Automatic Tuning of Deep Neural Networks with Applications

Submitted by
Yin Yumeng

Supervisor:
Prof. Chua Tat-Seng

In partial fulfilment of the
requirements for the Degree of
Bachelor of Engineering (Computer Engineering)
National University of Singapore

B.Eng. Dissertation

Automatic Tuning Of Deep Neural Networks With Applications

By

Yin Yumeng

National University of Singapore

2015/2016

Project ID: H009620
Project Supervisor: Prof. Chua Tat-Seng

Deliverables:
Report: 1 Volume
Source Code: https://github.com/yinyumeng/HyperParameterTuning

Abstract

Hyperparameter optimization is proven to be crucial for the success of machine learning algorithms. A few recent researches successfully applied optimization techniques to automatically tune hyperparameters of learning algorithms, namely Bayesian Optimization and Gradient-based hyper-parameter optimization. However, despite the success of these optimization methods, there exist certain challenges for these methods.

The selection of the type of kernel is proven to be crucial to the performance of Bayesian Optimization fitted with Gaussian Process. In this work we explore the Bayesian Optimization method with Bayesian Neural Network prior, which will be able to learn the kernel from scratch rather than requiring pre-setting of kernels, at the same time it performs competitively with the GP-based approaches within the same rounds of hyper-parameter optimization meta-iterations. As a result, it will take longer time for each hyperparameter tuning iteration.

What's more, we explore a hyperparameter server framework to perform distributed hyper-parameter optimization problems. Our preliminary experiments show that the hyperparameter server has the ability of performing competitively compared to the initial optimization strategy within the same rounds of hyper-parameter optimization meta-iterations.

Subject Descriptors:

Optimization

Artificial Intelligence

Keywords:

Hyperparameter tuning, Bayesian Optimization, Bayesian Neural Network, Gradient Based Method, Hyperparameter Server

Implementation language and libraries used:

Python, Keras, Spearmint, GpyOpt, autograd, SKLearn

Acknowledge

I would like to express my gratitude to my supervisor, Prof Chua Tat-Seng for patient guidance, enthusiastic encouragement and useful critique of my final year project. During the one year project, Prof Chua not only provided me with patient guidance and much technical advices of my project from a bigger perspective, he also gave me sincere help and encouragement during the journey.

I am hugely grateful to my evaluator, Prof Low Kian Hsiang for reviewing my final year project in the halfway of the journey, giving me valuable feedback and advice on the halfway of the final year project.

I would like to acknowledge and express special thank my mentor Mr. Fu Jie for providing steadfast guidance to me all the way throughout the final year project, for introducing me into various machine learning technologies, for giving me patient advice on the exploring of knowledges of all related topics and for always being around when I need help.

I would like to thank all people who supported me during the final year project, without which I could not carried on and complete the project.

List of Figures

Contents

Part I

Introduction

Deep neural networks have proven to constitute the state-of-art methods for many classification problems [6]. However, the success of deep neural networks algorithms rely heavily on the proper choices of high-level hyper-parameters [6]. The hyperparameters are normally not adapted in the learning process of elementary parameters [6].

Normally, the hyper-parameters are tuned manually and the tuning process always requires experience so that they can find proper set of hyper-parameter within only a few trails [32]. Presently, the development of computation hardware devices like GPU and computer clusters make it possible for researchers to run more trails. What's more, automated hyperparameter optimization methods have recently been shown to be able to find hyperparameter sets competitive with those found by human experts [8].

There were several methods to optimize the hyper-parameters of deep learning algorithms. In this work we mainly focus on two methods, namely Bayesian Optimization and Gradient Based optimization method.

Bayesian optimization is a global optimization methodology of noisy black-box functions [34], which offers a principled approach to model uncertainty and allows exploration and exploitation to be balanced. Recently Gaussian Process is the most commonly used model for Bayesian Optimization because of its flexibility and simplicity [34].

On the other hand, there is a newly invented gradient-based optimization method that performs optimizing work by making use of automatic differentiation to calculate gradients of error value with respect to hyperparameters [29]. The method is competitively with Bayesian Optimization because it is able to optimize thousands of hyper-parameters.

Despite the success of these optimization methods, there exist certain

challenges for these methods.

- For Bayesian Optimization fitted with Gaussian Process, according to Snoek's work in 2015, some certain choices such as the treatment of its own hyper-parameters and the type of kernel, which accounts for the nature of the GP, will play a crucial role in order to obtain a good optimizer that can achieve expert-level performance [34]. This can be regarded as another hyper-parameter that requires tuning for different hyper-parameter tuning tasks.

- Due to the growth of data and increase of model's complexity, no single machine can solve large scale machine learning problems sufficiently and repidly [26]. Nowadays distributed optimization and inference is becoming more and more popular in solving these large scale machine learning problems [26]. This problem also exists for hyper-parameter optimization fields.

In this work, we explore the extension to these two hyper-parameter optimization strategies.

The main contribution of the project is listed as following.

- We explore the Bayesian Optimization with Bayesian Neural Network prior. Bayesian Neural Network is selected because of its ability to predict the full posterior distribution of weights in the networks, integrating uncertainty at the same time[33]. Whats' more, the hidden layer of bayesian neural network will function equivalent to Gaussian Process with a certain covariance kernel [33]. In return, it will take longer time to get the similar results like bayesian optimization fitted with Gaussian Process.

- We propose a hyperparameter server framework to perform distributed hyper-parameter optimization problems. The idea is initially moti-

vated by Mr. MuLi's work [26] of parameter server which realizes the distributed machine learning methods.

Part II

Background and Related Work

1 Deep Neural Network

Originating from attempts to find mathematical representations of information processing in biological systems [38], artificial neural networks have aroused so much interests in recent years [35]. Massive neurons which represent simple computing cells are employed in neural networks to mimic the model that the brain uses to perform a specific task or function [15]. Computations are performed through a learning process [15] to modify the *weights* of the network, which will eventually make the neural network adapt to the inputs [15].

1.1 Learning paradigms and Network Architecture

There are three major learning paradigms, unsupervised learning, supervised learning and reinforcement learning [1].

In this work we focus on the supervised learning. For supervised learning, training datasets are given in the pair of features and corresponding targets $(x, t),\ x \in \mathcal{X}^{(\text{train})}, t \in T$ [15]. The training process is to find the function $f : \mathcal{X}^{(\text{train})} \to T$, which is usually done by optimizing a training criterion with respect to a set of *parameters* θ (*weights and bias*). The cost function here will related to the mismatch between the predicted targets of validation datasets and correct targets.

1.1.1 Multi-Layer Perception

Multi-layer perception(MLP) is proven to be of great practical value from neural network architectures [9]. A MLP network is a feedforward artificial

neural network includes multiple layers of neurons and each layer will be fully connected to the next one [31]. It use the supervised learning technology, back-propagation algorithm, for training [31]. In this work we focus on the MLP to multiple binary classification problems.

The MLP is a modification of the linear model for regression and classification. It can be represented as a series of functional transforms [9]. Considering a MLP with D input variables $x_1, ..., x_D$ and M = $NeuronCountOfNextLayer$ combinations of inputs are constructed, the activation units of first layer neurons have the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_j^{(1)} \tag{1}$$

where $j = 1, ..., M$, $i = 1, ..., D$, the superscript (1) represent the corresponding layer of the parameter [9]. The parameter $w_{ji}^{(1)}$ is referred to as *weight* for the $j-th$ *activation* unit of $i - th$ input of the $1 - st$ layer. The parameter $w_j^{(1)}$ is referred to as *biases* value for the $1 - st$ layer.

Each of the activation units will be transformed first using an nonlinear differentiable $activationfunction$ $h(\cdot)$ as [9]

$$z_j = h(a_j) \tag{2}$$

Normally functions $h(\cdot)$ are sigmoidal functions like *tanh* function or *logistic sigmoid* function. These z will be transformed again to get the next layer's activation units as

$$a_j = \sum_{i=1}^{M} w_{kj}^{(2)} Z_j + w_k^{(2)} \tag{3}$$

where $k = 1, ..., K$ is the total number of the second layer [9]. When the MLP has two layer only, the k is the number of outputs and a_j represent the output activation units. The activation units will be transformed using activation function to get the final output. When the MLP has more than

two layers, the process (2) and (3) will be performed again.

The output units of the MLP should be transformed using appropriate activation function according to the assumed distribution of targets, which usually different from *activation functions* using in previous layers [9]. For example, standard regression problems use the activation function $y_k = a_k$. For multiple binary classification problems, common used activation functions can be $Rectified\ LinearUnit$, *tangent*, *sigmoid*, *maxout* and $softmax$. In this work we use a logistic sigmoid function as example [9]

$$y_k = \frac{1}{1 + exp(-a_k)} \tag{4}$$

In this case, the final output of two layer MLP take the following form after combining all stages

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma(\sum_{i=1}^{M} w_{kj}^{(2)} h(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_j^{(1)}) + w_k^{(2)}) \tag{5}$$

where $(\mathbf{x}, \mathbf{w})$ represent the grouped *weights* and *bias*. From the formula(a $forward\ propagation$), we can conclude that the mapping of input $\{x_i\}$ to target $\{y_k\}$ of a MLP is controlled by vector of *weights* $\mathbf{w}$.

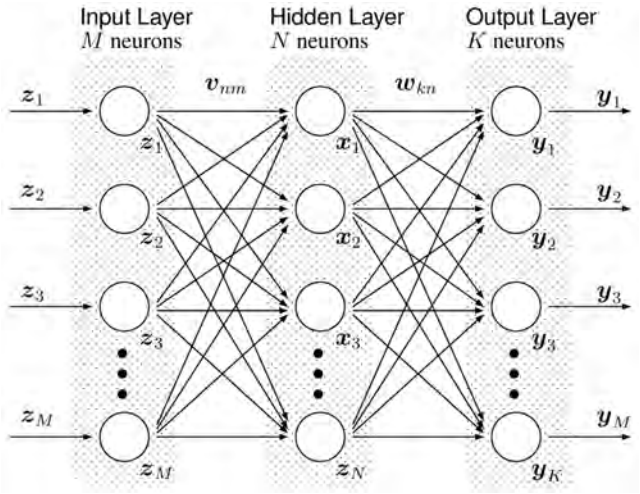

Figure 1: Two layer Multi-Layer Perception Network [36]

1.1.2 Convolutional Neural Network

Convolutional Neural Network(CNN), model of which will be invariant to certain transformation of the inputs, has been widely applied to image data [19]. Considering the task of recognizing handwritten digits, identity of the digit should be invariant even if pictures are translated, scaled or rotated. What's more, CNN should exhibit ability to identify pictures with more subtle transformations [19].

There is a key property of images that nearby pixels have stronger correlation than distant pixels [19]. Furthermore, images has property that local features useful in one region will be more likely to be useful in other images and regions.

These features are incorporated into CNN through three main strategy, namely local receptive fields, weight sharing and subsampling.

An example of CNN [19], consisting of alternating convolution and max-pooling networks, with MLP hidden layer with logistic regression in the upper- layers is shown in the following Figure.

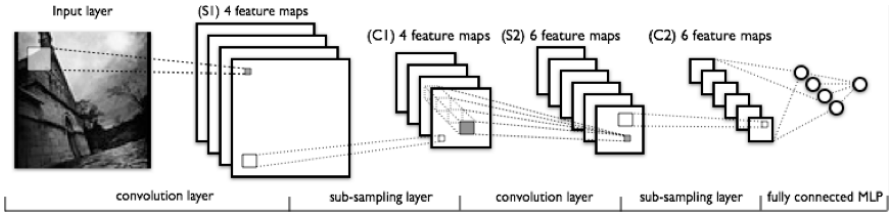

Figure 2: LeNet model of convolutional network [19].

1.2 Training using *backpropagation*

The process of training the neural network involve tuning of *weights* and *bias* of the network [7]. Two modes of training can be implemented, namely the incremental mode and batch mode. In incremental mode, gradient is computed to update weights after each data sample is trained [9]. In batch mode, weights will only be updated all inputs in the batch are applied to neural net. In this work, neural networks are implemented using batch mode because of its high speed and smaller errors compared to incremental mode [9].

The common training method used is *backpropagation*, in which multi-layer perception is training by gradient descent applied to a sum-of-squares error function [9]. For *backpropagation* the derivatives of the error function with respect to weights will be calculated propagated error value backwards to the previous layers [9].

1.2.1 Loss function

A loss function(error function) will be defined. In this work we focus on the MLP to multiple binary classification problems. Normally a common loss function will be the negative log-likelihood which is equivalent to maximizing

the likelihood L of the dataset D under the model parameterized by θ [7]

$$L(\theta = W, b, D) = \sum_{X}^{|D|} log(P(Y = y^{(i)}|x^{(i)}, W, b)) \tag{6}$$

$$l(\theta = \{W, b\}, D) = -L(\theta = \{W, b\}, D) \tag{7}$$

1.2.2 Gradient descent applied to error function

Gradient decent will be used to minimize the loss function, where weight was updated by small step in the direction of negative gradient [9]

$$w(\tau + 1) = w(\tau) - \eta \nabla E(w(\tau)) \tag{8}$$

2 Automatic Hyper-parameter Tuning Methods

Recently, some of the most simplest methods of automatic hyper-parameters tuning seems to be *grid search* and *random search* [16]. However, the drawback of these methods is that the number of trails will grows exponentially with the increase of number of hyper-parameters [5]. In this paper, we explored two automatic hyper-parameter tuning methods, namely Bayesian Optimization and Gradient Based Method of hyper-parameter tuning.

2.1 Bayesian Optimization

Bayesian optimization is a global optimization methodology of noisy black-box functions $f(x)$ [34]. When bayesian optimization is applied to the hyper-parameter tuning, the $f(x)$ refer to the *validation loss* value [32]. Bayesian optimization assumes a general prior over functions and then combine prior with observed hyperparameters and their function result to update the dis-

tribution of these hyperparameters over functions [34]. The methodology proceeds tuning by continuously picking new distribution point of hyperparameters to observe. The choosing strategy is in the manner that trades off exploration and exploitation, which refers to most uncertain point and best potential result point [34].

Bayesian Optimization function based on Bayes theorem' [17]:

$$P(A|D) \propto P(D|A)P(D). \tag{9}$$

There are two crucial bits in Bayesian Optimization (BO) procedure approach [23], the choice of prior function and the choice of acquisition function.

2.1.1 Prior probability measure on $f(\mathbf{x})$:

This function will capture the prior belief on the function $f(x)$ [9]. The probability of *posterior* is calculated as *likelihood* $\times$ *priorprobability* [17].

Maximum a posterior estimation is a mode of the posterior distribution [27].

$$\underset{\theta}{\operatorname{argmax}}\ P(\theta|\mathbf{x}) = \underset{\theta}{\operatorname{argmax}}\ P(\mathbf{x}|\theta)P(\theta) \tag{10}$$

- Gaussian Process

Its proven that the simplest type of prior over function is called Gaussian process [24]. The idea of Gaussian process modeling is to define a prior function for function before parameterizing the function [24]. Normally a Gaussian distribution is fully represented by the using of mean and covariance function [24]. In the same manner, Gaussian process is represented by the using of mean and covariance function [24]. Here, the mean is a function of x(which we will often take to be the zero function), and the covariance is a function which expresses the expected covariance between the value of the function y at the points $x$and x'.

In particular, it is proven that the Gaussian Process (GP) prior can work well with Bayesian Optimization [25][17]. GP is completely specified by its mean function, m and covariance function k [25]. Covariance function provide the method to measure how two variables change together and it accounts for the capacity of Gaussian Process to represent rich distribution of functions [16]. There are classes of covariance function that are normally used such as the squared exponential and Mat´ern covariance functions [16].

- Deep Neural Network

From Snoek's work in 2015, deep neural networks was used as an alternative to GPs to model the distribution over functions [34]. By adding a Bayesian linear regressor to the last hidden layer of the neural network, they use the point estimate for other parameters other than the output weights and the model result in an adaptive basis regression [34]. They proven in the work that performing adaptive basis function regression with neural network may perform competitively with the GP-based approaches [34]. In addition, with the increase of data, the approach will scale linearly, compared with the normal method which will scale cubically.

2.1.2 Acquisition Function

The acquisition function is used to help search for the next candidate point by balancing *exploration* and *exploitaion* trade-off [13]. By using rule of *exploration* we choose the next point with high surrogate variance to maximize the possibility of finding the new maximum point [13]. By using rule of *exploitaion* we choose the next point with high surrogate mean to maximize the cumulative reward [13].

These acquisition functions are calculated by the using of predictive mean function $\mu(x; xn, yn, \theta)$ and predictive variance function $\sigma 2(x; xn, yn, \theta)$ [32]. Some popular choices of acquisition functions are *maximum probability of improvement*, *expected imporvement*, and *upper confidence bound* [32].

Maximum Probability of improvement(MPI) [32]:

$$acqu_{MPI}(x;\{x_n,y_n\},\theta)=\Phi(\gamma(x)),where\gamma(x)=\frac{\mu(x;x_n,y_n,\theta)-f(x_{best})-\psi}{\sigma(x;x_n,y_n,\theta)} \tag{11}$$

Expected improvement(EI) [32]:

$$acqu_{EI}(x;x_n,y_n,\theta)=\sigma(x;x_n,y_n,\theta)(\gamma(x)\Phi(\gamma(x)))+N(\gamma(x);0,1) \tag{12}$$

Upper confidence bound(UCB) [32]:

$$acqu_{UCB}(x;x_n,y_n,\theta)=-\mu(x;x_n,y_n,\theta)+\psi\sigma(x;x_n,y_n,\theta) \tag{13}$$

ψ is a tunable parameter. The parameter can help the acquisition to be more flexible. In addition, in the case of the UCB, the parameter η is used to adjust the importance that we define for mean and variance of the model, which is also known as exploration/exploitation trade off [32].

2.2 Gradient-based Hyperparameter Optimization

2.2.1 Gradient-based methods for hyperparameter tuning

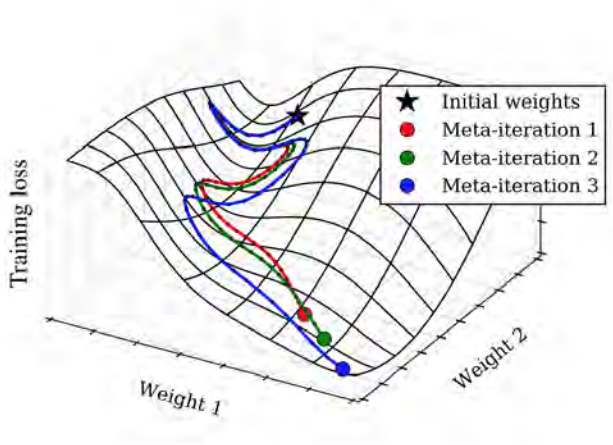

Figure 3: Hyperparameter optimization by gradient descent [29]

Tuning of hyper-parameters are not adapted by the learning algorithm itself [6] because the gradients of *validation loss* with respect to hyperparameters are hard to calculate. In the work of Maclaurin, they compute the exact gradient of $cross - validation\ loss$ with respect to all hyperparameters by applying chain rule and derivative backwards through the process of the training. The method can tune thousands of hyper-parameters including learning-rate, momentum schedule, weight initialization distributions, richly parameterized regularization schemes, and neural network architectures with the use of these gradients. The hyperparameter gradients is calculated by exactly reversing the dynamics of stochastic gradient descent with momentum [29].

The following Figure shows the chart of gradient-based hyper-parameter optimization. For the method, an entire training round of datasets will be run to optimize elementary parameters (weights 1 and 2, referring to learning rate and momentum schedule) [29]. After that, by using the reverse-mode

automatic differentiation, the gradient of loss value with respect to hyperparameters are calculated [29]. Hyperparameters will be updated in the direction of hypergradient then [29].

2.2.2 Reverse-mode Automatic differentiation (RMD)

The calculation of gradients will be implemented as the reverse-mode automatic differentiation package. AD systematically applies the chain rule to calculate gradients for elementary parameters for neural network model [11]. The RMD method is also known as back-propagation and it enable us to calculate the gradient of cross validation loss with respect to all hyperparameters [3]. This increases the computational burden by only a factor of two over evaluating the loss itself, regardless of the number of parameters.

Fundamental to RMD is the decomposition of differentials provided by the chain rule. For the $y = g(h(x)) = g(w)$ the chain rule gives $\frac{dy}{dx} = \frac{dy}{dw}\frac{dw}{dx}$ [37].

Figure 3 shows a Reverse-mode automatic differentiation example, with $y = f(x_1, x_2) = sin(x_1) + x_1 \cdot x_2$ at $(x_1, x_2) = (2, 3)$ and with seed $\bar{w_5} = 1$. Table 1 shows the operations needed to computer the value $f(x_1, x_2)$ as well as the operations of the corresponding backward automatic differentiation.

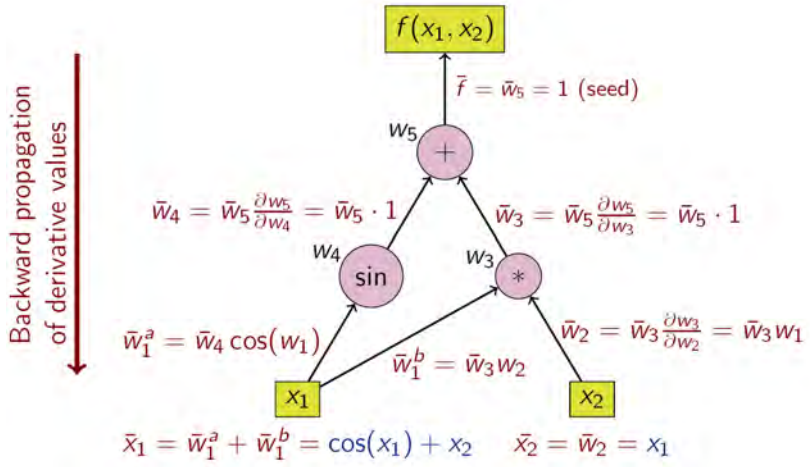

Figure 4: Reverse-mode Accumulation with computational graph [37]

Operations to compute value $f(x_1, x_2)$	
Input	$x_1 = 2$ $x_2 = 3$
Operations	$w_1 = x_1 = 2$ $w_2 = x_2 = 3$ $w_3 = w_1 \cdot w_2 = 6$ $w_4 = \sin(w_1) = \sin(2) = 0.034$ $w_5 = w_3 + w_4 = 6 + 0.034 = 6.034$
Output	$f(x_1, x_2) = w_5 = 6.034$

Operations to compute backward mode automatic differentiation	
Input	$\bar{w_5} = 1$ $x_1 = 2$ $x_2 = 3$
Operations	$\bar{w_4} = \bar{w_5} \cdot \frac{\partial w_5}{\partial w_4} = \bar{w_5} = 1$ $\bar{w_3} = \bar{w_5} \cdot \frac{\partial w_5}{\partial w_3} = \bar{w_5} = 1$
Output	$\bar{w_2} = \bar{w_3} \cdot \frac{\partial w_3}{\partial w_2} = \bar{w_3} \cdot w_1 = 1 \times 2 = 2$ $\bar{w_1} = \bar{w_3} \cdot \frac{\partial w_3}{\partial w_1} + \bar{w_4} \cdot \frac{\partial w_4}{\partial w_1} = \bar{w_3} \cdot w_2 + \bar{w_4} \cdot \cos(w_1) = 3.999$

Table 1: Operations of Reverse-mode automatic differentiation for Figure 3

Part III

Proposed Algorithms

In this work, I explored three methods to achieve the automatic tuning of hyperparameters, in which two of them showed significant effect in tuning hyperparameters and one showed insignificant effect.

The first method is the Bayesian Optimization which fitted with Bayesian Neural Network. This method is a strategy for global optimization of black-box functions [32]. Experiments are conducted to compare effects of Bayesian Optimization fitted using both Bayesian Neural Network as well as Gaussian Process. From the experiments conclusion can be made that this method will act successfully and will tune the hyperparameters as the similar result as the bayesian optimization fitted with gaussian process.

The second method is the hyperparameter server based on gradient-based optimization of deep neural networks. The idea is inspired by the parameter server.

The third method is to use of subclass data of datasets to tune the hyperparameters. In this work it will be shown that the method may not be efficient in tuning the hyperparameters. The work regarding this method may be found in the appendix.

3 Bayesian Optimization fitted with Bayesian Neural Network

3.1 Motivation

Gaussian Process has been a popular prior selected for Bayesian Optimization [25]. However, in fact, as the smoothness properties depend on the kernel function, the choice of kernel function thus becomes crucial for

GP [30]. According to Snoek's work in 2015, some certain choices such as the treatment of its own hyper-parameters and the type of kernel, which account for the nature of the GP, will play a crucial role in order to obtaining a good optimizer that can achieve expert-level performance [34]. This can be regarded as another hyper-parameter that requires tuning for different hyper-parameter tuning tasks. Normally people will choose some popular kernel functions suited to tasks at hand.

The exploration is inspired by the work of Jasper Snoek, in which the use of neural networks with a bayesian linear regressor to the last hidden layer is used as an alternative to GPs to model distribution over functions [34]. Snoek's work use point estimates for remaining parameters and perform an adaptive basis regression then. We explore the possibility of using Bayesian Neural Network as replacement of Gaussian Process prior.

The choice of bayesian neural network model is also because of the theoretical relationship between Gaussian processes and infinite Bayesian neural networks [24]. In 1996, Neal has already proven that the prior distribution over non-linear functions, which is implied by Bayesian Neural network, will fall in a class of probability distribution, which is known as Gaussian Process [28]. For bayesian optimization with bayesian neural network priors, there is no requirement of choosing kernel like Gaussian Process [33], the hidden layer of bayesian neural network will function equivalently to Gaussian Process with a certain covariance kernel. In return, it will take longer time to get the similar result like bayesian optimization fitted with Gaussian Process.

3.2 Bayesian Neural Network model

Within the bayesian neural network model, the uncertainty are expressed and measured by probabilities [14].

For bayesian neural, the prior distribution $p(w|\alpha)$ is firstly defined [14]. The prior function is usually assumed to be a Gaussian distribution with zero mean and inverse variance hyper-parameter α [14].

When applied to the dataset t, posterior probability over weight will be updated using Bayes' therem

$$p(w|t, \alpha, \beta) = \frac{p(t|w, \beta)p(w|\alpha)}{p(t|\alpha, \beta)}, \tag{14}$$

where $p(w|t, \alpha, \beta)$ is the likelihood function. The Bayesian Neural Network is based on the Bayesian inference and it use maximum likelihood to determine the network parameters(weights and biases). Regularized maximum likelihood can be interpreted as a MAP (maximum posterior) [27]

$$\underset{\theta}{\text{argmax}} \ P(\theta|\mathbf{x}) = \underset{\theta}{\text{argmax}} \ P(\mathbf{x}|\theta)P(\theta) \tag{15}$$

approach in which the regularizer can be viewed as the logarithm of a non-convex prior parameter distribution, corresponding to the multiple local minima of the error function [9].

The Bayesian Neural networks helps to get the full posterior distribution of the weights, integrating uncertainty at the same time [34].

3.3 Existing Package:

GPyOpt GPyOpt is a Bayesian Optimization framework written in python [23]. It is a tool for optimization of black-box functions using Gaussian process. It is based on GPy [22], which is a Gaussian Process framework written in python.

For GPyopt, most common acquisition functions are implemented such as Expected improvement(EI) and Maximum probability of improvement(MPI).

autograd Following David Duvenaud's work of black-box stochastic variational inference in a deep bayesian neural network [12]. The stochastic variational inference in a deep bayesian neural network is realized by the using of automatic differentiation.

3.4 Overall algorithm description

According to Snoek Jasper, there are two major choice that must be made for bayesian optimization. First is that one must choose a prior over functions that will express assumptions about the function being optimized. Traditionally we choose Gaussian Process because of its flexibility and tractability. Second we must choose an *acquisition function* to select the next best candidate of point to evaluate [32].

For the second point, we will focus on the EI criterion as *acquisition function* in this work , as it has been shown to be better-behaved than probability of improvement [32].

Same as other kinds of optimizations, in Bayesian Optimization we are trying to find the minimum of the target function $f(x)$, with the range of set X defined [32]. Bayesian optimization differentiates itself from other procedures in terms of the probabilistic model it constructed for $f(x)$ [32]. BO will integrating uncertainty when determining the next X to evaluate the function. The essential philosophy is to use all information available from previous evaluation of $f(x)$ rather than only local gradient [32].

With relatively few evaluations, the procedure of BO will be able to find the minimum of different non-convex function. On the other hand, more computation will be performed to determine the net point [32]. This feature will specially function better for $f(x)$ of which the evaluation requires training process of machine learning are expensive to perform.

Overall process of whole Bayesian Optimization using Bayesian Neural Network for prior There are four sub-process in the whole process of conducting Bayesian Optimization(BO).

The high level architecture of the process is shown in the Figure below.

- Initialization

First of all, some initial parameter settings for BO machine are required to be set by user. The parameters are the bounds of hyperparameters which need

to be tuned in the process, the layer_size of the Bayesian Neural Network, the evaluation function $f(x)$, the acquisition function that user wants to use in the process and number of initial random points user wants to generate. Some examples of these parameters are shown below.

One example of the hyperparameter *bounds* can be $[(0.001, 0.1), (0.6, 0.9), (0.2, 0.7), (0.2, 0.7)]$. The four bounds represent learning-rate, momentum, dropout1, dropout2 hyperparameters. Example of the $layersize$ is $[4, 20, 20, 1]$ when five hyperparameters will be tuned in the process. Detailed information of evaluation function $f(x)$ can be found in the part *evaluate the point* below. The available acquisition function in the current package GPyOpt are MPI, EI and UCB. We focus on EI in this work [23]. The number of initial random points will be used in the next step to give the BO machine starter points.

- Determine the next point to evaluate

When no point has been evaluated, the number of initial random points as well as the seed value will be used to generate sets of hyperparameters to give machine starter points.

When there exist points in the storage. The determination of the next point will be related to current point distribution and acquisition function.

- Evaluate the point: value of $f(x)$ on a specific point

The function $f(x)$ will be the result of $f(x)$ when synthetic datasets is used. For example, $f(x) = \cos(x)$ when input point is 1 dimension and graph of the point distribution is a cosine graph.

When BO is conducted on datasets such as MNIST and CIFAR-10, $f(x)$ will be the process which will return loss value after training the datasets with a specific point (a specific set of hyperparameters value). The function $f(x)$ represents the loss value distribution based on its hyperparameters values.

- Calculate posterior distribution of hyper-parameters

Express assumptions about the distribution function being optimized: predict the distribution of $f(x)$

As introduced before, Bayesian Neural Network is used to make assumptions about distribution of $f(x)$ instead of Gaussian Process. After every meta-iteration, a new point with its own output $f(x)$ values will be added into the bayesian neural network to update network weights. The Bayesian Neural Network is using the activation function of *tanh*. The prior distribution of hyperparameters we assume in the work is normal distribution.

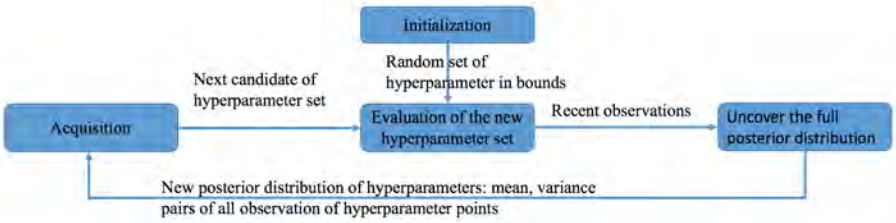

Figure 5: Overall procedure of Bayesian Optimization

4 Hyperparameter server which using the gradient-based optimization

4.1 Motivation: Parameter Server

Due to the growth of data and increase of model's complexity, no single machine can solve large scale machine learning problems sufficiently and rapidly [26]. Nowadays distributed optimization and inference is becoming more and more popular in solving these large scale machine learning problems [26].

The distribution of training process that optimizes elementary parameters of machine learning problem is introduced by Mr. MuLi's work [26]. For parameter server, realistic quantities of training data can range between 1TB

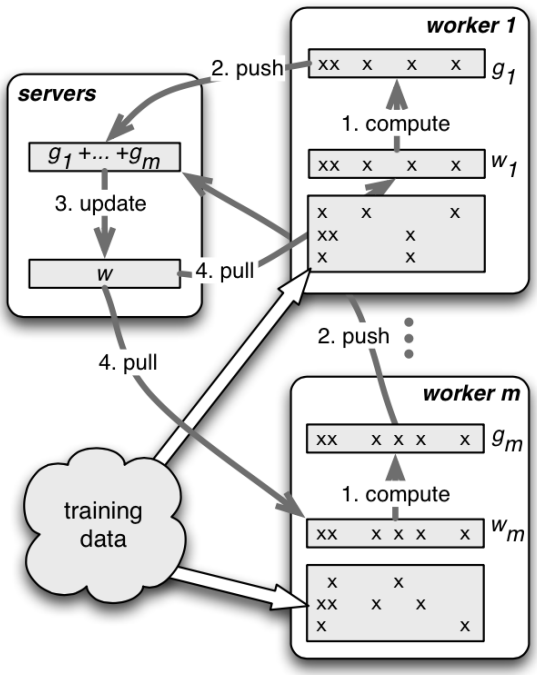

Figure 6: Parameter server mechanism [26]

and 1PB. These models are often shared globally by all worker nodes, which must frequently access the shared parameters as they perform computations to refine them [26].

The mechanism of parameter server is shown in the following Figure.

In a parameter server architecture, there are two group of nodes, namely server nodes and client nodes [26]. Only a part of parameters are stored in each server node. In addition, each corresponding client node will require a subset of these parameters in the process.

A server node will be in charge of maintaining a partition of shared global parameters. Server node will also communicate with each other to migrate and replicate parameters to achieve the scaling and reliability [26].

Each client only stores its own working set of weights w rather than all parameters. In the process of distributed data analysis, the reading and updating of shared parameters between different client nodes is ubiquitous. A client node will store a part of training data to compute gradients locally. Client node will not communicate with each other, they will directly communicate with server nodes to update gradients to server node and retrieve new shared parameters from server node [26].

4.2 Overall algorithm description

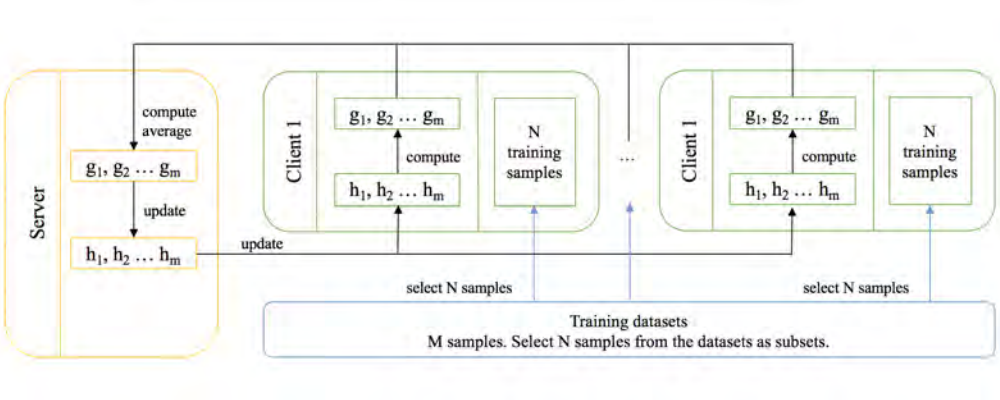

Figure 7: Hyperparameter server mechanism.

Motivated by the parameter server approach to distributed elementary parameter optimization [26]. We propose the *hyperparameter server* framework for distribution hyperparameter optimization using hypergradient. In parameter server, computational nodes are partitioned into clients and servers, and communication between nodes is asynchronous, and each node only caches the working set of w rather than all parameters [26]. However, for hyperparameter server there are several client node which handle the training on subsets of full datasets and computing of subsets hypergradients. There is only one server node, which maintains the computation of the global hy-

perparameters.

The initial weights will be set using initially by the using of L2 regularization value. In each meta-iteration of updating hyperparameters, clients will run training on different subsets using gradient based method. The gradient of hyperparameters will be sent back to server node after the meta-iteration. Server Node will update the hyperparameters using average hypergradient and send back the updated hyperparameters to client node to run the next round of training process. The whole mechanism is shown in Figure 23. The algorithm is shown in following Algorithm 2.

Part IV

Experiments and Evaluation

5 Bayesian Optimization fitted with Bayesian Neural Network

To demonstrate the effectiveness of applying Bayesian Neural Network to Bayesian Optimization, experiments were conducted of BO to both synthetic datasets and benchmark datasets. In this work I didn't strive for state-of-art performance on datasets. The performance of BO with BNN and GP to these datasets are compared.

In this experiments, Radial basis function kernel, which is a popular kernel function in various kerneled learning algorithms, is chosen for Gaussian Process. The BNN neural network is implemented of $layersize$=[i,20,20,1], where $i = input\ dimension$. The activation $tanh$ was used in this case to increase speed to converge than sigmoid and logistic function, and it also performs more accurately than the other two [2]. The acquisition function used for all experiments are Expected Improvement.

5.1 Experiments on synthetic datasets

Following some of the settings of Snoek's work [32], we evaluate the Bayesian Optimization with prior Bayesian Neural Network as well as Gaussian Process on several sets of synthetic datasets.

Two kinds of charts are plotted for synthetic datasets, namely the Figure of convergence and Figure of Acquisition(available for datasets with input dimension smaller than 3).

Three evaluations plots are available in the Figure of convergence.

- The distance between the last two observations.

- The value of f at the best location previous to each iteration.
- The predicted standard deviation of each new location.

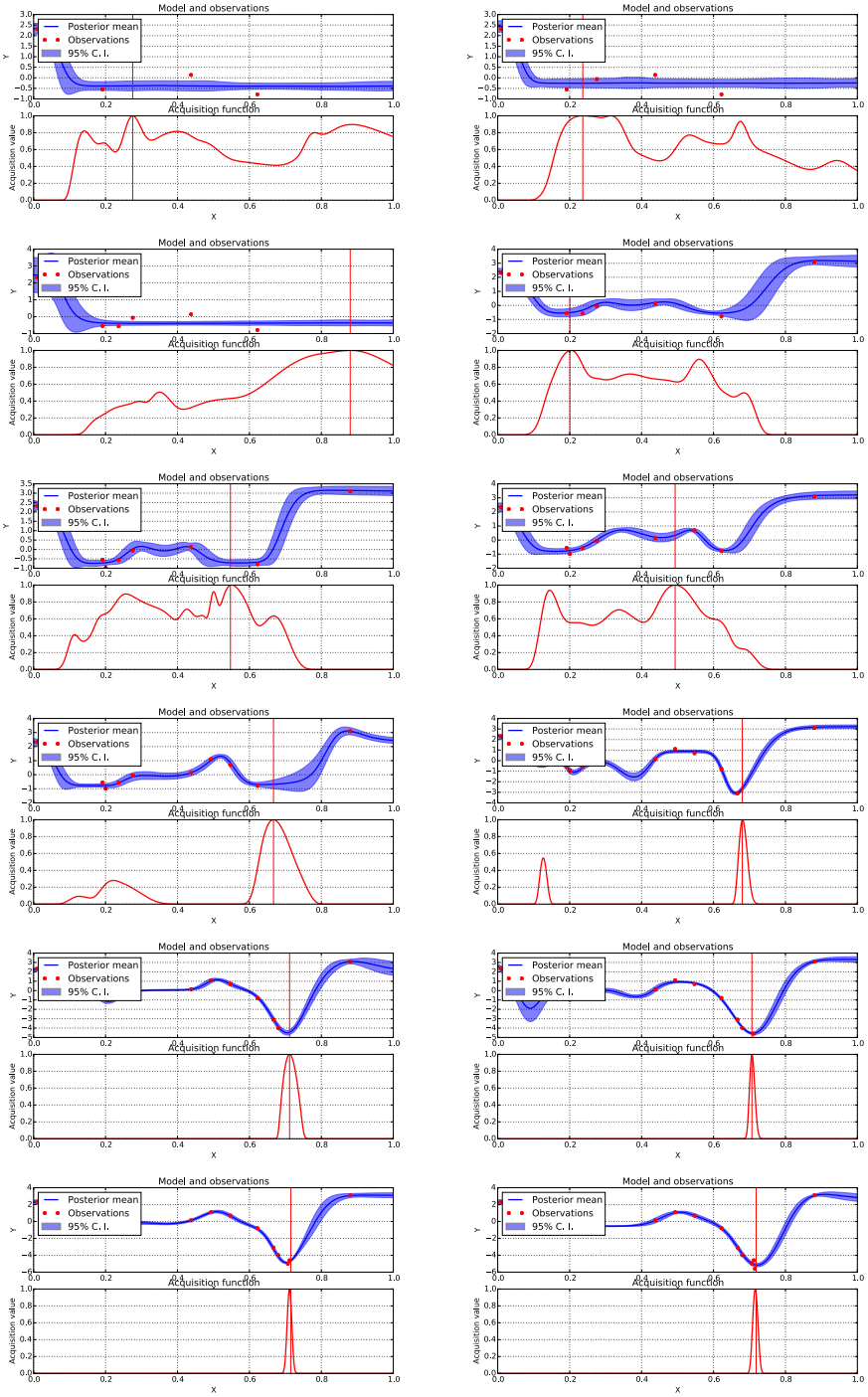

Figure 8: Whole process of experiment on a 1 dimension datasets using BNN model

Experiment on a 1 dimension datasets The first experiment is to conduct Bayesian optimization on a 1D design problem to show if the bayesian optimization provides the correct acquisition value. This set of value is not used to evaluate the effectiveness of the model. Here the $f(x) = (6x - 2)^2 \times sin(12x - 4)$, with $standarddeviation = 0.25$. The $bounds\ of\ input\ x = [(0, 1)]$. The expected minimum of the function is -6. Two points are generated randomly as starting point. The tuning process of hyper-parameter is run for $meta - iteration = 15$ times.

The Figure shows the whole hyper-parameter tuning process of Bayesian Optimization using BNN.

The figures show both the Bayesian Neural Network(BNN) and Gaussian process (GP) approximation of the objective function over 15 iterations of sampled values of the objective function. The acquisition function in the lower shaded plots is also shown in the Figure. The acquisition will achieve a high value where the model predicts a high objective(exploitation) as well as have a high prediction uncertainty. Areas with both of the attributes are sampled first. Note that for the Bayesian Optimization with BNN, it almost converged at the end of 15 meta-iterations.

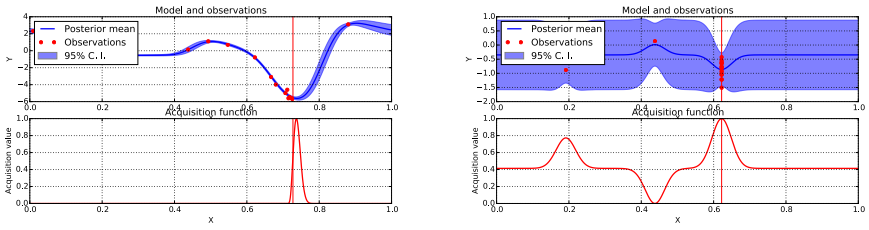

Figure 9: Experiment on a 1 dimension datasets Figure of acquisition plot BNN(left), GP(right)

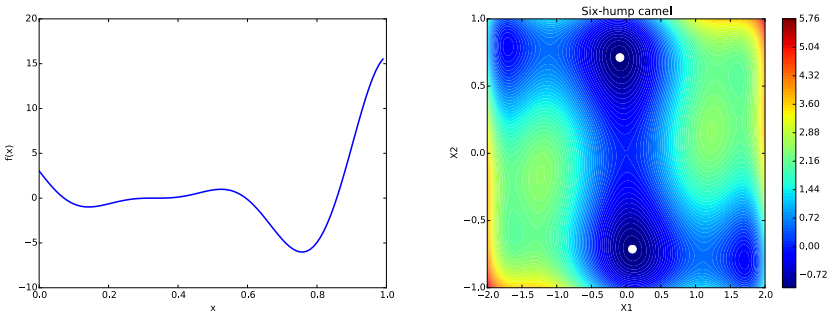

Figure 10: Real graph for Experiments with datasets of on dimension 1 and dimension 2

Experiment on a 2 dimension datasets The experiment is to conduct Bayesian optimization on a 2D design problem to show if the bayesian optimization provides the correct acquisition value. Here the $f(x) = (4 - 2.1 \times x_1^2 + \frac{1}{3} \times x_1^4) \times x_1^2 + x_1 \cdot x_2 + (-4 + 4 \times x_2^2) \times x_2^2$ with $standarddeviation = 0.1$. The $bounds\ of\ input\ x = [(-2, 2), (-1, 1)]$. The expected minimum of the function is -1.0316. The tuning process of hyper-parameter is run for $meta - iteration = 70$ times.

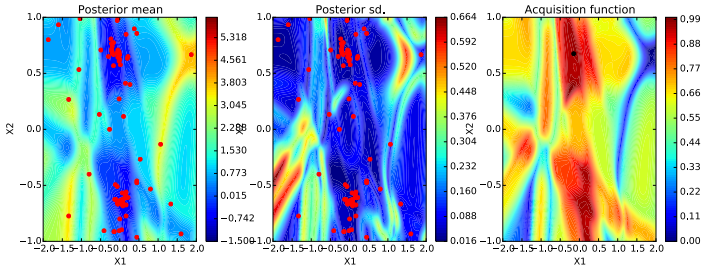

Figure 11: Experiment on a 2 dimension datasets BNN

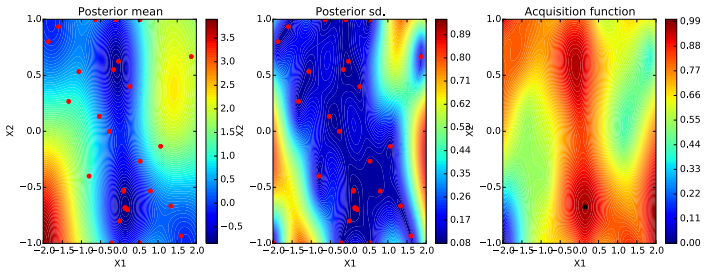

Figure 12: Experiment on a 2 dimension datasets GP

Experiment on a 5 dimension datasets The experiment is to conduct Bayesian optimization on a 2D design problem to show if the bayesian optimization provides the correct acquisition value. Here the $f(x)$ has $standarddeviation = 0.1$. The $bounds\ of\ input\ x = [(0, 10), (0, 10), (0, 10), (0, 10), (0, 10)]$. The expected minimum of the function is -100. The tuning process of hyperparameter is run for $meta - iteration = 1000$ times.

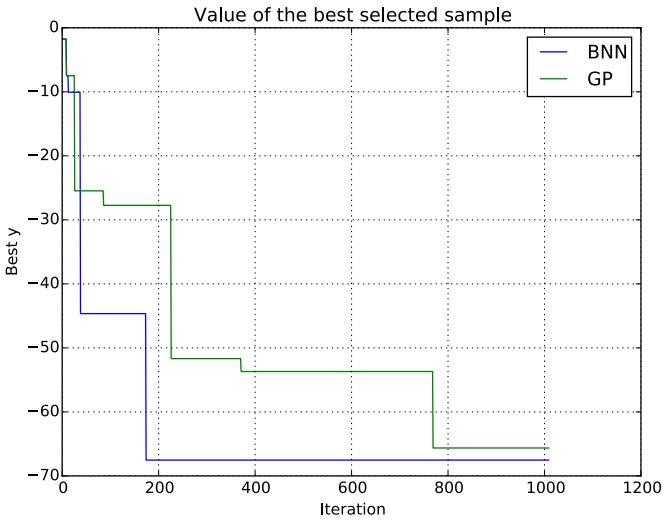

Figure 13: Experiment on a 5 dimension datasets Comparison

5.2 Experiments on benchmark datasets

In this experiment, I explore the effectiveness of Bayesian Optimization to benchmark datasets MNIST [39] and CIFAR-10 [18].

Experiment on a MNIST [39] datasets In this experiment, I evaluate these two kinds of Bayesian Optimization on MNIST dataset(40,000 for training, 10,000 for validation and 10,000 for testing) using a convolutional neural network(CNN) with two convolutional layers followed by a maxpolling layer and one hidden layer of 128 neurons as the architecture.

I optimize four hyper-parameters including learning rate, momentum, and two dropout values, with *bounds of input* $x = [(0.001, 0.1), (0.6, 0.9), (0.2, 0.7), (0.2, 0.7)]$. SGD was used to fit and train the neural network. We set the mini-batch size to 128 and each training will run for 10 epochs.

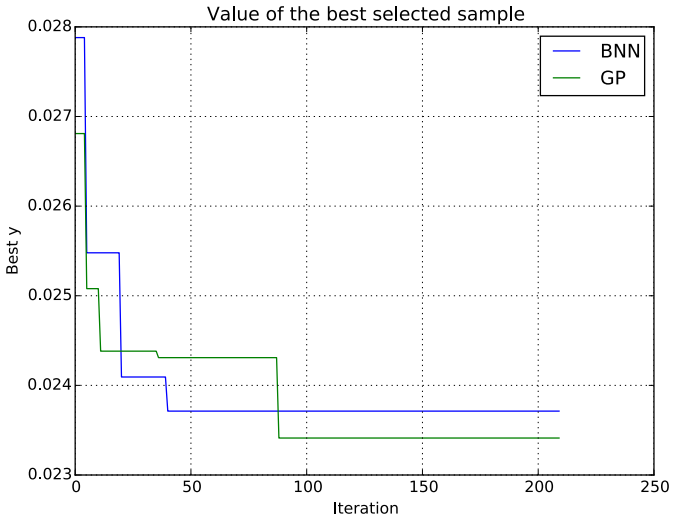

Figure 14: Experiment on MNIST datasets Comparison

Experiment on a CIFAR-10 [18] datasets In this experiment, I evaluate these two kinds of Bayesian Optimization on CIFAR-10 dataset(40,000

for training, 10,000 for validation and 10,000 for testing) using a convolutional neural network(CNN) with four convolutional layers followed by a maxpolling layer and one hidden layer of 512 neurons as the architecture.

I optimize five hyper-parameters including learning rate, momentum, and three dropout values, with *bounds of input* $x = [(0.001, 0.1), (0.6, 0.9), (0.2, 0.7), (0.2, 0.7), (0.2, 0.7)]$. SGD was used to fit and train the neural network. We set the mini-batch size to 128 and set epoch to 100. Total $meta - iteration = 100$

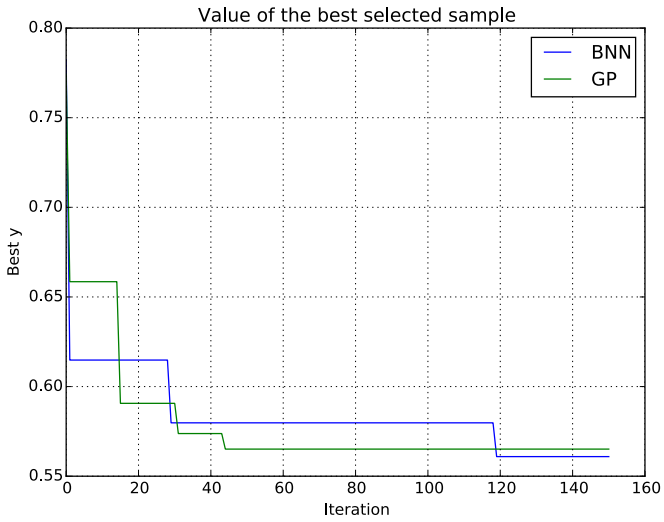

Figure 15: Experiment on CIFAR-10 datasets Comparison

5.3 Results

Dimension	f_{min}	GP	Time_s	BNN	Time_{minute}
1	-6	-1.509	22	-5.689	5.6
2	-1.0316	-1.007	22	-0.733	43
5	-100	-65.235	120	-67.524	2519

Table 2: Summarize of all results from all synthetic datasets

Dataset	GP			BNN		
	Accuracy	Loss	Time_h	Accuracy	Loss	Time_h
MNIST	0.9926	0.0234	4	0.9927	0.237	4
CIFAR-10	0.8131	0.565	51	0.8113	0.560	103.3

Table 3: Summarize of accuracy and loss of the benchmark experiments

From the results of experiments of Bayesian Optimization with Gaussian Process and Bayesian Neural Network as well as the Figures before, we can conclude that Bayesian Optimization retains the statical efficiency of Bayesian Optimization with Gaussian Process. Bayesian Optimization with Bayesian neural network performs competitively with Bayesian Optimization with Gaussian Process within the same rounds of meta-iteration. However, it will take more time for each iteration of hyper-parameter tuning.

6 Hyperparameter server for Bayesian Optimization

$layersize = [784, 50, 50, 50, 10]$
$hyperparameter\ count = 784 + 50 + 50 + 50 + 10 + 4 = 948$
$L2\ regularization = -4.0$
$metaalpha = 0.2$
$alpha = 0.1$

Table 4: Parameters related to neural networks

Experiment on MNIST datasets To demonstrate the effectiveness of hyperparameter server, I compare hyperparameter server to the initial gradient based method. Experiment is done using the MNIST datasets [39].

Although the gradient based method can work in principle for many different type of continuous hyperparameters [11], In the experiment I focus on the tuning of hyperparameters of *L2 regularization* for all weights and biases in each layer. Following the settings of Maclaurin's work [29], I evaluate

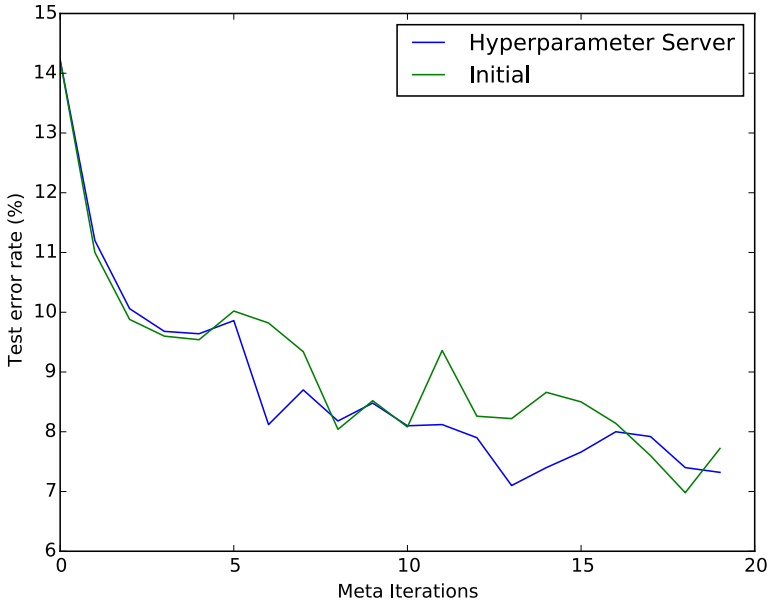

Figure 16: Comparison of test error rate obtained for hyperparameter server and the orginal optimization

the hyperparameter server on MNIST using a multilayer perceptron network (MLP) with five layers. In the experiment the total number of hyperparameters count up to 948. All hyperparameters are set initially to by initial global $L2$ *regularization* value, which is -4 in this case. Meta-iteration is run 20 times to update hyperparameters. In each meta-iteration, updating of neurons' weights is performed 500 times. The learning rate for optimizing neurons weights and biases parameters is fixed as 0.01, the learning rate for tuning of hyperparameters is 0.2. The mini-batch size is 50.

For the experiment of initial gradient based method of hyperparameter tuning, I use total 50000 samples to train and update hyperparameters(40000 is used for training purpose and 10000 is used for validating purpose).

On the other hand, for the experiment of hyperparameter server, I have

1 server node and 5 client nodes. For each client node I use 20000 samples to train and update hyperparameters and 5000 samples to validate. The sample selection is conditional random. The validation samples will be selected first and these samples will not be used in training. The first client will select first 20000 samples in the datasets and the second will select 20000 samples start from number 20000 sample and so on. Once all samples have been selected, the next client will randomly select the samples from the datasets.

Some parameters with their calculation related to the neural network are listed in Table 4 and the Figure show the effectiveness of hyper-parameter server compared to initial gradient-based hyper-parameter optimization on MNIST dataset. The Figure shows that the result with hyperparameter server framework can achieve the similar performance as the original method.

Part V

Conclusion

In this final year project, I explore the Bayesian Optimization method with Bayesian Neural Network prior. I also prove that the approach will be able to learn the kernel from scratch rather than requiring pre-setting of kernels, at the same time it will perform competitively with the GP-based approaches within the same rounds of hyper-parameter optimization meta-iterations. In return, it will take longer time for each hyperparameter tuning iteration.

What's more, I explore a hyperparameter server framework to perform distributed hyper-parameter optimization problems. The preliminary experiments show that the hyperparameter server has the ability of perform competitively compared to the initial optimization strategy within the same rounds of hyper-parameter optimization meta-iterations.

Part VI

Appendix

7 Explore of using subsets of datasets to tune the hyperparameters of neural networks

7.1 Motivation: Efficient transfer learning method

To get a machine learning method working well, many hyperparameters required to be tuned when training the models [40]. Normally the best hyperparameters set is different for different datasets [40]. Machine learning experts can find a proper well-perform hyperparameters quickly based on their expertise [40]. However, it will usually hard for end users without machine learning experiences to tune the hyperparameters.

From the recent research, one efficient way to speed up the hyperparameter tuning is to transfer information from previous trails [4] which is conducted on other datasets. The key challenge for the transferring of hyperparameters is that the evaluation metrics may not be comparable for different datasets. For example, a machine learning algorithm may perform well on a dataset and get a 90% accuracy [4]. However, the same machine learning algorithm may only get 60% accuracy on a difficult dataset [4]. Although the best accuracy got from different datasets may be different, the best hyperparameter setting region will be same [40].

7.2 Overall algorithm description

Motivated by the transferring method of hyperparameters, we explore the transferring of hyperparameters among different subsets of one dataset. We assume that the random subsets of training samples will give similar behavior

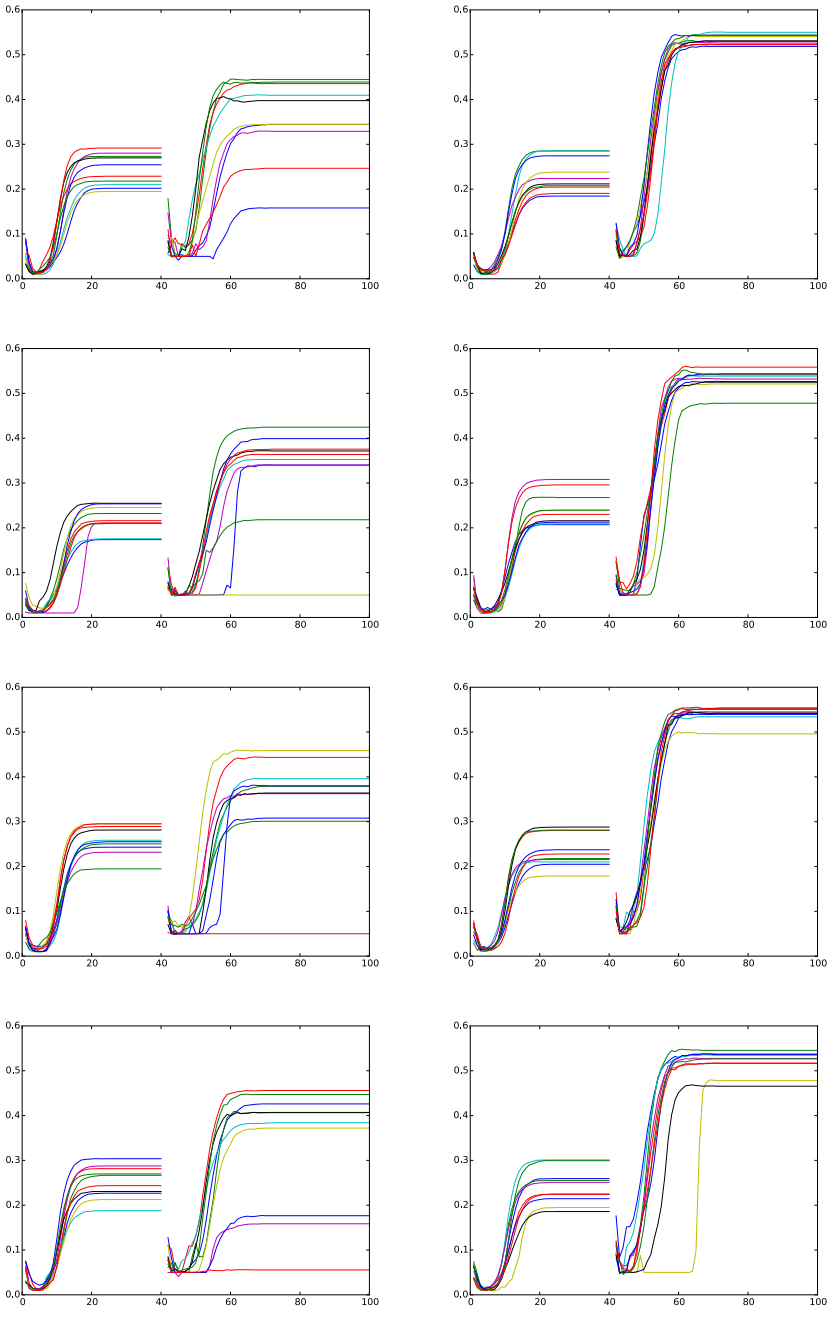

Figure 17: Experiment result on Subset 1(left 4), Subset 2(right 4)

so that we can transfer the hyperparameter from subsets of datasets to full datasets.

The whole procedure of the algorithm consist mainly two parts, namely applying bayesian optimization using subsets of datasets, apply the best hyperparameters got from subsets to the training procedure of full datasets. Subsets will be selected randomly. For example, there are total 10 classes ranging from 0 to 9, subsets of datasets containing 4 subsets may be {4,3,8,9}.

7.3 Experiments

In this experiment, we explore the effectiveness of subset method of hyperparameter tuning method to benchmark datasets $CIFAR-100$ [18]. The objective of the experiments is to validate that the best hyperparameter got for subsets will be similar to the best hyperparameter for full datasets, which means the subsets is expected to be able to perform similar behavior as full datasets.

We perform Bayesian Optimization on different subsets of the CIFAR-100 datasets and corresponding subset best hyperparameters will be saved. Each subsets of CIFAR-100 contain 20 random classes. Those hyperparameters will be applied to full datasets and one round of training process will be performed to get the loss value. The training method of datasets were using CNN. Hyperparameter tuned in the process is $learningrate$, $Inputshape1$, $Inputshape2$, $Inputshape3$, $Inputshape4$, $poolingsize1$, $poolingsize2$, $momentum$.

In this work we conducted experiments on 8 subsets of the CIFAR-100 and for each subsets we run hyperparameter for 100 $mateiterations$. The result of two experiments is shown in the following Figures. Comparison of hyperparameter with respect to accuracy distribution for subset and fullest are shown in the Figures. In the Figures the first 40 epochs represent the result when tuning the hyperparameter point on subset datasets. The last 60 epochs represent the corresponding result got when apply the same hyperparameter to the full datasets.

Subset 1 contain subclasses {5, 43, 37, 20, 10, 85, 21, 4, 0, 97, 31, 59, 83, 58, 69, 93, 38, 90, 36, 55}

Subset 2 contain subclasses {67, 3, 2, 66, 76, 79, 41, 93, 33, 43, 95, 27, 0, 96, 23, 64, 60, 98, 82, 59}

According to the Figures we can see that the subset 2 will give more similar behavior with the fullset of CIFAR-10. There exist big difference between the distribution of hyperparameters for subset 1 and full datasets of CIFAR-10.

7.4 Conclusion

From the experiments we find out that the best hyperparameters of subsets may not be the best hyperparameters for the full datasets so that the using of subset to tune the hyperparameters may not be an efficient way to tune the hyperparameters.

8 Kaggle competitions

8.1 Grasp-and-Lift EEG Detection

Objective Actions in daily life including getting dressed, brushing teeth, making coffee require the use of our body. However, many patients who lose hand function due to amputation or neurological disability are not be able to conduct such daily actions. This competition explored the possibility of building a brain-computer interface prosthetic device to increase these patients' quality of life. The target group is such patients with amputation disabilities.

The methodology of building the device is to conduct human scalping on healthy subjects first and then collect EEG signals evoked by brain activities when they perform special actions. Thereafter, the next step is to use these data to build models. Such model will be applied to EEG signals collected

from patients to classify the action patients they want to perform. The Objective of the competition is to identify from EEG signals when a hand is grasping, lifting or replacing an objects.

Data source and format Data source: EEG Recording of subjects performing grasp-and-lifti(GAL) trails.

One sample of EEG [21] recorded data include 12 subjects with 10 series of trails for every subject. Each series contains about 30 trails; the trails number may vary for each series.

Labels: There are six targets to be classified, namely handstart, first digit touch, both start load phase, lift off, replace and both released.

Trails attempted At first trail, a basic cnn deep learning is implemented to check the accuracy of cnn method on the problem. The output accuracy turned out to be 93.831%, compared to the team achieve the highest accuracy of nearing 97%. After research into the datasets, we found that the datasets contained noisy data. Deep learning is so sensitive to noise that the filtering of the initial data is required to achieve a better performance of deep learning on this case. Much time was spent on the trail of pre-processing of the datasets to decrease the effect of noisy data. The evaluation result is given in the following Figure.

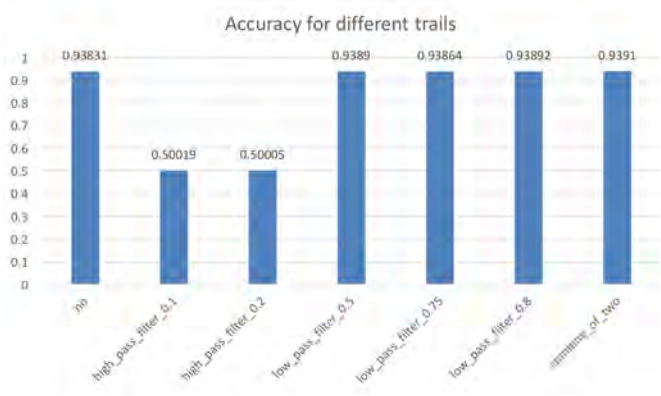

Figure 18: Accuracy for different trails

Final solution After several trails, the full solution that proven to perform a better result is shown below. For preprocessing of the datasets, a signal filter is used to filter out noisy signals. The filter technology that used for the competition was Fast Fourier transform filter. A fast Fourier transform algorithm will compute the discrete Fourier transform(DFT) of a sequence or its inverse [20]. To convert the EEG signals from time domain to a representation in frequency domain, we use Fourier analysis. After filtering process, data was then convert back to time domain. An cnn network was used to train and get the final prediction result.

The best prediction loss wa 0.939, which was slightly better than the initial one without filter. For th competition we got a ranking of top 25%.

8.2 Springleaf

Objective Springleaf offers personal and auto loans to its customers to help customer to take control of their finances. One of the most important methods to connect the company with potential customers in need of loan is direct email. In this competition, a large set of features of customers was provided by Springleaf. The aim of the competition was to explore methods

to do analysis of these features and find out the customers that will be likely to respond to the email and be a potential user for the service.

Data source and format Every row is a sample of customer information. Every column is a feature provided by Springleaf. Training data: 145231($rows$)×1934(columns) Testing data: 145232($rows$)×1933(columns)

Trails attempted Design of the detailed plan of processes is summarized as following:

- Fundamental neutral deep learning technology used was MLP, which was implemented by using of nolearn.lasagne.
- SKLearn was used in the competition to perform random tuning on several hyper- parameters for MLP. The hyper-parameters and included were hidden_num_units(range from 200 to 600), dropout_p(range from 0.2 to 0.7), update_learning_rate(range from 0.001 to 0.01) and update_momentum(range from 0.6 to 0.9)
- To increase accuracy, we plan to implement the following strategy after validation of effectiveness of MLP training on datasets: Using subsets to train model strategy. For example, the total training size is 5000, then each time 4000 of them was chosen to be used to train and test. Conduct the subset training and testing for several times. Then for each test sample, compute the mean value of the previous predicted data. The mean value will be the final result.

Final solution

1. A simple preprocessing of data was done so that all NA values will be replaced by -1. However, the result was still bad. Another data cleaning method is through the use of Google Refine(OpenRefine)3.

However, the operation of google refine on such large size of data required large computing source and was extremely slow on server with even 64G CPU. For this reason, this approach was finally given up after attempting.

2. With the date approach experiences gained in the previous Kapple competition, one more method was proposed by one member in our team. Improper values like NA and 99999999 would be filtered out from datasets first. These values will be set to NaN initially. A training of random forest will be used to predict the NaN value. For example, we have 10 columns in total. A random of places have NaN as the value. The data cleaning will perform data completion of column one by one. For column one, all other 9 columns will be used as training data and the first column will be used as label. Training will be performed by using non NaN samples and a prediction will be performed to predict the missing data. Since there was no time left for the competition, the method was not implemented. The method will be validated in next competition if the datasets provided are dirty.

3. Due to time limitation, a practical classifier random forest was then implemented finally for this competition. The data used to train and test were datasets after after undergoing preprocessing of replacing all meaningless and nan data to -1. According to LeoBreiman, using of Random Forest is more robust with respect to noise [10].

By implementing RandomForest on datasets as well as above methods, the accuracy finally achieve was 79.48%, which was much better than the results predicted by using deep learning.

References

[1] Janez Bester Andrej Krenker and Consalta d.o.o. Andrej Kos. Introduction to the artificial neural networks. Technical report, 2Faculty of Electrical Engineering, University of Ljubljana, 2011.

[2] Andrew Y. Ng Andrew L. Maas, Awni Y. Hannun. Rectifier nonlinearities improve neural network acoustic models. 2013.

[3] Barak A Pearl-mutter Atilim Gunes Baydin and Alexey Andreyevich Radul. Automatic differentiation in machine learning: a survey. 2015.

[4] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, Michele Sebag, et al. Collaborative hyperparameter tuning. In *JMLR Workshop and Conference Proceedings*, volume 28, pages 199–207, 2013.

[5] Richard Bellman, Richard Ernest Bellman, Richard Ernest Bellman, and Richard Ernest Bellman. *Adaptive control processes: a guided tour*, volume 4. Princeton University Press Princeton, 1961.

[6] Ian Goodfellow; Yoshua Bengio; and Aaron Courville. Deep learning, mit press(preparation version 2016). Book in preparation for MIT Press, 2016.

[7] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[8] Remi; Bengio Yoshua; Kegl Balazs Bergstra, James; Bardenet. Algorithms for hyper-parameter optimization. 2011.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[10] Leo Breiman. Random forest, statistic depertment, university of california,berkeley.

[11] Jie Fu; Hongyin Luo;Jiashi Feng; Kian Hsiang Low; Tat-Seng Chua. Drmad: Distilling reverse-mode automatic differentiation for optimizing hyperparameters of deep neural networks. 2016.

[12] Ryan P. Adams David Duvenaud. Black-box stochastic variational inference in five lines of python. 2015.

[13] Vlad M. Cora Eric Brochu and Nando De Freitas. A tutorial on bayesian opti- mization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. 2010.

[14] JoÃ£o F. G. De Freitas. Bayesian methods for neural networks. Technical report, Trinity College, University of Cambridge and Cambridge University Engineering Department., 1999.

[15] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1994.

[16] David Heryanto. Incremental training of neural network with knowledge distillation. Technical report, National University of Singapore, 2015.

[17] Fu Jie. Decentralized bayesian hyper-parameter optimization. 2015.

[18] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. In *Computer Science Department, University of Toronto, Tech.* 2009.

[19] Boser B. Denker J. S. Henderson D.-Howard R. E. Hubbard W. LeCun, Y. and L. D. Jackel. Back-propagation applied to handwritten zip code recognition. 1989.

[20] Charles Van Loan. Computational frameworks for the fast fourier transform, siam. 1992.

[21] Edin BB Luciw MD, Jarocka E. Multi-channel eeg recordings during 3,936 grasp and lift trials with varying weight and friction. scientific data 1:140047, 2014.

[22] Sheffield machine learning group. Gpy: A gaussian process framework in python, http://github.com/sheffieldml/gpy, 2012-2015.

[23] Sheffield machine learning group. Gpyopt: A bayesian optimization framework in python, http://github.com/sheffieldml/gpyopt, 2015.

[24] D. J. C. MacKay. Introduction to gaussian processes. pages 133–165, 1998.

[25] Jonas Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4(4):347–365, 1994.

[26] Jun Woo Park Alexander J. Smola Amr Ahmed Vanja Josifovski James Long Eugene J. Shekita BorYiing Su Mu Li, David G. Andersen. Scaling distributed machine learning with the parameter server. 2014.

[27] Kevin P Murphy. Machine learning : a probabilistic perspective. 2012.

[28] R. M. Neal. *Bayesian Nlearning for neural networks*. PhD thesis, 1995.

[29] Maclaurin Dougal; Duvenaud David; Adams Ryan P. Gradient-based hyperparameter optimization through reversible learning. 2015.

[30] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[31] Frank.x Rosenblatt. Perceptrons and the theory of brain mechanisms. *Principles of Neurodynamics*, 1961.

[32] Hugo; Adams Ryan Snoek, Jasper; Larochelle. Practical bayesian optimization of machine learning algorithms. 2012.

[33] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md Patwary, Mostofa Ali, Ryan P Adams, et al. Scalable bayesian optimization using deep neural networks. *arXiv preprint arXiv:1502.05700*, 2015.

[34] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In David Blei and Francis Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. JMLR Workshop and Conference Proceedings, 2015.

[35] Qing Song. Book review: Neural networks: Systematic introduction,, raul rojas, springer-verlag, new york, ny, 1995, 502 pp. isbn 3-540-60505-3. *International Journal of Robust and Nonlinear Control*, 1998.

[36] Nobuyuki Matsui Teijiro Isokawa, Haruhiko Nishimura. Quaternionic multilayer perceptron with local analyticity. 2012.

[37] Wikipedia. Automatic differentiation, 2016.

[38] James D Wu Xiping, Westervelt. Engineering applications of neural computing: A state-of-the-art survey. 1991.

[39] Cortes Corinna Yann LeCun and C. J. C. bURGES. Mnist handwritten digit database, yann lecun, corinna cortes and chris burges. 1998b.

[40] Dani Yogatama and Gideon Mann. Efficient transfer learning method for automatic hyperparameter tuning. In *International Conference on Artificial Intelligence and Statistics*, pages 1077–1085, 2014.