Adaptive Neural Network Representations for Parallel and Scalable Bayesian Optimization

James Brofos
Dartmouth College
Hanover, NH 03755

Rui Shu
Dartmouth College
Hanover, NH 03755

Matthew Jin
Dartmouth College
Hanover, NH 03755

Abstract

Bayesian optimization has emerged as a powerful, new technique for interpolating and optimizing a wide range of functions which are expensive to compute. The primary tool of Bayesian optimization is the Gaussian process, which permits the investigator to define a prior belief, which is then transformed into a posterior through sequential sampling of points. Unfortunately, Gaussian process interpolation suffers from a major computational bottleneck that makes it prohibitively expensive to utilize in large-scale optimization routines. In this work, we investigate the use of deep neural networks for learning a representation of the space for which a much less computationally expensive interpolation algorithm may be used. Our implementation adaptively updates the representation of the neural network as more data becomes available over time. Since this neural network-based approach is parallelizable, we are able to exploit simultaneous function evaluation and network parameter learning. We show that this neural network-based approach leads to many of the same appealing properties of Gaussian processes, and which scales only linearly in the number of observations *ceteris paribus*.

We additionally seek to provide a theoretical explanation regarding *why* an optimization algorithm of this form functions at all. For this, we develop a probabilistic theory of martingales and apply it to the cumulative regret incurred by the neural network-based approach to Bayesian optimization. We find that the performance of the deep neural network is consistent with the predictions of our probabilistic theory. We therefore speculate on the kind of representation that is learned by the network in its final hidden layer. We also show that our approach is easily extended to a parallel situation, which results in faster optimization. Numerical experiments demonstrate the efficacy of using deep networks for learning representations amenable to computationally inexpensive interpolation methods.

1 Introduction

Modern machine learning algorithms are dependent on hyperparameter tuning to increase their performance. Recently Bayesian optimization, wherein the hyperparameter space of the model is interpolated by a Gaussian process, has been recognized as a popular methodology for online hyperparameter tuning. To select the next hyperparameter configuration in this setting, one can maximize an *acquisition function* such as the expected improvement or greatest confidence bound. The acquisition function incorporates knowledge of the posterior mean and variance to derive a quantifiable exploration-exploitation trade-off at every point in the hyperparameter space. Bayesian optimization performs an intermediate maximization over the acquisition function criterion to determine the next experimental configuration of the hyperparameters to evaluate.

In Bayesian optimization the goal is to optimize a black-box function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is a compact subset of the unit hypercube. Bayesian optimization models f as a (possibly noisy) Gaussian process, wherein uncertainty is captured by a conditional normal distribution at every point $\mathbf{x} \in \mathcal{X}$. Given a set of inputs in $\mathcal{X}$ and responses from f, we can condition the Gaussian process on the set of tuples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ for $n \in \mathbb{N}$. The mean and variance of the posterior Gaussian process interpolation are provided to the acquisition function to make an informed decision which point of $\mathcal{X}$ to next query.

Bayesian optimization is especially useful when f is expensive to compute, taking perhaps hours or days to complete. Examples of functions fitting into this category include Gaussian mixture models, wherein $\mathbf{x}$ represents the number of clusters to fit and $f(x)$ is the BIC of the Gaussian mixture for some fixed data set. For problems of this form, hyperparameter optimization which assumes the form of a grid-search through $\mathcal{X}$ quickly become intractable. Hence, it is necessary to have a means of inferring the behavior of f and through which intelligent and sequential point selection can be attained. Gaussian processes have traditionally filled this role throughout the Bayesian optimization literature.

Bayesian optimization classically imposes some assumptions on the functional form of f. Namely, f is said to be drawn from a Gaussian process $\mathcal{GP}(m, k)$ with mean function $m : \mathcal{X} \to \mathbb{R}$ and covariance function (also called the *kernel function*) $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$. It can be shown that without loss of generality, the mean can be taken to be zero for every point of $\mathcal{X}$ as the Gaussian process can be completely parametrized by the covariance function.

More specifically, Bayesian optimization represents a methodology for solving the maximization task,

$$\mathbf{x}^\star = \arg\max_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}). \tag{1}$$

The efficiency of an optimization algorithm and its ability to balance exploitation with exploration is measured cumulatively for each sequential point selection. We use the *empirical cumulative regret* to evaluate the performance, which is defined at the n^{th} iteration to be the quantity,

$$R_n = \sum_{i=1}^{n} f(\mathbf{x}^\star) - f(\mathbf{x}_i), \tag{2}$$

where $\mathbf{x}_i \in \mathcal{X}$ is the point sampled at the i^{th} iteration of the algorithm. We additionally report the performance of our neural network-based optimization in terms of the *empirical minimum regret*. The minimum regret is the best performance of the algorithm on all query points at the n^{th} iteration; more formally,

$$Q_n = \min_{i \in [n]} \{ f(\mathbf{x}^\star) - f(\mathbf{x}_i) \}. \tag{3}$$

The aim of this research is to solve the optimization problem in Equation **??** without relying on a Gaussian process infrastructure. The motivation behind this being that Gaussian processes are not scalable as more information is obtained from f (for a more thorough explanation of this limitation, refer to Section **??**). Following the work of [**?**], we investigate the use of deep learning architectures for learning a representation of the hyperparameter space which is related linearly with the output of $f(\mathbf{x})$ for any $\mathbf{x} \in \mathcal{X}$.

In Section **??** we discuss the recent literature on Bayesian optimization and recent techniques that have emerged as being useful for query point selection. We then discuss how a neural network may learn representations and how these representations may be used to interpolate the hyperparameter space in Section **??**. Some theoretical justification regarding why the optimization procedure works is provided in Section **??**. We finally evaluate the performance of our neural network-based approach in Section **??** on a collection of real and synthetic machine learning problems. We find our approach to parameter optimization to be highly effective in practice and suggest it as an alternative to Gaussian process optimization when scalability becomes a concern.

2 Background and Prior Work

Our work centers around the use of neural network regression. Neural network regression is an interpolation algorithm such that for some input hyperparameter $\mathbf{x}$ outputs a prediction $\hat{y}$, where $\hat{y}$ is an estimate for some expensive black-box function $f(\mathbf{x})$. The neural network's prediction is generated by simply feeding $\mathbf{x}$ forward through the network using weights that are learned via backpropogation and stochastic gradient descent. In practice, neural network regression may be used to generate a regression estimate for a learning algorithm's performance on a test data set as a function of a set of hyperparameters.

In the context of this paper, rather than feed each training set element $\mathbf{x}_i$ through the entire network to generate some prediction y_i, we instead output the representation $\nu(x_i)$, which consists of the values of the nodes in the final hidden layer of the neural network. The set of all $\nu(x_i)$ is then regressed on using ordinary least squares to generate a predictor which, in conjunction with an acquisition function such as expected improvement, allows us to decide the next point to query.

Our experiments use the *expected improvement* criterion, which has an appealing closed-form and natural intuition. The expected improvement measures how much the function value $f(\mathbf{x})$ can be expected to improve over the current best function evaluation $f(\mathbf{x}_{\text{best}})$. For any $\mathbf{x} \in \mathcal{X}$ we define the quantity,

$$\gamma(\mathbf{x}) = \frac{\mu(\mathbf{x}; \{(\mathbf{x}_i, y_i)\}_{i=1}^n) - f(\mathbf{x}_{\text{best}})}{\sigma(\mathbf{x}; \{(\mathbf{x}_i, y_i)\}_{i=1}^n)}, \tag{4}$$

where $\mu(\cdot;\cdot)$ and $\sigma(\cdot;\cdot)$ are the posterior mean and

variance, respectively. From here, expected improvement is computed to be the quantity,

$$\sigma\left(\mathbf{x}; \{(\mathbf{x}_i, y_i)\}_{i=1}^n\right)\left(\gamma\left(\mathbf{x}\right)\Phi\left(\gamma\left(\mathbf{x}\right)\right) + \phi\left(\gamma\left(\mathbf{x}\right)\right)\right), \quad (5)$$

where $\Phi(\cdot)$ and $\phi(\cdot)$ are the cumulative and probability density functions of a standard normal random variable, respectively. In addition to the expected improvement, other common acquisition criteria include *probability of improvement* $(\Phi(\gamma(\mathbf{x})))$ and *greatest confidence bound* $(\mu(\mathbf{x}; \{(\mathbf{x}_i, y_i)\}_{i=1}^n) + \kappa_n \sigma(\mathbf{x}; \{(\mathbf{x}_i, y_i)\}_{i=1}^n)$, for $\kappa_n > 0)$.

Recent work has empirically shown that the expected improvement has better performance capabilities than does the probability of improvement. Moreover, unlike the greatest confidence bound, there is not an acquisition hyperparameter that must be tuned. The expected improvement represents an appealing nonparametric choice and is therefore the focus of these results.

Recently, much research has been devoted to the study of Gaussian process optimization, including substantial theoretical results, distributed optimization, practical considerations in Bayesian optimization, constrained optimization, and the application of reversible learning to hyperparameter selection. Gaussian processes can be classically considered as defining a distribution over functions; this usage makes them popular in the Bayesian optimization setting due to their desirable analytic, adaptable, and closed-form uncertainty properties. Recent research has focused on scaling Gaussian processes to high-dimensional optimization problems, as well as augmenting the standard approach through the incorporation of Monte Carlo simulations of the response variable and slice sampling.

3 Deep Learning Architectures and Regression

An important limitation of Gaussian processes regression is that updating the posterior belief is a cubic computation in the number of observations. In particular, the major computational bottleneck in the update equation comes from the inversion of an $n \times n$ kernel matrix. In this work we are interested in practical methods for large-scale Bayesian optimization.

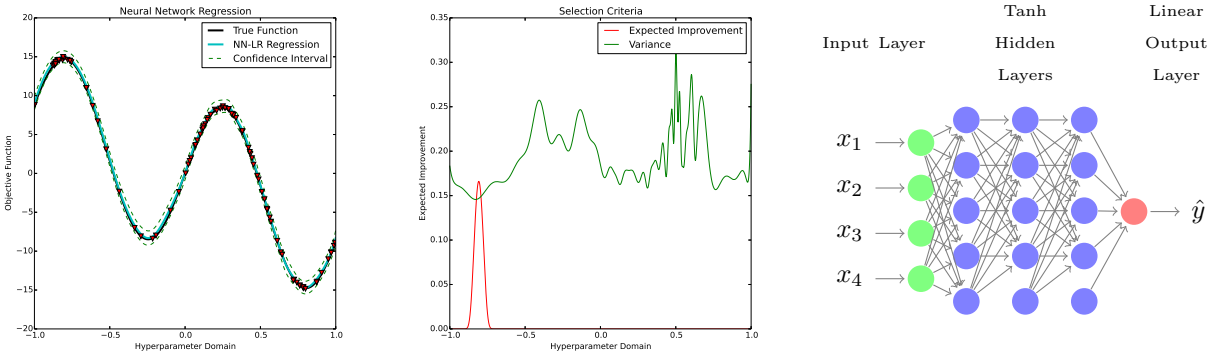
Linear regression represents the simplest form of interpolation in the modern statistics literature. In this model, a target variable is assumed to be a linear combination of some underlying set of model covariates. Under the assumptions of the model, it is possible to compute standard errors of the coefficients and confidence intervals for a point prediction made by the interpolant.

Supposing that $\mathbf{y} = \mathbf{X}\beta + \boldsymbol{\epsilon}$ for $\mathbf{X} \in \mathbb{R}^{n \times k}$, then it can be inferred from the linear regression normal equation that the least squares estimator of β is obtained by inverting a $k \times k$ matrix. This operation has the desirable property of of scaling linearly in the number of observations n and cubically in the number of covariates.

We therefore use a neural network to learn a k-dimensional representation of $\mathbf{x}$ which is *linearly* related to the response variable. In particular, we use the last hidden layer of the neural network, which, if learning was successful, can undergo a weighted sum of the activations in each hidden unit to estimate the target. The neural network also has an advantage in the sense that it is unnecessary to learn the parameters of the classical Gaussian process kernel via maximum likelihood. Parameter learning in this sense is computationally difficult, since closed-form solutions for the maximum likelihood estimates do not exist and gradient ascent algorithms are oftentimes ill-behaved.

More specifically, we propose to learn a network with k hidden layers and l hidden units per layer. The hyperparameter configuration $\mathbf{x}$ (which we had assumed early had a compact support in the unit hypercube) is provided to the input layer of the neural network. We represent the response as $y = f(\mathbf{x})$ and this value is interpolated by a single neuron at the output layer of the neural network. Let $g_i(\mathbf{x})$ be the feed-forward process by which an input $\mathbf{x}$ to the neural network is mapped to the value in the i^{th} hidden unit in the last hidden layer. Taken as a column, the l-dimensional vector $\{g_i(\mathbf{x})\}_{i=1}^l$ gives the neural network's learned representation of the input, which is linearly related with the response variable to be maximized. We therefore see that the functions $g_i(\cdot)$ depend crucially on the learned parameters of the neural network. These parameters are trained using backpropogation and stochastic gradient descent. The activation functions of the neural network are hyperbolic tangent functions for the hidden layers, but the output layer makes use of a linear activation.

For notational simplicity, let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the set of hyperparameters and associated responses for the first n training examples obtained by sampling f. Then it is possible to use the neural network to transform $\mathbf{x}_i$ into a representation that is linearly related to y_i. This is achieved by feeding $\mathbf{x}_i$ forward in the deep neural network until it reaches the last hidden layer. We denote by $\boldsymbol{\Phi}$ the $n \times l$ matrix created by feeding $\mathbf{x}_i$ through the network for every $i \in [n]$ and concatenating the representations. Then the predictive mean and variance using the learned representation of the



(a) Network Regression (b) Acquisition Functions (c) Network Architecture

Figure 1: An illustration of our proposed optimization algorithm showing in **??** the interpolation using the neural network's representation at the last hidden layer and in **??** the selection criteria (expected improvement and greatest variance) used to identify the next query point. Using linear regression we obtain confidence envelopes about the predicted mean, which are necessary for weighing the exploitation-exploration trade-off. In **??** we shown a *scaled down* version of the architecture used in our experiments: namely, our approach can handle more than four inputs to f and each hidden layer uses fifty (rather than five) hidden units. However, the architecture shown here is at least illustrative of the overall optimization algorithm. The weights connecting the last hidden layer are replaced by those learned via linear regression *post hoc*.

neural network are,

$$\mu(\mathbf{x};\mathcal{D}) = \left(\left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{y}\right)^T \nu(\mathbf{x}) \quad (6)$$

$$\sigma^2(\mathbf{x};\mathcal{D}) = \hat{\sigma}^2(1+\mathcal{G}), \quad (7)$$

where $\nu(\mathbf{x}) = (g_1(\mathbf{x}),\ldots,g_l(\mathbf{x}))^T$ and $\hat{\sigma}^2$ is the maximum likelihood estimate of the amount of noise present in the linear relationship between $\mathbf{y}$ and $\mathbf{\Phi}$ and,

$$\mathcal{G} = \nu(\mathbf{x})^T\left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\nu(\mathbf{x}). \quad (8)$$

Replacing the weights connecting the last hidden layer of the network with the linear output layer is advantageous since it allows us to capture uncertainty in the final layer's weights via the usual assumptions of linear regression. Unlike other methods of regression, such as Bayesian linear regression or ℓ_1-regularization, vanilla linear regression benefits from being without hyperparameters of its own to tune. We also find that when, as in the early states of sampling f, there is little training data, it is often the case that the neural network does not learn an effective interpolant of the hyperparameter space. However, empirically, it is the case that the representation contained in the last hidden layer are informative in the sense that their weighted sum can be taken to produce a mean function with low squared error.

3.1 Model and Optimization Details

Obviously when the dimension of the neural network's representation exceeds the number of samples thus far, linear regression leads to a degenerate matrix inversion. As a result of this, we sample a number of points equal to the number of hidden units in the final hidden layer of the network.

The expected improvement criterion may conceivably be negative throughout the support of the hyperparameter space when the optimization procedure becomes quite sure of its identified maximum. In other words, every point in $\mathcal{X}$ is expected to lead to a function evaluation $f(\mathbf{x})$ which will not exceed the current maximum. When this occurs, we change our point selection strategy to pursue pure exploration. This amounts to querying that point for which the prior variance estimate is largest.

3.2 Parallel Optimization Structure

We augment the neural network optimization model by incorporating parallel function evaluations and network learning. In many settings, it may be desirable to use a system wherein batches of parameter configurations, $\{\mathbf{x}_i\}_{i=1}^m$ are evaluated in parallel. In the design of our parallel representation learning, we consider a *high-throughput* wherein each of the m configurations to evaluate are delegated to separate processes and the response feedback is obtained asynchronously as each experiment is completed.

In practice, we adopt the master-worker parallelization paradigm. This means that one process – the eponymous master process – is devoted to selection of individual query points, which are then assigned to individual worker processes. The worker processes, upon receiving the selected hyperparameter configurations from the master process, then perform the expensive function evaluation of f. We augment this process by relearning the representation of the hyperparameter space as more information about the relationship between the domain and range of f become available. In particular, after n observations have been observed since the last time the network was learned, a single worker process is tasked with performing backpropogation and stochastic gradient descent to train the parameters of the neural network. These parameters are then broadcast to the master process, which makes use of them in all subsequent representation and regression tasks.

4 Theoretical Bounds

We seek now to provide some theoretical justification for why an algorithm of this kind should perform well on black-box optimization. We return now to the idea of regret and introduce a formal notion of regret to analyze it more carefully as a random variable using probability theory. In particular, we regret as the difference,

$$r_i = f\left(\mathbf{x}^\star\right) - f\left(\mathbf{x}_i\right)|\mathcal{D}_i. \tag{9}$$

Note that we condition on the set of existing data $\mathcal{D}_i$ by the i^{th} iteration. For n total iterations, the cumulative regret and augmented cumulative regret are given respectively by,

$$R_n = \sum_{i=1}^{n} r_i \tag{10}$$

$$M_n = \sum_{i=1}^{n} r_i - \mu\left(\boldsymbol{\nu}_i\right) + \mu\left(\boldsymbol{\nu}^\star\right)|\mathcal{D}_i. \tag{11}$$

We use the notation $\boldsymbol{\nu}$ as shorthand for $\nu\mathbf{x}$ which the representation learned by the neural network on input $\mathbf{x} \in \mathcal{X}$. Superscripts and subscripts on $\boldsymbol{\nu}$ ($\boldsymbol{\nu}^\star$ or $\boldsymbol{\nu}_i$) refer to the representation of the optimal $\mathbf{x}^\star$ or the representation of i^{th} hyperparameter query, respectively. We prove a probabilistic bound on the average cumulative regret of the neural network optimization with respect to the number of iterations that have occurred:

$$\mathbb{P}\left[\frac{R_n}{n} > \epsilon\right] = e^{-\Omega(n)}. \tag{12}$$

The proof relies on demonstrating that $(M_n)_{n\in\mathbb{N}}$ is a particular type of martingale with properties that are crucial to the derivation of Equation **??**. We begin by presenting two relevant lemmas: First, that $(M_n)_{n\in\mathbb{N}}$ is a martingale; and, second, that $(M_n)_{n\in\mathbb{N}}$ is both heavy on left and sub-Gaussian.

Lemma 4.1. *The sequence $(M_n)_{n\in\mathbb{N}}$ is a martingale with respect to its difference sequence:*

$$Y_n = M_n - M_{n-1} \tag{13}$$

$$= r_n - \left(\mu_n\left(\boldsymbol{\nu}_n\right) - \mu_n\left(\boldsymbol{\nu}^\star\right)\right)|\mathcal{D}_n. \tag{14}$$

Proof. For notational simplity, let $\mathcal{Y}_n = \{Y_i\}_{i=1}^n$. Realizations of $(M_n)_{n\in\mathbb{N}}$ are contingent on the Bayesian optimization procedure used to find the minimum of f. The key insight to understanding the GP update process is recognizing that f, conditioned on the presence of data by the i^{th} iteration $\mathcal{D}_i$, is a new Gaussian process: $f|\mathcal{D}_i \sim \mathcal{GP}(\mu_i, k_i)$. It is easy, then, letting $\mathcal{Q} = \mathcal{Y}_{n-1}, \mathcal{D}_n$, to see that $\mathbb{E}\left[Y_n|\mathcal{Y}_{n-1}\right]$ can simply be written as,

$$\mathbb{E}\left[Y_n|\mathcal{Y}_{n-1}\right] = \mathbb{E}\left[r_n - \left(\mu_n\left(\boldsymbol{\nu}_n\right) - \mu_n\left(\boldsymbol{\nu}^\star\right)\right)|\mathcal{Q}\right] \tag{15}$$

$$= 0. \tag{16}$$

Crucially, we note that the conditioning on the existing data, $\mathcal{D}_n$, causes the expectation of f to be μ_n. The variance can also be demonstrated to be,

$$\begin{aligned}\mathbb{V}\left[Y_n|\mathcal{Y}_{n-1}\right] = \; & k_n(\mathbf{x}_i, \mathbf{x}_i) + k_n(\mathbf{x}^\star, \mathbf{x}^\star) \\ & - 2k_n(\mathbf{x}^\star, \mathbf{x}_i),\end{aligned} \tag{17}$$

simply by computing the variance of $f(\mathbf{x}_i) - f(\mathbf{x}^\star)|\mathcal{D}_n$, which is the difference of two normally distributed random variables. Together, this shows that $Y_N|\left\{Y_n\right\}_{n=1}^{N-1}$ is zero-mean and normally distributed. Because this random variable has zero-mean, it can now be shown that $(M_N)_{N\in\mathbb{N}}$ is, in fact a martingale. Indeed,

$$\mathbb{E}\left[M_{n+1}|\mathcal{Y}_n\right] = \mathbb{E}\left[M_n + Y_{n+1}|\mathcal{Y}_n\right] \tag{18}$$

$$= M_n. \tag{19}$$

This completes the proof. □

Lemma 4.2. *The martingale sequence $(M_n)_{n\in\mathbb{N}}$ is heavy on left, and sub-Gaussian.*

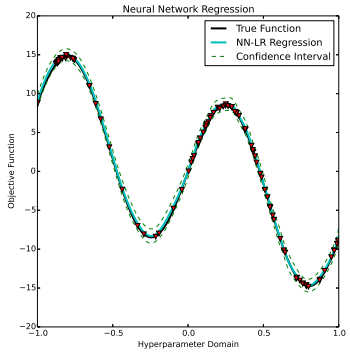
Proof. We say that a martingale is heavy on left if all its increments are conditionally heavy on left. The increments are heavy on left if it can be shown that

$$\mathbb{E}\left[Y_n|\mathcal{Y}_{n-1}\right] = 0. \tag{20}$$

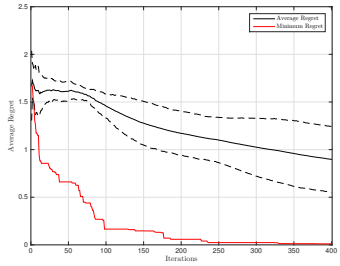
Furthermore,

$$\mathbb{E}\left[\min(|Y_n|\mathcal{Y}_{n-1}, a)\,\text{sign}(Y_n|\mathcal{Y}_{n-1})\right] \leq 0, \tag{21}$$

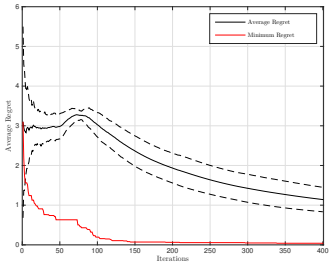
for every $a > 0$. Because $Y_n|\mathcal{Y}_{n-1}$ is a zero-mean Gaussian random variable, the above conditions are



(a) Parallel Speed Increase



(b) Gaussian Mixture Regret



(c) 4-D Hartmann Regret

Figure 2: We show an illustration of the speed-up that is possible by exploiting a parallel optimization structure in **??**. In this example, we use four total processes running at parallel: two devoted to evaluating f at the chosen points in $\mathcal{X}$, one for managing work as new information becomes available, and one for updating the neural network representation learned by the neural network as more training data is acquired. In **??** we show the results of running our neural network-based optimization algorithm on a Gaussian mixture model with multiple modes in the unit hypercube. Additionally in **??** we show the results of trying to optimize the four-dimensional Hartmann function. Notice that in both cases the optimization successfully finds the global maximum with the function domain as indicated by the minimum regret. Furthermore, the average cumulative regret is trending downwards, as desired.

trivially true. The second condition Equation **??**, in particular, can be shown by exploiting the symmetry of a zero-mean Gaussian distribution. Additionally, we say that the martingale is sub-Gaussian if there exists some $\alpha > 0$ such that for all $n \geq 1$ and $t \in R$,

$$\mathbb{E}\left[e^{t \cdot Y_n} | \mathcal{Y}_{n-1}\right] \quad \leq \quad e^{\frac{\alpha^2 t^2}{2} \mathbb{V}[Y_n | \mathcal{Y}_{n-1}]}. \tag{22}$$

Since $Y_n | \mathcal{Y}_{n-1}$ is zero-mean Gaussian, it has a well-defined moment-generating function that can be used to show that Equation **??** is true when we take $\alpha = 1$, thus demonstrating that the martingale $(M_n)_{n \in \mathbb{N}}$ is sub-Gaussian. □

Theorem 4.3. *We now prove the main result, demonstrating the rapid convergence of the average cumulative regret to zero. In particular,*

$$\mathbb{P}\left[\frac{R_n}{n} > \epsilon\right] = e^{-\Omega(n)} \tag{23}$$

Proof. There are many existing inequalities known for martingales which are self-normalized with respect to their quadratic variation. For sub-Gaussian, heavy on left martingale such as (M_n), it is known that, for all values $x > 0, a \geq 0, b > 0$,

$$\begin{aligned} &\mathbb{P}\left[\frac{M_n}{a + b\left\langle M \right\rangle_n} \geq x\right] \\ &\leq \inf_{p > 1} \left\{ \mathbb{E}\left[e^{-(p-1)\frac{x^2}{\alpha^2}\left(ab + \frac{b^2}{2}\left\langle M \right\rangle_n\right)} \right] \right\}^{\frac{1}{p}}, \end{aligned} \tag{24}$$

where α (in the case of $(M_n)_{n \in \mathbb{N}}$, $\alpha = 1$) is the value used to satisfy the condition in Equation **??** and $\left\langle M \right\rangle_n$ is the predictable quadratic variation, given by,

$$\begin{aligned} \left\langle M \right\rangle_n &= \sum_{i=1}^{n} \mathbb{E}\left[\left(M_i - M_{i-1}\right)^2 | \mathcal{Y}_{n-1} \right] && (25) \\ &= \sum_{i=1}^{n} \mathbb{V}\left[Y_i | \mathcal{Y}_{n-1} \right]. && (26) \end{aligned}$$

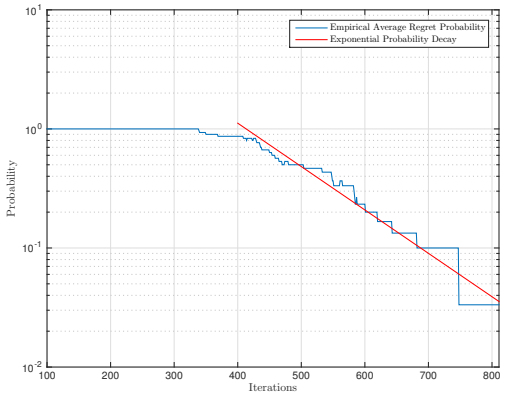
For notational convenience, we shall express $\sum_{i=1}^{n} \mathbb{V}\left[Y_i | \mathcal{Y}_{n-1}\right]$ as $\sum_{i=1}^{n} \mathbb{V}_i$. To attain a form more amenable to our main result, we set the parameters to be $a = 0, b = n, \alpha = 1$. Thus,

$$\begin{aligned} &\mathbb{P}\left[\frac{M_n}{n\left\langle M \right\rangle_n} \geq x\right] \\ &\leq \inf_{p > 1} \left\{ \mathbb{E}\left[e^{-(p-1)x^2 \frac{n^2}{2}\left\langle M \right\rangle_n} \right] \right\}^{\frac{1}{p}}. \end{aligned} \tag{27}$$

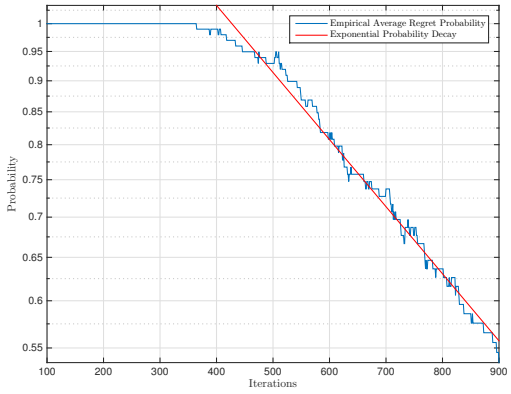
The internal expectation expression can quickly be reduced as follows,

$$\mathbb{E}\left[e^{-(p-1)x^2\left(\frac{n^2}{2}\left\langle M \right\rangle_n\right)} \right] = \left(e^{\frac{x^2 n^2}{2} \sum_{i=1}^{n} \mathbb{V}_i} \right)^{1-p}. \tag{28}$$

Thus far, we have dealt purely with the augmented cumulative regret, (M_N), exploiting the known properties of a self-normalized, heavy on left, sub-Gaussian martingale to arrive at the inequality demonstrated in Equation **??**. To derive an inequality with respect to



(a) Empirical Probability on Gaussian Process



(b) Empirical Probability on 4-D Hartmann Function

Figure 3: We show an empirical experiment that demonstrates behavior consistent with the prediction of the theory. In **??**, we seek to find the global maximum of a synthetic Gaussian process defined on an interval of $\mathbb{R}$, which is scaled to $[-1, +1]$. The theory implies that for any ϵ (here $\epsilon = 0.7$), the probability that average regret will exceed ϵ decreases exponentially for every $n > n_0$, for some n_0 dependent on ϵ. We speculate here that $n_0 \approx 400$. We run the optimization procedure one-hundred times and calculate empirically the number of times that the average cumulative regret exceeded ϵ for every n. We show the probability on a logarithmic scale and confirm that it does display the exponential decrease. An identical experiment was performed for the (much harder) Hartmann optimization problem in four dimensions. The results are shown in **??**.

the cumulative regret, we reduce the augmented cumulative regret into its components, $M_N = R_n - S_n$, where $S_n = \sum_{i=1}^{n} \mu_i(\mathbf{x}_i) - \mu_i(\mathbf{x}^\star)$. It can now be stated that,

$$\mathbb{P}\left[\frac{R_n}{n} \geq x \sum_{i=1}^{n} \mathbb{V}_i + \frac{S_n}{n}\right] \leq \left(e^{\frac{x^2 n^2}{2} \sum_{i=1}^{n} \mathbb{V}_i}\right)^{-\frac{1}{2}}. \quad (29)$$

Setting $\epsilon = x \sum_{i=1}^{n} \mathbb{V}_i + \frac{S_n}{n}$, the derivation of the main result is completed,

$$\begin{aligned} \mathbb{P}\left[\frac{R_n}{n} \geq \epsilon\right] &\leq \left(e^{\frac{(n\epsilon - S_n)^2 n^2}{2n^2 (\sum_{i=1}^{n} \mathbb{V}_i)^2} \sum_{i=1}^{n} \mathbb{V}_i}\right)^{-\frac{1}{2}} & (30) \\ &\leq \left(e^{\frac{S_n^2 - 2S_n n\epsilon + n^2\epsilon^2}{2\sum_{i=1}^{n} \mathbb{V}_i}}\right)^{-\frac{1}{2}} & (31) \\ &= e^{-\Omega(n)}. & (32) \end{aligned}$$

This shows the desired result. □

We demonstrated empirically that the neural network-based optimization algorithm is consistent with the predictions of this theory. In particular we calculate the empirical probability that the cumulative regret exceeds ϵ on two global optimization problems. We observe that in both experiments the probability did decrease exponentially after a particular number of queries had been performed. This result is shown in Figure **??**. For the Gaussian process in Figure **??** the rate of exponential decay is approximately $\frac{-84}{1000}$ while for the Hartmann function in Figure **??** the rate is approximately $\frac{-12}{1000}$.

We would therefore like to speculate that the neural network is learning a representation of the hyperparameter space $\mathcal{X}$ such that generally the Gauss-Markov assumptions of ordinary linear regression hold. If then this is true, the martingale properties hold.

5 Experimental Results

Python code for implementing neural network-based learning of hyperparameter representations is made available online at https://github.com/RuiShu/Neural-Net-Bayesian-Optimization.

In order to empirically demonstrate the efficacy of the proposed approach, we evaluate its performance on a number of benchmark global optimization problems. In particular, we analyze the performance of the neural network's representation by initializing the optimization procedure with a sample of fifty points drawn uniformly at random from the support of the hyperparameter space.

The first of our experiments involves finding the maximum of a two-dimensional Gaussian mixture. We

choose the Gaussian mixture so that it has a multimodal property, which permits us to assess the capability of the algorithm to balance exploitation with exploration. We also implement the four dimensional Hartmann function to evaluate the optimization performance of our approach. The Hartmann function is also multimodal, and it permits us to examine performance on optimization problems with a number of hyperparameters consistent with many modern statistical learning algorithms such as ℓ_1-regularized regression or SVMs. It has been used as a benchmark by, for instance, Snoek *et al.* [**?**].

We compare the empirical average regret, which is the ratio R_m/m for each $m \in \{1, 2, \ldots, n\}$. Ideally, an optimization algorithm will possess the property that it is asymptotically *no-regret* [**?**]: that is, in the limit $\lim_{m\to\infty} R_m/m = 0$. The average regret can be interpreted as a convergence rate for the optimization algorithm, wherein $\arg\max_{x\in\{\mathbf{x}_1,\ldots,\mathbf{x}_n\}} f(\mathbf{x})$ does not deviate from $f(\mathbf{x}^\star)$ by more than the average.

Figure **??** shows the average and minimum regret obtained by our algorithm on the benchmark problems. We see that generally, by using the expected improvement criterion, the algorithm explores the domain of the function first before honing in on a particular global minimum. This is seen by the initial non-decreasing nature of the average cumulative regret and the eventual attainment of zero minimum regret.

We performed our experiments on a dual-core 2.9 GHz MacBook Pro using in each experiment a total of four processes. Using this computer, optimizing the Gaussian mixture took an average of 104.003 seconds to perform one-hundred queries; for the Hartmann function, the algorithm took 288.394 seconds for the same number of queries.

6 Conclusion

In this work, we proposed a distributed methodology for optimizing expensive black box functions through deep learning. In particular, we learn a representation of the hyperparameter space which has a linear relationship with the target variable to be maximized. Unlike earlier approaches using deep learning, our approach does not have additional hyperparameters that must be tuned beyond the network architecture. Because of the nature of linear regression, the computational complexity of interpolation scales linearly in the number of points queries; this is an immense order improvement over the cubic performance of traditional Gaussian process optimization. Our approach also benefits from being fully parallelizable on a computing cluster which leads increased wall-clock time improvements in speed. We demonstrate empirically that our distributed approach achieves competitive performance with state-of-the-art Bayesian optimization procedures.

We were able to demonstrate a theory whose prediction the neural network optimization algorithm appears to obey.

Several approaches have been considered in the Gaussian process optimization literature for reducing the computational complexity of the full Bayesian update. One method in particular is the Lazy Variance Calculation [**?**], which avoids the inversion of the $n \times n$ covariance matrix altogether. One direction of future research is to consider similar techniques as this for linear regression, so that the posterior variance can be made, in a sense, to be independent of the response of querying f. This could lead to an additional degree of parallelism and a superior distributed exploration-exploitation trade-off.

Acknowledgments

Our neural networks make use of the Theanets network model regression software, which is itself reliant on the Theano software library. Basic linear algebra, including the computation of the least-squares coefficient estimates, is done using Python's scientific computing package NumPy. Parallelization was implemented using MPI4Py, a Python wrapper for an underlying C implementation of the Message Passing Interface.

We wish to thank Professor Amit Chakrabarti of Dartmouth College for supervising the independent study by which the authors derived the theoretical results contained herein.

References

[1] Auer, P. Using confidence bounds for exploitation exploration trade-offs. *JMLR*, 3:397422, 2002.

[2] Bercu, B. and Touati, A. Exponential inequalities for selfnormalized martingales with applications. *The Annals of Applied Probability*, 18(5):18481869, 2008.

[3] Bergstra, James S., Bardenet, Remi, Bengio, Yoshua, and Kegl, Balazs. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*. 2011.

[4] Contal, E., Buffoni, D., Robicquet, A., and Vayatis, N. Parallel Gaussian process optimization with upper confidence bound and pure exploration. In *Machine Learning and Knowledge Discovery in Databases*, volume 8188, pp. 225240. Springer Berlin Heidelberg, 2013.

[5] Cover, T. M. and Thomas, J. A. *Elements of Information Theory*. Wiley Interscience, 1991.

[6] de Freitas, N., Smola, A. J., and Zoghi, M. Exponential regret bounds for Gaussian process bandits with deterministic observations. In *Proceedings of the 29th International Conference on Machine Learning*. icml.cc / Omnipress, 2012.

[7] Thomas Desautels, Andreas Krause, and Joel Burdick. Parallelizing exploration-exploitation tradeoffs with gaussian process bandit optimization. In *International Conference on Machine Learning (ICML)*, 2012.

[8] Hensman, J, Fusi, N, and Lawrence, N.D. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, 2013.

[9] Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick, Zhang, Zhengyou, and Tu, Zhuowen. Deeply supervised nets. In *Deep Learning and Representation Learning Workshop*, *NIPS*, 2014.

[10] Osborne, Michael A., Garnett, Roman, and Roberts, Stephen J. Gaussian processes for global optimization. In *Learning and Intelligent Optimization*, 2009.

[11] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable Bayesian Optimization Using Deep Neural Networks. *arXiv:1502.05700 [stat.ML]*, 2013. URL http://arxiv.org/abs/1502.05700.

[12] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[13] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

[14] Srinivas, N., Krause, A., Kakade, S., and Seeger, M. Information-theoretic regret bounds for Gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):32503265, 2012.

[15] Wang, Ziyu, Zoghi, Masrour, Hutter, Frank, Matheson, David, and de Freitas, Nando. Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, 2013.