require the derivative of the training loss function to be taken with respect to each value in the weight matrices and bias vectors — this can be done using the *Backpropagation* algorithm which exploits the chain rule for differentiation [4, 13].

3.3 Ensembles

An ensemble in Machine Learning refers to a learner that is a combination of many other learners. The predictive power (as measured by the expected generalisation risk) of an ensemble is stronger than any of the individual learners in the ensemble. To see this we first decompose the expected generalisation risk of a learner $g_{\mathcal{T}}$ [4]:

$$\mathbb{E}\,\ell(g_{\mathcal{T}}) = \underbrace{\ell^*}_{\text{irreducible risk}} + \underbrace{\mathbb{E}(\mathbb{E}(g_{\mathcal{T}}(\boldsymbol{X})|\boldsymbol{X}) - g^*(\boldsymbol{X}))^2}_{\text{expected squared bias}} + \underbrace{\mathbb{E}[\mathrm{Var}[g_{\mathcal{T}}(\boldsymbol{X})|\boldsymbol{X}]]}_{\text{expected variance}}$$

where the irreducible risk is $\ell^* = \ell(g^*)$ and g^* is as in Theorem 3.1.1 — the minimiser of the risk with no restriction of the class of functions that g belongs to. Denote by $g_{\mathcal{T}}^{(1)}, \ldots, g_{\mathcal{T}}^{(N)}$ distinct learners of a training set $\mathcal{T}$. Form the ensemble

$$\bar{g}_{\mathcal{T}}(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} g_{\mathcal{T}}^{(i)}(\boldsymbol{x}) \tag{4}$$

as a new learner for $\mathcal{T}$, then the expected variance term in the expected generalisation risk is smaller for $\bar{g}_{\mathcal{T}}$ than it is for any of $g_{\mathcal{T}}^{(i)}, i = 1, \ldots, N$. However, it will not decrease to zero, because each learner in the ensemble is typically correlated due to the unavailability of independently and identically (iid) distributed training sets to train each individual learner on. The exact behaviour is summarised in the following result; whose proof follows from decomposing the variance of a sum into the sum of covariances:

Theorem 3.3.1: Let $\mathcal{T}_1, \ldots, \mathcal{T}_N$ be random samples of a training set which have the same distribution as $\mathcal{T}$ but are correlated with each other and let corresponding learners of each training set be denoted by $g_{\mathcal{T}}^{(1)}, \ldots, g_{\mathcal{T}}^{(N)}$. Suppose that the positive pairwise correlation between any two of the learners is ϱ and $\mathrm{Var}(g_{\mathcal{T}}^{(i)}) = \sigma^2$ for each $i = 1, \ldots, N$. If

$$\bar{g}_{\mathcal{T}}(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} g_{\mathcal{T}}^{(i)}(\boldsymbol{x})$$

then

$$\mathrm{Var}(\bar{g}_{\mathcal{T}}) = \varrho\sigma^2 + \sigma^2\frac{(1-\varrho)}{N}. \tag{5}$$

So the variance of the ensemble is bounded below by the constant $\varrho\sigma^2$ in such a case. Increasing the number of learners in the ensemble, N, will decrease the variance of the ensemble and decreasing the pair-wise correlation between each learner ϱ will decrease the lower bound of the variance. Decreasing ϱ is typically done by either using the same class of predictor functions and getting each learner to learn a bootstrapped training set (known as *bagging* [4]), or to create N separate training sets by performing random feature selection on an original training set $\mathcal{T}$ and train train a new learner on each set (known as a *random forests* [4]). In this project we reduce the pairwise correlation by training multiple learners from different classes of prediction functions on the same training set and use these to form an ensemble such as in (4). Each learner is an MLP with the same number of hidden layers L and width of each hidden layer D, however random elements of the weight matrices and bias vectors are set to zero with probability $(1-p)$ — equivalent to randomly removing connections in Figure 1 (See Figure 2 for an example of such an ensemble).

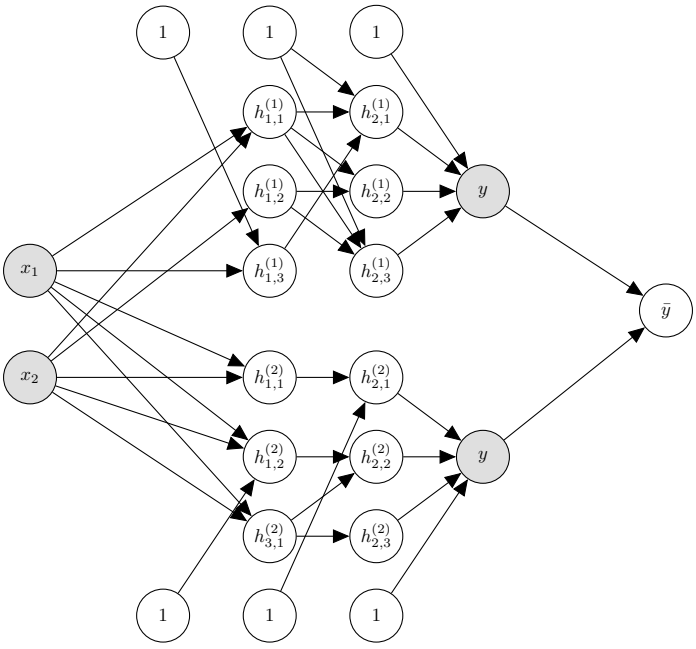

Figure 2: Ensemble of two MLPs.

When using an ensemble such as in Figure 2 to solve a regression problem with training set τ, handling the choice of the ensemble design L, D, N and p can be done by minimising the generalisation risk $\ell(\bar{g}_{\tau}) =: \mathcal{L}_{\star}(\boldsymbol{\theta})$ as a function of $\boldsymbol{\theta} = (L, D, N, p) \in \mathscr{O}$. Gradient based methods can't be used to solve this optimisation problem because $\mathcal{L}_{\star}$ is not differentiable and we only have access to noisy evaluations of $\mathcal{L}_{\star}$ using the test loss $\ell_{\tau'}(\bar{g}_{\tau}) =: \mathcal{L}(\boldsymbol{\theta})$ for a given test set τ'. Evaluations of the test loss are also noisy due to the use of stochastic optimisation methods in training and the cost of time to make evaluations of $\mathcal{L}(\boldsymbol{\theta})$ is high as a result of training N separate neural networks and testing the corresponding ensemble that they form. *Bayesian Optimisation* is

a global optimisation algorithm that is suitable for solving such optimisation problems.

3.4 Bayesian Optimisation Using Gaussian Processes

Machine Learning algorithms have hyperparameters which greatly affect their performance, as measured by test loss. Denote by $\boldsymbol{\theta}$ the vector containing values of hyperparameters for some Machine Learning algorithm (in the case of an ensemble of MLPs, $\boldsymbol{\theta} = (L, D, N, p)$). To tune the vector $\boldsymbol{\theta}$ to minimise test loss, the cost of time involved with trying a configuration of $\boldsymbol{\theta}$ is large due to the requirement of training each network in the ensemble and doing testing. Bayesian optimisation [3] provides a way of making informed decisions about good values of $\boldsymbol{\theta}$ to try next based on the performance of previously tested values of $\boldsymbol{\theta}$.

The process of Bayesian optimisation is as follows: use previously tested values of $\boldsymbol{\theta}$ to form a sample $\mathscr{T} = \{(\boldsymbol{\theta}_1, \mathcal{L}_1), \ldots, (\boldsymbol{\theta}_k, \mathcal{L}_k)\}$ ($\boldsymbol{\theta}_i$ is the design of the i-th tested ensemble and $\mathcal{L}_i$ is the corresponding value of the test loss $\mathcal{L}(\boldsymbol{\theta}_i)$), use Gaussian Process (GP) regression to guess what the performance of untested values of $\boldsymbol{\theta}$ are as well as corresponding confidence intervals then maximise an acquisition function to say which $\boldsymbol{\theta}$ has the highest expected improvement over the current best as measured by the generalisation risk.

3.4.1 Gaussian Process Regression

A GP [4, 14] on a space $\mathscr{O}$ is a stochastic process $\{\mathcal{L}_{\boldsymbol{\theta}}, \boldsymbol{\theta} \in \mathscr{O}\}$ where, for any choice of indices $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k \in \mathscr{O}$, the vector $(\mathcal{L}_{\boldsymbol{\theta}_1}, \ldots, \mathcal{L}_{\boldsymbol{\theta}_k})$ has a multivariate Gaussian distribution. As such, the distribution of a GP is completely specified by its mean and covariance functions $\mu : \mathscr{O} \to \mathbb{R}$ and $\kappa : \mathscr{O} \times \mathscr{O} \to \mathbb{R}$, respectively. The functions μ, κ are typically parametric functions with parameters collected into the vector $\boldsymbol{\vartheta}$. The covariance function is a *positive semidefinite function* (See appendix A.2).

We are wanting to use GP regression [14] to approximate the generalisation risk $\mathcal{L}_\star(\boldsymbol{\theta})$ of an average weights ensemble of MLP's trained on τ as a function of $\boldsymbol{\theta}$. Recall that exact evaluations of the generalisation risk are typically intractable, hence they are approximated by noisy evaluations of the test loss $\mathcal{L}(\boldsymbol{\theta})$.

Suppose we have tested the performance of some ensembles and record the results in $\mathscr{T} = \{(\boldsymbol{\theta}_1, \mathcal{L}_1), \ldots, (\boldsymbol{\theta}_k, \mathcal{L}_k)\}$ consisting of input output pairs of ensemble designs $\boldsymbol{\theta}_i$ and corresponding test loss evaluations $\mathcal{L}_i = \mathcal{L}(\boldsymbol{\theta}_i)$. The aim is to infer information about $\mathcal{L}_\star(\boldsymbol{\theta})$ on the basis of $\mathscr{T}$. The GP regression framework starts by supposing that each observation

$$\mathcal{L}(\boldsymbol{\theta}_i) = \mathcal{L}_\star(\boldsymbol{\theta}_i) + \varepsilon_i, \quad \varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_0^2) \tag{6}$$

for $i = 1, \ldots, k$ (σ_0^2 treated as known). A GP prior is put on the unknown function

$$\mathcal{L}_\star(\boldsymbol{\theta}) \sim GP(\mu(\boldsymbol{\theta} \mid \boldsymbol{\vartheta}), \kappa(\boldsymbol{\theta}, \boldsymbol{\theta}' \mid \boldsymbol{\vartheta}))$$

for some mean function μ and positive definite covariance function κ (σ_0^2 is an element of $\boldsymbol{\vartheta}$). Denote by $\boldsymbol{\Theta}$ the design matrix of $\mathscr{T}$ and by $\boldsymbol{l}$ the vector containing all values of $\mathcal{L}_i$ in $\mathscr{T}$ with corresponding random vector $\boldsymbol{L}$.

The model in (6) gives rise to the conditional distribution

$$\boldsymbol{L} \mid \mathcal{L}_\star, \boldsymbol{\Theta}, \boldsymbol{\vartheta} \sim \mathcal{N}(\mathcal{L}_\star(\boldsymbol{\Theta}), \sigma_0^2 \mathbf{I})$$

with $\mathbf{I}$ being an appropriately sized identity matrix. The *marginal* distribution is

$$\boldsymbol{L} \mid \boldsymbol{\Theta}, \boldsymbol{\vartheta} \sim \mathcal{N}(\mu(\boldsymbol{\Theta} \mid \boldsymbol{\vartheta}), \kappa(\boldsymbol{\Theta}, \boldsymbol{\Theta} \mid \boldsymbol{\vartheta}) + \sigma_0^2 \mathbf{I})$$

which follows by properties of the Gaussian distribution. Here $\mu(\boldsymbol{\Theta} \mid \boldsymbol{\vartheta})$ is evaluated row-by-row and $\kappa(\boldsymbol{\Theta}, \boldsymbol{\Theta} \mid \boldsymbol{\vartheta})$ is a Gram matrix (see appendix A.2). Denote by $\boldsymbol{\Theta}_*$ the design matrix of some ensemble designs which haven't been tested yet, then we can model the joint distribution

$$\begin{pmatrix} \boldsymbol{L} \\ \mathcal{L}_\star(\boldsymbol{\Theta}_*) \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu(\boldsymbol{\Theta} \mid \boldsymbol{\vartheta}) \\ \mu(\boldsymbol{\Theta}_* \mid \boldsymbol{\vartheta}) \end{pmatrix}, \begin{pmatrix} \kappa(\boldsymbol{\Theta}, \boldsymbol{\Theta} \mid \boldsymbol{\vartheta}) + \sigma_0^2 \mathbf{I} & \kappa(\boldsymbol{\Theta}, \boldsymbol{\Theta}_* \mid \boldsymbol{\vartheta}) \\ \kappa(\boldsymbol{\Theta}_*, \boldsymbol{\Theta} \mid \boldsymbol{\vartheta}) & \kappa(\boldsymbol{\Theta}_*, \boldsymbol{\Theta}_* \mid \boldsymbol{\vartheta}) \end{pmatrix} \right).$$

It can be shown that the *predictive* distribution $\mathcal{L}_\star(\boldsymbol{\Theta}_*) \mid \mathscr{T}, \boldsymbol{\vartheta} \sim \mathcal{N}(\mu_*(\boldsymbol{\Theta}_* \mid \mathscr{T}, \boldsymbol{\vartheta}), \kappa_*(\boldsymbol{\Theta}_* \mid \mathscr{T}, \boldsymbol{\vartheta}))$ is such that

$$\mu_*(\boldsymbol{\Theta}_* \mid \mathscr{T}, \boldsymbol{\vartheta}) = \mu(\boldsymbol{\Theta}_* \mid \boldsymbol{\vartheta}) + \kappa(\boldsymbol{\Theta}_*, \boldsymbol{\Theta} \mid \boldsymbol{\vartheta})[\kappa(\boldsymbol{\Theta}, \boldsymbol{\Theta} \mid \boldsymbol{\vartheta}) + \sigma_0^2 \mathbf{I}]^{-1}(\boldsymbol{l} - \mu(\boldsymbol{\Theta} \mid \boldsymbol{\vartheta}))$$
$$\kappa_*(\boldsymbol{\Theta}_* \mid \mathscr{T}, \boldsymbol{\vartheta}) = \kappa(\boldsymbol{\Theta}_*, \boldsymbol{\Theta}_* \mid \boldsymbol{\vartheta}) - \kappa(\boldsymbol{\Theta}_*, \boldsymbol{\Theta} \mid \boldsymbol{\vartheta})[\kappa(\boldsymbol{\Theta}, \boldsymbol{\Theta} \mid \boldsymbol{\vartheta}) + \sigma_0^2 \mathbf{I}]^{-1}\kappa(\boldsymbol{\Theta}, \boldsymbol{\Theta}_* \mid \boldsymbol{\vartheta})$$

Notice that the uncertainty in predicting the performance of untested ensembles is decreased in terms of Loewner partial-order (see appendix A.3) when the design is similar to those that have already been tested.

3.4.2 Mean Functions

When doing GP regression, it is common practice to use a mean function which is identically equal to zero $\mu \equiv 0$ [4, 14]. However when trying to predict on inputs which are away from observed data, the predictive distribution behaves like the prior distribution hence will be centred about the prior mean function. The performance of Bayesian optimisation depends heavily on having a good GP model. The GP prior mean can be thought of as an exploration hyper-parameter — increasing the constant value of the mean function will decrease exploration of Bayesian optimisation in favour of exploitation. Similarly, decreasing the mean function will increase exploration and decrease exploitation. The choice of

$$\mu(\boldsymbol{\Theta} \mid \boldsymbol{\vartheta}) = \min(\boldsymbol{l}) + 0.5\,(\text{mean}(\boldsymbol{l}) - \min(\boldsymbol{l})) \tag{7}$$

strikes a balance between exploration and exploitation (rewarding more exploration of unexplored regions of $\mathscr{O}$ as opposed to rewarding the exploitation of regions of $\mathscr{O}$ close to the location of the current optimum)[6].

[6]The handling of a constant mean function can also be done by including it in the vector $\boldsymbol{\vartheta}$ as in [15].

3.4.3 Covariance Functions

The performance of the GP regression depends mostly on the covariance function κ. The form of the covariance function determines a priori what type of functions to expect the data to be generated from — usually encoding information about smoothness and number of times the function is differentiable. The hyperparameters $\boldsymbol{\vartheta}$ in $\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}' \mid \boldsymbol{\vartheta})$ control how rapidly the data generating function is allowed to vary for certain small changes in the input and how uncertain we should be in the predictive distribution about training points and away from training points. Some common covariance functions include the *squared exponential* covariance function [15]

$$\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}' \mid \boldsymbol{\vartheta}) = \vartheta_0 \exp\left(-\frac{1}{2} r^2(\boldsymbol{\theta}, \boldsymbol{\theta}')\right), \quad r^2(\boldsymbol{\theta}, \boldsymbol{\theta}') = \sum_{i=1}^{d} (\theta_i - \theta_i')^2 / \vartheta_i^2$$

and the Matérn 5/3 kernel

$$\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}' \mid \boldsymbol{\vartheta}) = \vartheta_0 \left(1 + \sqrt{5r^2(\boldsymbol{\theta}, \boldsymbol{\theta}')} + \frac{5}{3} r^2(\boldsymbol{\theta}, \boldsymbol{\theta}')\right) \exp\left(-\sqrt{5r^2(\boldsymbol{\theta}, \boldsymbol{\theta}')}\right). \tag{8}$$

The latter is recommended in [15] for Bayesian optimisation of Machine Learning hyperparameters.

3.4.4 Acquisition Functions

An acquisition function $a : \mathscr{O} \to \mathbb{R}$ is used to determine which values of $\boldsymbol{\theta} \in \mathscr{O}$ provide the most promise of minimising $\mathcal{L}_\star(\boldsymbol{\theta})$ based on the GP regression model for $\mathcal{L}_\star$. The function a depends on the history $\mathscr{T}$ and GP parameters $\boldsymbol{\vartheta}$. The next value of $\boldsymbol{\theta}$ to try is given by

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \mathscr{O}}{\operatorname{argmax}}\ a(\boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta}) \tag{9}$$

Denote by $\phi(\cdot), \Phi(\cdot)$ the probability density function (pdf) and cumulative distribution function (cdf) of the standard normal distribution $\mathcal{N}(0, 1)$ and $\mathcal{L}^- := \min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta})$. Two common acquisition functions are:

Probability of Improvement: An intuitive strategy is to maximise the probability of improvement over the current best value. Using GPs, this can be computed analytically [15]

$$a(\boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta}) = \mathbb{P}\left(\mathcal{L}_\star(\boldsymbol{\theta}) < \mathcal{L}^-\right) = \Phi\left(Z\right), \quad Z = \frac{\mathcal{L}^- - \mu_*(\boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta})}{\kappa_*(\boldsymbol{\theta}, \boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta})}$$

Expected Improvement: Alternatively, one could choose to maximize the expected improvement (EI) over the current best. This also has a closed form under the GP [15]:

$$a(\boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta}) = \mathbb{E} \max\{\mathcal{L}^- - \mathcal{L}_\star(\boldsymbol{\theta}), 0\} = Z\kappa_*(\boldsymbol{\theta}, \boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta})\Phi(Z) + \kappa_*(\boldsymbol{\theta}, \boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta})\phi(Z)$$

In both cases, $\mathcal{L}_\star(\boldsymbol{\theta})$ is drawn from the current predictive distribution and μ_*, κ_* are the corresponding predictive mean and covariance functions.

The choice of the next iterate in (9) results in poor performance of the Bayesian optimisation when using an unsuitable $\boldsymbol{\vartheta}$. The choice of a good $\boldsymbol{\vartheta}$ can be automated by putting a hyper-prior distribution $p(\boldsymbol{\vartheta})$ on all possible GPs, then using the *integrated acquisition* (in place of the regular acquisition) which is an average of the acquisition function over the hyper-posterior distribution $p(\boldsymbol{\vartheta} \mid \mathscr{T})$

$$a(\boldsymbol{\theta} \mid \mathscr{T}) = \int a(\boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta}) p(\boldsymbol{\vartheta} | \mathscr{T}) \, \mathrm{d}\boldsymbol{\vartheta} \tag{10}$$

$$= \mathbb{E}_{\boldsymbol{\vartheta} \sim p(\boldsymbol{\vartheta} | \mathscr{T})}[a(\boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta})]. \tag{11}$$

The expectation in (11) can be approximated with a Monte Carlo estimate [16]

$$\mathbb{E}_{\boldsymbol{\vartheta} \sim p(\boldsymbol{\vartheta} | \mathscr{T})}[a(\boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta})] \approx \frac{1}{K} \sum_{i=1}^{K} a(\boldsymbol{\theta} \mid \mathscr{T}, \boldsymbol{\vartheta}_i)$$

after acquiring samples from $p(\boldsymbol{\vartheta} \mid \mathscr{T})$ using MCMC sampling.

3.5 MCMC Sampling

Markov chain Monte Carlo (MCMC) is a Monte Carlo sampling technique for (approximately) generating samples from an arbitrary distribution referred to as the target distribution. A refresher/introduction to MCMC sampling is given in appendix A.4 with an overview of a common method known as the Metropolis Hastings (MH) sampler. There is one major drawback of using the MH sampler, which is having to choose the proposal density. A remedy is to avoid having to make such a decision on which proposal density to use and have a method which is robust to exploration procedure such as slice sampling [15].

3.5.1 Slice Sampler

Slice sampling [17] is a method for sampling from a distribution with pdf of the form $p(\boldsymbol{x}) = p^*(\boldsymbol{x})/Z$ that only utilizes the function p^*. Like MH, it creates a Markov chain of states but unlike MH it only depends on a width parameter $W \in \mathbb{R}$ rather than a proposal distribution. This parameter W can be made adaptive so the method is robust. An algorithm for adaptive width parameter slice sampling is in appendix A.4.2 and is a modification of slice sampling as presented in the lecture delivered by David MacKay [18].

4 Bayesian Optimisation for Design of Deep Neural Network Ensembles

The original contribution of this project is an effective strategy for automatically finding an ensemble of MLPs to solve a regression problem. In a regression problem, let $\tau = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ denote the training set that is an outcome of a random training set $\mathcal{T}$, $\tau' = \{(\boldsymbol{x}'_1, y'_1), \ldots, (\boldsymbol{x}'_{n'}, y'_{n'})\}$ be an outcome of the random test set $\mathcal{T}'$ distributed independently of $\mathcal{T}$. Let $g_\tau^{(1)}, \ldots, g_\tau^{(N)}$ be learners of the train set τ with corresponding

sample mean ensemble

$$\bar{g}_\tau(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} g_\tau^{(i)}(\boldsymbol{x}) \tag{12}$$

whose design is characterised by the vector of quantities $\boldsymbol{\theta} = (L, D, N, p)$. Formulate the optimisation problem

$$\min_{\boldsymbol{\theta}\in\mathscr{O}} \mathcal{L}_\star(\boldsymbol{\theta}) \tag{13}$$

with Loss being the squared error loss. The problem is to minimise the generalisation risk of the neural network ensemble given by (12) as a function of $\boldsymbol{\theta} = (L, D, N, p)$. To tackle the optimisation problem in (13) using Bayesian optimisation, we use GP prior mean function μ given as in (7) and following the advice laid out in [15], the covariance function is chosen to be the Matérn 5/3 kernel in (8). A good choice of hyper-prior on $\boldsymbol{\vartheta}$ in the integrated acquisition function is to have each element independently and identically distributed as Half-Cauchy random variables[7] with pdf [19]

$$p(\vartheta) = 2\mathbb{1}\{\vartheta \geqslant 0\}/\left(\pi(1+\vartheta^2)\right).$$

Hence the joint distribution of $\boldsymbol{\vartheta}$ is[8]

$$p(\boldsymbol{\vartheta}) = \prod_{i=1}^{6} \frac{2}{\pi(1+\vartheta_i^2)}\mathbb{1}\{\vartheta_i \geqslant 0\}. \tag{14}$$

To sample from the hyper-posterior $p(\boldsymbol{\vartheta}|\mathscr{T}) = p(\boldsymbol{l} \mid \boldsymbol{\Theta}, \boldsymbol{\vartheta})p(\boldsymbol{\vartheta})/p(\mathscr{T})$, note that the GP prior for $\mathcal{L}_\star(\boldsymbol{\theta})$ and Gaussian likelihood of $(\boldsymbol{L}|\mathcal{L}_\star(\boldsymbol{\theta}), \boldsymbol{\Theta}, \boldsymbol{\vartheta})$ imposed by GP regression results in a marginal likelihood of $\boldsymbol{L}|\boldsymbol{\Theta}, \boldsymbol{\vartheta} \sim \mathcal{N}(\mu\mathbf{1}, \kappa(\boldsymbol{\Theta}, \boldsymbol{\Theta} \mid \boldsymbol{\vartheta}) + \sigma_0^2\mathbf{I})$. Thus slice sampling with the un-normalised density $p(\boldsymbol{l} \mid \boldsymbol{\Theta}, \boldsymbol{\vartheta})p(\boldsymbol{\vartheta})$ can be done with $p(\boldsymbol{l} \mid \boldsymbol{\Theta}, \boldsymbol{\vartheta})$ being a Gaussian likelihood and $p(\boldsymbol{\vartheta})$ is as in (14).

To decide which $\boldsymbol{\theta}$ to evaluate next using the test loss $\mathcal{L}$, the integrated acquisition $a(\boldsymbol{\theta} \mid \mathscr{T})$ is maximised via random search — random points are generated from $\boldsymbol{\theta} \in \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times [0, 1] \equiv \mathscr{O}$ uniformly at random and $a(\boldsymbol{\theta} \mid \mathscr{T})$ is evaluated at each point, seeing which admits the largest value.

This process of maximising $a(\boldsymbol{\theta} \mid \mathscr{T})$ and evaluating $\mathcal{L}$ is repeated until the budget is exhausted; which in this case is until a set amount of iterations have been completed.

5 Main Findings

Pytorch was used to implement the Neural Networks in Python due to its streamline support for GPU processing and similarity to numpy array manipulations. Some small tricks to speed up computation were to solve linear systems $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ rather than explicitly computing inverses $\mathbf{A}^{-1}$ and utilising `@jit` from the `numba` library for

[7] This is a good choice of prior because it is non/weakly-informative; giving higher likelihood to potentially unsuspecting values of the hyperparameters, and the values of the hyperparameters either are required to be non-negative or get squared anyway - becoming positive.

[8] Recall that the dimension of $\boldsymbol{\vartheta}$ should be 6 because in the Matérn 5/3 covariance function from (8): one covariance hyperparameter is needed for each dimension of $\boldsymbol{\theta}$, one more is needed for the 'ϑ_0' and another for the observation noise 'σ_0^2' parameter.

efficient numpy operations [4]. Neural Networks were trained for a minimum of 1000 epochs, maximum of 5000 epochs or until consecutive training losses were within tolerance 10^{-4} using batch size equal to the sample size. ADADELTA [11] was used for optimisation with mean squared error loss. Test Loss was measured by euclidean distance between predictions and test data. Activation functions on hidden layers were ReLU and linear activation on output layers. Poor performing networks were excluded from the ensemble on the basis of variability in predictions; such networks arise when too many weights are dropped from a network and it suffers from under-fitting. Integrated acquisition using 5000 posterior samples from slice sampling with adaptive width parameter (See appendix A.4.2) was optimised by random search. Mean function for the GP model of $\mathcal{L}_\star(\boldsymbol{\theta})$ was as in (7) for increased exploration behaviour of the Bayesian Optimisation.

5.1 Experiment 1

The training set $\tau = \{(x_1, y_1), \ldots, (\boldsymbol{x}_{150}, y_{150})\}$ was generated via $x_i \sim \mathcal{U}(-14, 11), y_i = f(x_i) + \varepsilon_i, \varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, 5^2)$ with

$$f(\boldsymbol{x}) = \begin{cases} x, & x < -10 \\ -x, & -10 \leqslant x < -8 \\ x, & -8 \leqslant x < -6 \\ (x+5)^2 - 7, & -6 \leqslant x < 2 \\ x^2 \sin(2x), & else \end{cases}$$

Integrated acquisition was optimised via random search with 2000 samples of $L \sim \mathcal{U}(\{1, \ldots, 5\}), D \sim \mathcal{U}(\{1, \ldots, 15\}), p \sim \mathcal{U}(0, 1), N \sim \mathcal{U}(\{1, \ldots, 30\})$.

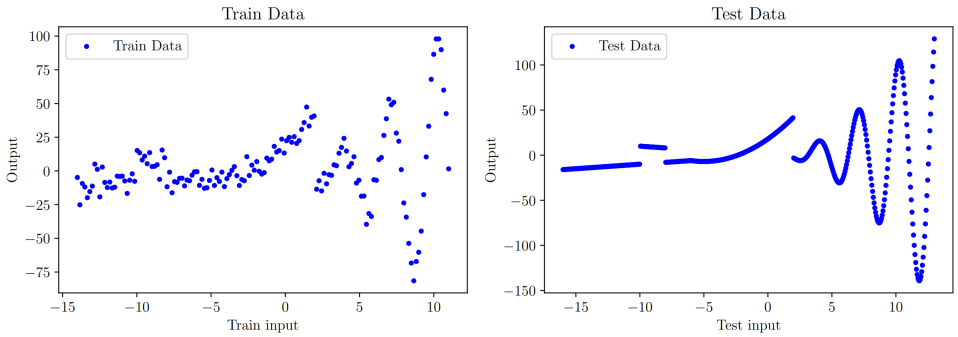

Figure 3: Experiment 1: (Left) train data, (Right) test data.

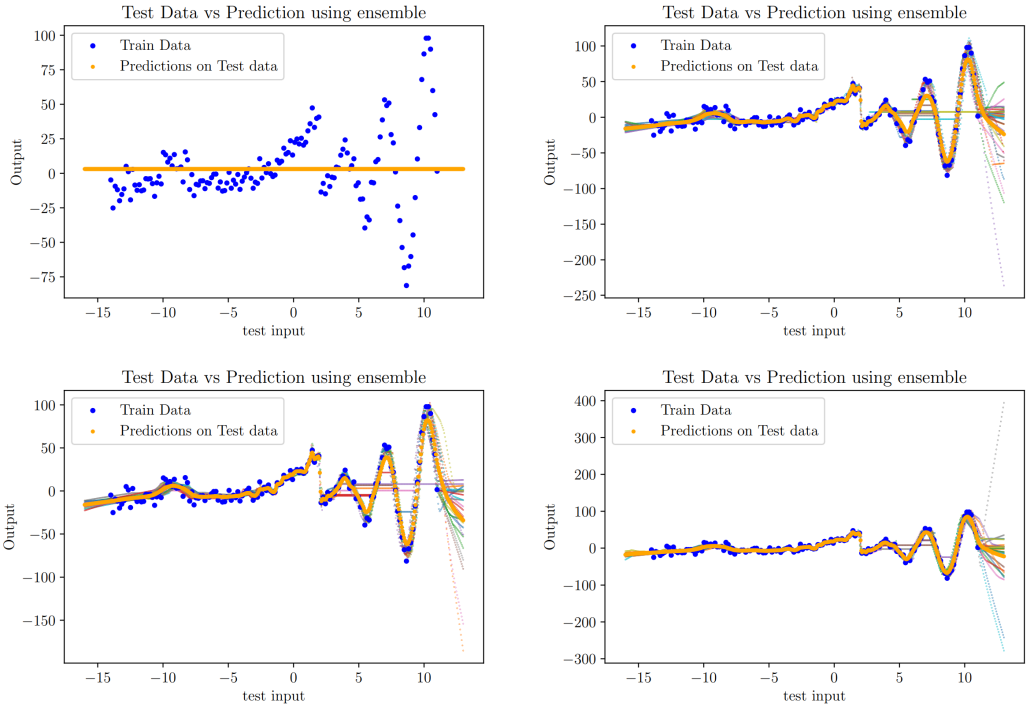

Figure 4: Experiment 1: Fit to train data on iterations 0,1,2,13. Smallest test loss iteration 13 with $(L, D, N, p) = (5, 15, 23, 0.949)$.

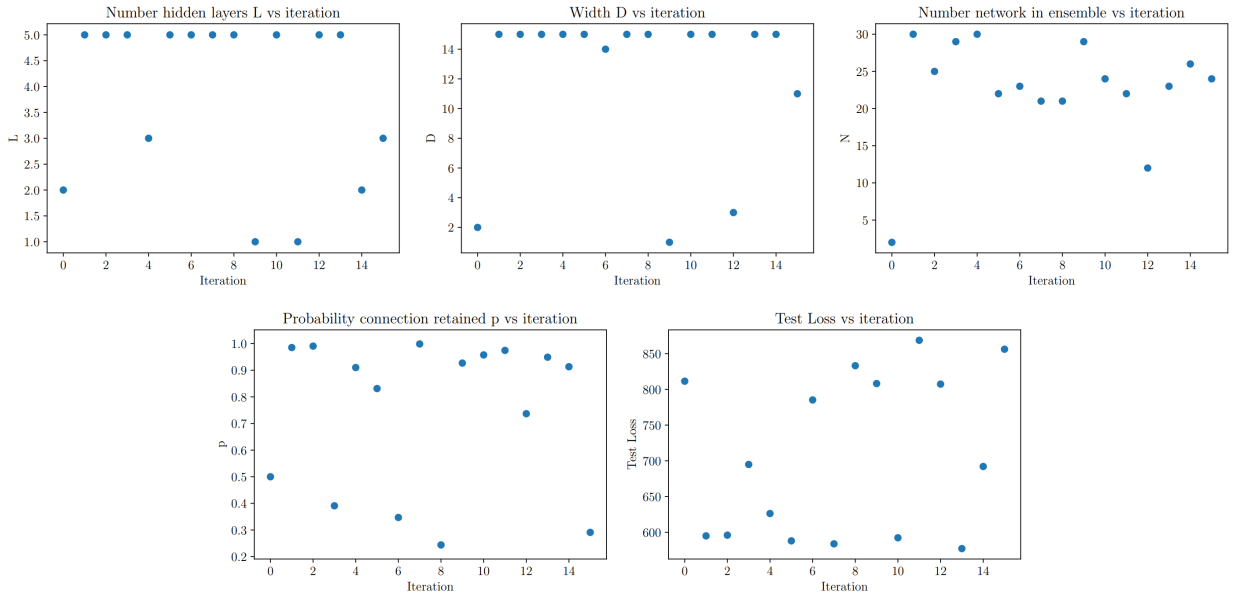

Figure 5: Experiment 1: Exploration behaviour of Bayesian Optimisation.

5.2 Experiment 2

The training set $\tau = \{(x_1, y_1), \ldots, (x_{100}, y_{100})\}$ was generated via $x_i \sim \mathcal{U}(0, 1), y_i \mid \boldsymbol{x}_i \sim \mathcal{N}(10 - 140x_i + 400x_i^2 - 250x_i^3, 5^2)$ (from [4]). The integrated acquisition was approximated using 5000 hyper-posterior samples of adaptive width slice sampling and optimised via random search with 1000 samples of $L \sim \mathcal{U}(\{1, \ldots, 10\}), D \sim$

$\mathcal{U}(\{1,\ldots,10\}), N \sim \mathcal{U}(\{1,\ldots,30\}), p \sim \mathcal{U}(0,1)$. Neural Networks were restricted to a maximum of 2000 training epochs to acts as a form of regularisation.

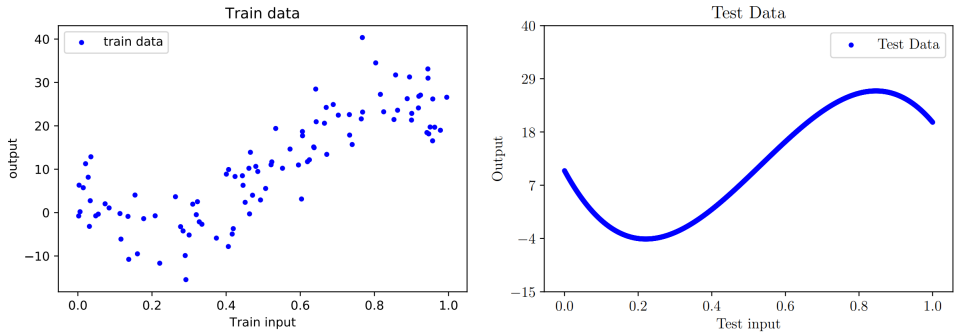

Figure 6: Experiment 2: Polyreg dataset from [4].

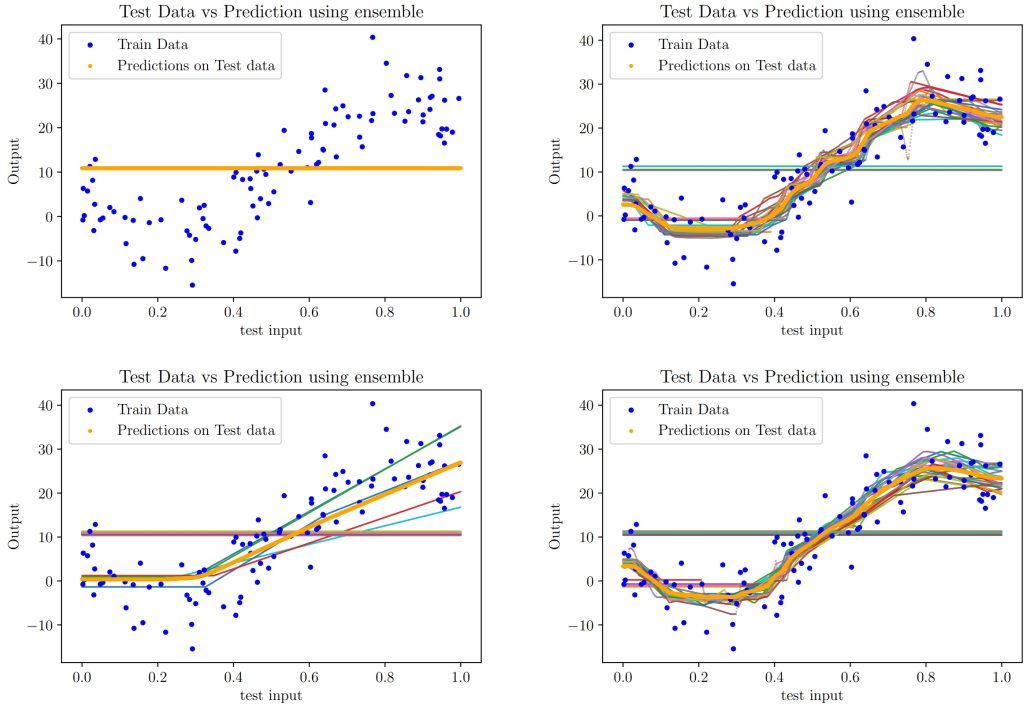

Figure 7: Experiment 2: Fit to train data on iterations 0,1,2,3. Iteration 3 — $(L, D, N, p) = (7, 7, 30, 0.73)$.

6 Conclusions and Further Research

The effective use of Bayesian optimisation for optimising hyperparameters associated with an ensemble of Neural Networks was discussed. Choosing effective mean and covariance functions for the Gaussian process regression sub-problem of Bayesian optimisation along with the use of an integrated acquisition function with non informative/weakly informative prior on GP hyperparameters was discussed as well as ways of doing linear algebra computations with numpy and pytorch more efficiently.

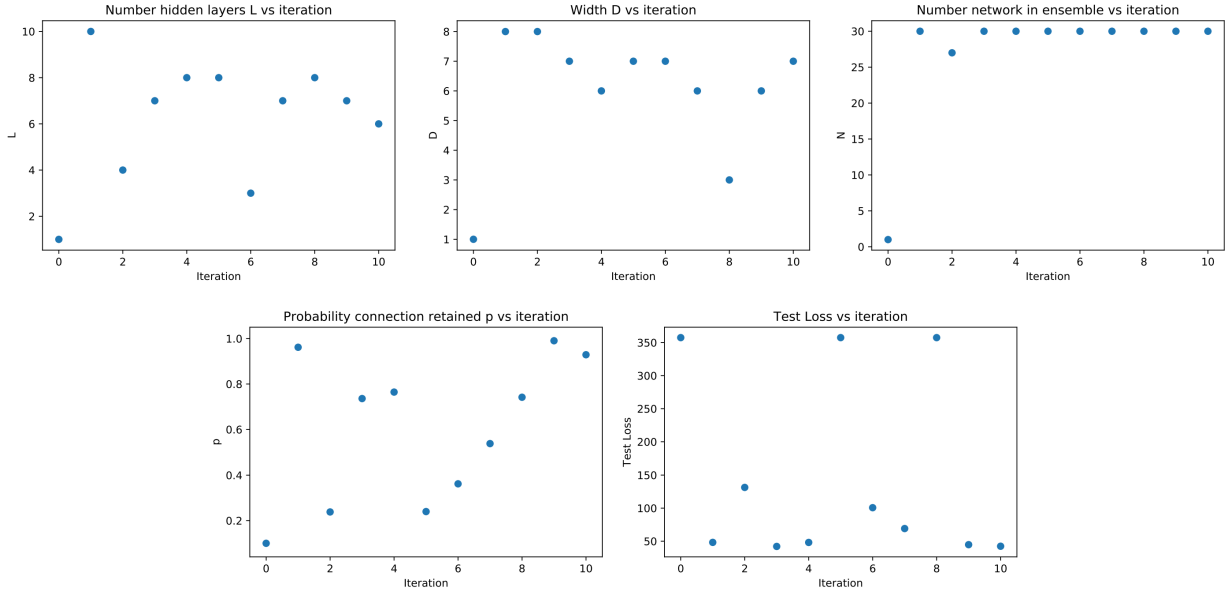

Figure 8: Experiment 2: Exploration behaviour of Bayesian Optimisation.

A slight modification onto the results could be to make each member of the ensemble learn a certain subset of the training data rather than getting each to learn the entire training set. This could be much better in cases where you are trying to learn a discontinuous function as each continuous part of the function can be learnt by a neural network, then combining the output of the Neural Networks piecewise acts as a solution to the problem (confer regression using tree methods). The results in this report can easily be generalised to other related problems such as classification tasks being solved using ensembles of convolutional Neural Networks (CNNs) [20] instead of MLPs. Effective regularisation for Bayesian optimisation of Neural Network ensemble designs can also be explored so to penalise the use of overly large networks that require too many resources to train sufficiently. Finally, developing methods for exploiting high performance computers and parallel computing is an interesting direction since the use of ensembles of Neural Networks is very computationally intensive.

7 Acknowledgements

I would like to give special thanks to Dirk Kroese for supervising me throughout the duration of the project, being available for frequent consultation sessions and allowing me to make an early start. I would also like to thank the University of Queensland for facilitating the completion of the project as well as AMSI for the funding and great opportunity to be involved in a Vacation Research Scholarship.

A Appendix

A.1 Notation

x	scalar value/outcome of random variable taking on scalar values
$\boldsymbol{x}$	vector value/outcome of random variable taking on vector values
X	random scalar
$\boldsymbol{X}$	random vector
$\mathbf{X}$	matrix
$\mathcal{X}$	set
$\widehat{x}$	approximation
x^*	optimal
$\bar{x}$	arithmetic mean
$\mathbb{P}$	probability measure
$\mathbb{E}$	expectation
$\mathcal{N}$	normal/Gaussian distribution
$\mathcal{U}$	uniform distribution

Any other notation introduced throughout the paper will be clarified on a case-by-case basis.

A.2 Positive Definite Functions and Gram Matrices

Let $\mathcal{X} \subseteq \mathbb{R}^d$. A function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is said to be a *positive semidefinite function* if for every choice of $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$ it holds that

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \alpha_j \geqslant 0$$

Let $\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_n} \in \mathcal{X}$ and $\boldsymbol{x}_{j_1}, \ldots, \boldsymbol{x}_{j_m} \in \mathcal{X}$. Create two *design* matrices: $\mathbf{X}_i = (\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_n})^\mathsf{T} \in \mathbb{R}^{n \times d}, \mathbf{X}_j = (\boldsymbol{x}_{j_1}, \ldots, \boldsymbol{x}_{j_m})^\mathsf{T} \in \mathbb{R}^{m \times d}$ by stacking row vectors on top of one another. Then a *Gram matrix* $\mathbf{K}$ can be created from these two design matrices as

$$\mathbf{K} := \kappa(\mathbf{X}_i, \mathbf{X}_j) := \begin{pmatrix} \kappa(\boldsymbol{x}_{i_1}, \boldsymbol{x}_{j_1}) & \cdots & \kappa(\boldsymbol{x}_{i_1}, \boldsymbol{x}_{j_m}) \\ \vdots & & \vdots \\ \kappa(\boldsymbol{x}_{i_n}, \boldsymbol{x}_{j_1}) & \cdots & \kappa(\boldsymbol{x}_{i_n}, \boldsymbol{x}_{j_m}) \end{pmatrix} \in \mathbb{R}^{n \times m}$$

An equivalent definition of a positive semidefinite function is that every square Gram matrix $\mathbf{K}$ associated with the function κ is a positive semidefinite matrix, that is, for any $\boldsymbol{\alpha} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, $\boldsymbol{\alpha}^\mathsf{T} \mathbf{K} \boldsymbol{\alpha} \geqslant 0$.

A.3 Loewner Partial-Order

A symmetric matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$ is called positive semi-definite if for any vector $\boldsymbol{v} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$, $\boldsymbol{v}^\mathsf{T}\mathbf{A}\boldsymbol{v} \geqslant 0$. We write $\mathbf{A} \succeq 0$ to denote that $\mathbf{A}$ is positive semi-definite. Similarly for positive-definiteness $\boldsymbol{v}^\mathsf{T}\mathbf{A}\boldsymbol{v} > 0$ we write $\mathbf{A} \succ 0$. Let $\mathbf{B} \in \mathbb{R}^{d\times d}$, then the Loewner Partial-Order relation is defined by

$$\mathbf{A} \succ \mathbf{B} \iff \mathbf{A} - \mathbf{B} \succ 0,$$
$$\mathbf{A} \succeq \mathbf{B} \iff \mathbf{A} - \mathbf{B} \succeq 0.$$

A.4 MCMC Sampling

The basic idea of MCMC sampling is to generate samples from an arbitrary distribution referred to as the target distribution. This is achieved by running a Markov chain with a limiting distribution equal to the target distribution for a long time such that eventually the iterates in the Markov chain are approximately distributed according to the limiting/target distribution. The random variables form an approximate and dependent sample from the target distribution (the dependence structure is the same as a Markov chain, the distribution of one state depends only on the previous state). We discuss two MCMC samplers: the Metropolis-Hastings sampler and the Slice sampler.

A.4.1 Metropolis-Hastings (MH) sampler

The MH sampler is perhaps the most well-known MCMC sampler as it is a prototypical example of such a method. Suppose you aim to generate a random sample from a distribution with pdf

$$f(\boldsymbol{x}) = \begin{cases} \frac{f^*(\boldsymbol{x})}{Z}, & \boldsymbol{x} \in \mathcal{X} \\ 0, & else \end{cases} \tag{15}$$

where $Z = \int_{\mathcal{X}} f^*(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$ is a normalisation constant which is unknown and typically computationally expensive to approximate well. One area which densities of this type arise is in Bayesian statistics when dealing with a posterior distribution $f(\theta|\tau) = f(\tau|\theta)f(\theta)/f(\tau)$; here, $f(\tau) = Z$. MH avoids having to compute Z explicitly by only requiring the computation of a ratio of the form $f(\boldsymbol{x}_1)/f(\boldsymbol{x}_2)$ which has no dependence on Z. The central idea in MH is the *proposal density* $q(\boldsymbol{y}|\boldsymbol{x})$ which governs how the space $\mathcal{X}$ is explored. For example $\mathcal{X} = \mathbb{R}, \boldsymbol{y}|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}, \mathbf{I}), q(\boldsymbol{y}|\boldsymbol{x}) = \phi(\boldsymbol{y} - \boldsymbol{x})$. The *acceptance probability*

$$\alpha(\boldsymbol{x}, \boldsymbol{y}) = \min\left\{\frac{f(\boldsymbol{y})q(\boldsymbol{x}|\boldsymbol{y})}{f(\boldsymbol{x})q(\boldsymbol{y}|\boldsymbol{x})}, 1\right\}$$

is the probability of accepting a proposed state $\boldsymbol{y}$ from the current state $\boldsymbol{x}$. The MH Markov chain is formed by choosing an initial state $\boldsymbol{x}_0$, proposing a new state $\boldsymbol{Y} \sim q(\boldsymbol{y}|\boldsymbol{x}_0)$, then accepting this new state with probability $\alpha(\boldsymbol{x}_0, \boldsymbol{Y})$ or rejecting it with probability $1 - \alpha(\boldsymbol{x}_0, \boldsymbol{Y})$. So, either the new state of the Markov chain remains

the same as the old state, or a different state is accepted. This process is repeated for as long as the user is willing to run the chain for.

A.4.2 Slice Sampling Algorithm

Algorithm 1: Slice Sampling with Adaptive Width

Inputs: Initial point $\boldsymbol{\theta}_0 = \left(\theta_{0,1}, \ldots, \theta_{0,dim(\boldsymbol{\theta})}\right)$, width parameter W, target density $p(\boldsymbol{x}) = p^*(\boldsymbol{x})/Z$, number samples required M,max loop time T;

for $i = 0, \ldots, M-1$ **do**
  for $j = 0, \ldots, \dim(\boldsymbol{\theta})$ **do**
    $H \sim \mathcal{U}(0, p^*(\theta_{i,j}))$;
    $C \sim \mathcal{U}(0, W)$;
    $L = \theta_{i,j} - C$;
    $R = \theta_{i,j} + (W - C)$;
    $t = \text{time}$;
    while $p^*(L) > H$ **do**
      $L \leftarrow L - W$;
      if $\text{time} - t > T$ **then**
        $W \leftarrow 2W$;
      else
        $W \leftarrow W$;
      end
    end
    $t = \text{time}$;
    while $p^*(R) > H$ **do**
      $R \leftarrow R + W$;
      if $\text{time} - t > T$ **then**
        $W \leftarrow 2W$;
      else
        $W \leftarrow W$;
      end
    end
    $\theta_{i+1,j} \sim \mathcal{U}(L, R)$;
    while $p^*(\theta_{i+1,j}) < H$ **do**
      if $\theta_{i+1,j} > \theta_{i,j}$ **then**
        $R = \theta_{i+1,j}$;
        $\theta_{i+1,j} \sim \mathcal{U}(L, R)$;
      else
        $L = \theta_{i+1,j}$;
        $\theta_{i+1,j} \sim \mathcal{U}(L, R)$;
      end
    end
  end
end

References

[1] Bengio, Y., 2019, 'From System 1 Deep Learning to System 2 Deep Learning', *NeurIPS' 2019 Keynote*. Available at: `https://nips.cc/Conferences/2019/ScheduleMultitrack?event=15488`.

[2] Srivastava, N., et al, 2014, 'Dropout: A simple way to prevent Neural Networks from overfitting', *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958.

[3] Mockus, J., Tiesis, V., Zilinskas, A., 1978, 'The application of Bayesian methods for seeking the extremum', *Towards Global Optimization*. vol. 2, pp. 117–129.

[4] Kroese, D. P., Botev, Z.I., Taimre, T., Vaisman, R., 2019, *Data Science and Machine Learning: Mathematical and Statistical Methods*. Chapman and Hall/CRC.

[5] Cybenko, G., 1989, 'Approximation by superpositions of a sigmoidal function', *Math. Control Signal Systems*, vol. 2, pp. 303–314.

[6] Hanin, B., Sellke, M., 2017, 'Approximating Continuous Functions by ReLU Nets of Minimal Width', *arXiv:1710.11278*.

[7] Glorot,X., Bordes, A., Bengio, Y., 2011, 'Deep Sparse Rectifier Neural Networks', *Proceedings of Machine Learning Research*, vol. 15, pp. 315–323.

[8] LeCun, Y., Bengio, Y., Hinton, G., 2015, 'Deep Learning', *Nature*. vol. 521, no. 7553, pp. 436–444.

[9] Ramachandran, P., Barret, Z., Quoc, V. L., 2017, 'Searching for Activation Functions', *arXiv:1710.05941*.

[10] Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L., 2017, 'The Expressive Power of Neural Networks: A View from the Width', *arXiv:1709.02540*.

[11] Zeiler, M. D., 2012, 'ADADELTA: An Adaptive Learning Rate Method', *arXiv:1212.5701*.

[12] Kingma, D. P., Ba, J., 2017, 'Adam: A Method for Stochastic Optimization', *arXiv:1412.6980*.

[13] LeCun, Y., Bottou, L., Orr, G. B., Müller, K. R., 1998, 'Efficient BackProp', *Neural Networks: tricks of the trade, Springer*, vol. 1, pp. 9–50.

[14] Rasmussen, C. E., Williams, C., 2006, *Gaussian Processes for Machine Learning*. The MIT Press.

[15] Snoek, J., Larochelle, H., Adams, R. P., 2012, 'Practical Bayesian Optimisation of Machine Learning Algorithms', *Advances in Neural Information Processing Systems*, vol. 4, pp. 2951–2959.

[16] Kroese, D. P., Taimre, T., Botev, Z. I., 2011, *Handbook of Monte Carlo Methods*, John Wiley & Sons, New York.

[17] Neal, R., 2003, 'Slice Sampling' *The Annals of Statistics*. vol. 31, no. 3, pp. 705–767.

[18] Foerster, J., 2014, 'Lecture 13: Approximating Probability Distributions (III): Monte Carlo Methods (II): Slice Sampling', Available at: https://www.youtube.com/watch?v=Qr6tg9oLGTA&t=5290s.

[19] Gelman, A., 2006, 'Prior distributions for variance parameters in hierarchical models', *Bayesian Analysis*. vol. 1, no. 3, pp. 515–533.

[20] LeCun, Y., et al, 1989, 'Backpropogation Applied to Handwritten Zip Code Recognition', *Neural Computation*, vol. 1, pp. 541–551.