Literature Review

Christopher MacKinnon

November 2020

1 Hyper-Parameters

Hyper-parameter optimisation is traditionally considered a noisy, black-box optimisation problem. A manual search combing a grid search and expert knowledge has been the standard approach to this problem in the past. It was shown that a random search could perform as well as, or better than, grid search in the same time [6]. This type of search has become a minimum benchmark or starting point for many hyper-parameter optimisation techniques. [14]**48** Recently, taking advantage of the increase in parallel and cloud computing power has become a key aspect of these systems, requiring robust methods that can operate efficiently at scale. Hyper-parameters can contain continuous, discrete and categorical variables requiring flexible approaches to find optimal solutions across a wide range of architectures and algorithms.

2 Architecture Search

There has been an increase in interest in Neural Architecture Search (NAS) recently as these methods began to outperform human designed models. [24][22] These systems are able to design at scale creating intricate topologies for large, deep neural networks (DNN), while discovering novel variants in the components used to build these networks. [16] [18]

3 Genetic Algorithms

Genetic algorithms borrow optimisation strategies from those found in nature. One of the main benefits of genetic algorithms is the flexibility in terms of categorical, discrete and continuous variables. GAs are naturally suited to the creation of novel topologies[3] [16], and can often produce targeted architectures that deviate from the generalised templates that commonly used due to their effectiveness across a range of domains. GAs have also been used on a more granular level, creating novel variants of network components suited to a specific task [16]. Evolutionary optimisation is composed of two main processes, Mutation and Selection. Mutation in this application broadly describes

an operation which alters the traits, or creates new members of, a population. Selection contrasty, is a process which contracts the population based on some criterion. This is commonly some fitness evaluation on the objective function.

3.1 Selection

The selection process in a GA defines much of the character of the overall system, due to tenancy of naive evolutionary systems to become trapped in local optima.

The selection process can take place prior to the mutation of population or subsequently, reducing the expanded population back to its original size. This can change the expression of the overall system significantly as the former can result in a greedier search, however, the latter generally significantly larger computational costs, warm-starting or surrogate evaluation methods sub-sample the expanded population [14], [19], [21].

3.2 Truncation Selection

Truncation Selection or Elitist Selection is a simple approach to selection where the population is ranked by a criterion and all members below a threshold P are eliminated.

3.3 Crossover and Genetic Encoding

Crossover is the recombination of two successful networks into a "child" network which is common to many GA approaches to NAS. [20][3][21] This can be implemented through a wide variety of mechanisms based on the network encoding method which is used. Ablation tests on GA systems have shown that while crossover can result in an overall improvement if carried out correctly, it can often be far less significant that other aspects of GA. [21][3]

One of the core components to GA is the genetic encoding of model architectures. This is an important aspect of a successful GA due to the reliance of crossover on this representation of the network to carry meaningful aspects of the models forward. [3] Another issue that is important to consider when designing a genetic encoding methods is the problem of competing conventions or the permutation problem[1][3]. In the context of ANNs this refers to a scenario where the same architecture can be represented by multiple encodings. This can lead to inefficient allocation of computational resources and irregularities during crossover operations. [3]

3.4 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) [3] describes a framework for evolving neural networks which addresses many of the issues faced by GA, such as a formalised system for crossover, genetic encoding, the permutation problem and the protection of innovation. NEATs success is limited to smaller

networks, however, there have been several extensions to NEAT, adapting it for larger, deeper networks. [4][16]

NEAT makes use of an innovation number which denotes the order in which mutations occurred to the network. This enables a matching of connections or nodes in the crossover process between networks giving the method a structured process. Speciation is a component of GA which separates models based on genetic distance into subcategories. This protects innovation allowing solutions to be explored and reach maturity without competing with the wider population. In practice this creates a less greedy algorithm supporting a greater diversity of solutions and, in theory, improving performance on multi-modal functions.

Ablation testing was able to show the dependence of the wider system on each of these components. Speciation was shown to offer a very significant contribution to overall success of the method, while aspects such as crossover was shown to have a much smaller, but still statically significant, improvements in performance.

3.5 CoDeepNEAT

CoDeepNEAT [16] is an expansion to NEAT for use with deep neural networks. This approach was able to achieve near parity with SOTA on CIFAR-10 as well as image captioning benchmarks.

CoDeepNEAT is based on a bi-level optimisation approach to the CASH problem, implementing a hierarchical approach in which "blueprints" and modules are evolved independently. A blueprint is a graph of nodes which are replaced with modules to create complete networks for evaluation. The fitness score of a network is applied to both the blueprint and the modules, with the module score being an average of all the networks which contained the blueprint. This method was also applied to LSTM networks, which was able to produce an LSTM variant which outperform the standard LSTM cell.

3.6 N-Level Hierarchical Representation

[18] introduced a nested system for NAS which expands upon the method described in [16] from a bi-level hierarchy to an Nnlevel hierarchy. In this system, primitive operations (i.e. convolutional cells, linear cells) are considered level one. level two representations are a set of graph structures combining these lower level cells. These second level representations are then combined replacing the nodes in a graph to create higher level representation.

3.7 Population Based Training

[14] introduced a form of GA which takes inspiration from bandit problems called Population Based Training (PBT). This was able to outperform human tuned networks on reinforcement learning and imagine classification tasks.

This is a system for hyper-parameter optimisation which utilizes a variant of the successive-halving**SH** method used in Hyperband (see Sec 4.1) for

early-stopping, A portion of the population abandon their search and copy the structure, hyper-parameters and weights of high performing networks. The hyper-parameters of this new replicated network are then mutated randomly to allow for more thorough examination of lucrative search spaces. [19] introduces a similar system targeted at Neural Architecture Search (NAS). This method of exploiting successful networks has the additional advantage of producing hyper-parameter schedules that vary over the training process.

3.8 LEMONADE

[19] approach the problem of NAS with a multi-objective based method that seeks to retain network function across generations via Lamarckian inheritance. For applications in embedded systems or other computational limited environments, the complexity of the network can be an important factor. This system utilizes a Pareto front in the selection process to select a curve of models based on performance and complexity.

3.9 NSGA-NET

NSGA-Net [21] is a GA based multi-objective optimisation system for NAS which is build upon NSGA-II[2] and incorporates Bayesian Optimisation methods. This approach was able to achieve parity with SOTA on the CIFAR-10 data-set compared with other NAS solutions (in particular RL based methods) at a significantly lower computational cost using a similar multi-objective method as [19].

This approach encodes a group of nodes (a single computational unit, i.e. convolution, pooling in this particular application) as binary strings, these are describes as phases, which are then stacked to describe an entire model. Mutations are applied through bit-flipping of this encoding. Crossover was achieved through a comparison of binary strings where common bits from both parents are retained and unique bits are chosen at random from either parent.

The final stage of this technique involves applying a Bayesian Network to exploit the phases and their ordering in successful networks to probability distributions which are sampled from to create new solutions.

3.10 Aging Evolution

[2] produces a GA variant in which the population is selected against based upon the "age" of the network. This method was able to set a new SOTA when scaled up on ImageNET. This method maintains a population of networks from which a sub-population of size S are selected and evaluated. The highest scoring network is mutated to create a new child network which replaces the oldest model in the population.

4 Many-Armed Bandit and Early-Stopping

The problem of hyper-parameter optimisation can also be framed as a many armed bandit problem. A bandit problem describes a problem where an agent has a number of possible actions which yield a reward. Initially the association between an action and its reward is unknown to the agent. In a classic stochastic bandit problem, each action produces a reward randomly based on the probability distribution of reward linked to that specific action. The agent must therefore make a series of actions to gain information about reward associated with each action while also attempting to maximise the total reward. Figure **??** shows an example of this type of problem.

Figure 1: Bandit problem example with three actions denoted as x_1,x_2 and x_3. Each of these actions has an probability curve over a range of reward values which represents an agents belief about the value of each action. In this case there is a much higher uncertainty in x_2 than x_3 which has a higher mean value. A highly greedy approach to this problem would lead to the exploitation of x_3 rather than the exploration of x_2, which may not the optimal solution.

Hyper-parameter optimisation can be considered a non-stochastic variant of this problem. In this case an action or 'pull' becomes a single, or collection of, training iterations with a set of hyper-parameters. The models cost function or loss on the validation data set becomes the associated reward. In this application it is only the simple regret that is of interest rather than the cumulative regret as it is only the final recommendation which the search is evaluated on.

$$l_{\theta,t} = \frac{1}{|VAL|} \sum_{i \in VAL} loss(f_{\theta,t}(x_i), y_i) \tag{1}$$

Equation 1 describes the reward in this application, the cost function over the validation set. where $f_{\theta,t}$ is the function modelled by the algorithm that is being optimised, after a number of training steps t and a set of hyper-parameters θ

4.1 Successive Halving

Successive Halving is an algorithm first introduced in **??** for stochastic bandit problems and later adapted for the non-stochastic domain of hyper-parameter

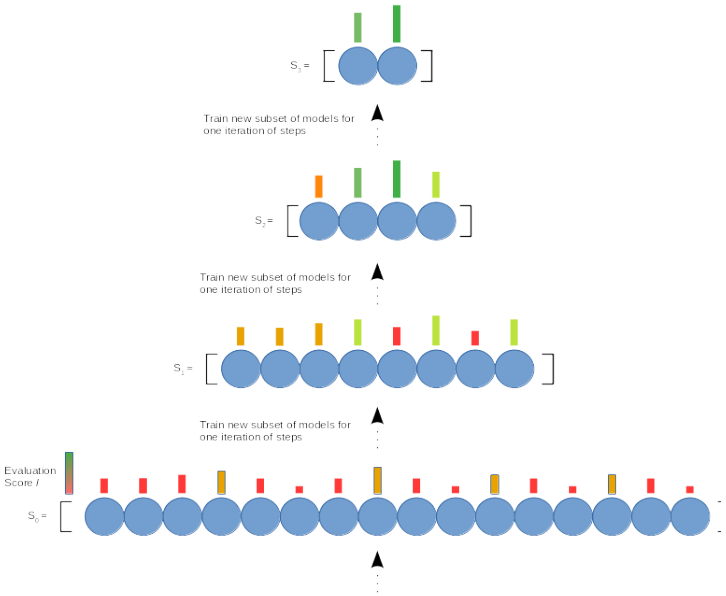

Figure 2: Four steps of the Successive Halving algorithm. This allows more resources from a set budget to be allocated to promising models.

optimisation **??**. This technique has become the basis for much of the bandit style approaches to hyper-parameter optimisation. In this approach the set of possible actions is constrained to a randomly initialised set of hyper-parameter settings, S_0. Each iteration of the algorithm has the set of models trained for a number of steps based on a budget, B, and the number of models ,n. after each training interval the models are evaluated according to 1. The set of S_0 is then subset to half its size based on this metric. Figure 2 shows an a series of iterations of Successive Halving which is more formally described in Algorithm 1.

Algorithm 1: Successive Halving for Hyper-parameter Optimisation

Data: Initial Set of hyper-parameter settings $S_0 = \{n\}$,*budget* B

Result: Final set containing single model, $S_{k=log_2(n)}$

for $k = \{1,2,...,log_2(n)\}$ **do**
  for $j \in S_k$ **do**
    Train $S_{k,j}$ *for* $\frac{B}{|S_k|log_2(n)}$*steps*
    $l_k = \{l_k, \frac{1}{|VAL|}\sum_{i \in VAL} loss(S_{j,k}(x_i), y_i)\}$
  end
  for $j_s, j_l \in S_k, l_k$ **do**
    if $j_l > median(l_k)$ **then**
      $S_{k+1} = \{S_{k+1}, j_s\}$
    end
  end
end

This type of approach is also referred to as Early-Stopping and is based on the assumption that a partially trained models performance will be predictive of its final performance.[32][47] This assumption can, however, become strained in applications where the convergence time of models has a significant variance across the search space. One of the limitation of this type of solution, in particular when the initial set of models is seeded from a simple random search, is final performance with a large computational budget. Sequential methods such as Bayesian Optimisation or Genetic Algorithms are often able to achieve better performance given a large enough budget.This has lead to the incorporation of bandit based strategies into Bayesian and Evolutionary techniques to improve the any-time performance as well as allowing for greater exploration, due to the increase in efficiency. [14], [17], [27]

4.2 HyperBand

HyperBand **HB** is an popular example of this strategy and has shown the effectiveness of this method, in particular with respect to any-time performance. HyperBand was able to achieve a reduction in training time of an order of magnitude over Bayesian methods while maintaining only a minor reduction in performance.

HyperBand can be considered a resource allocation algorithm for Successive-Halving **SH**, running as a meta layer around a series of Successive-Halving inner loops which are referred to as 'brackets'. This approach attacks the problem of balancing the resources budget B, between the training time for each model r, with the total number of models trained n. Due to the variance of model convergence time between different algorithms or problem domains, this cannot be a static, fit all value.

HyperBand effectively performs a grid search over values of n and r, running a sequence of tests for different values of n within a bounded range.

Algorithm 2: HyperBand

Data: R, η

Initialise: $s_{max} = [log_\eta(R)], B = (s_{max} + 1)R$, *Model evaluation scores* $l = \{\}$

for $s = \{s_{max}, s_{max} - 1, ..., 0\}$ **do**

- $n = \left\lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \right\rceil$
- $r = R\eta^{-s}$
- $S_0 \leftarrow$ *randomly_generate_hyperparameter_sets*(n)
- $l \leftarrow SucessiveHalving * (S_0, r, \eta)$

end

Return *max(l)*

Algorithm 2 describes HyperBand and its relation to Successive-Halving which is modified in this case to take a minimum amount of resources, r and η which denotes fraction of models which are eliminated at each round. R and η are user defined hyper-parameters, with the former defining the maximum resources allocated to a single model between evaluations. The total number of brackets is defined by s_max.

- R is the maximum resources that can be allocated to a single model between evaluations
- S_{max} is the total number of rounds
- n_i is the current number of models, n_0 is the initial number
- η is the factor of models that are removed after each evaluation
- r_i is the amount of resources allocated to a model
- s is the current round from s_{max} to zero

$$n = \left\lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \right\rceil \tag{2}$$

$$s_{max} = [log_\eta(R)] \tag{3}$$

$$r = R\eta^{-s} \tag{4}$$

4.3 ASHA

Asynchronous Successive Halving Algorithm (ASHA) [27] is a more recent parallelisation technique for the Successive-Halving algorithm. This method has been shown to perform as well as or better than other techniques that incorporate early stopping such as PBT, BOHB and Hyperband on a variety of hyper-parameter optimisation and NAS benchmarks.

5 Bayesian Optimisation

Bayesian Optimisation (BO) methods for hyper-parameter optimisation have become popular over the last decade due to the SOTA performance they can produce [17] [5]. One of the core weaknesses of many BO implementations is the lack of scalability, with models such as Gaussian Processes computationally scaling cubically with observations.

The objective of BO and SMBO in general, in the context of hyper-parameter optimisation, can be described as trying to find $x^* = argmin\ f(x)$ where $x \in X$ and $X \subseteq \mathbf{R}^k$, X is a bounded and compact region and k is the dimensionality of the search space in our case the number of hyper-parameters. These methods assume correlation between observations and endeavour to use all of the available information to produce useful points for evaluating, x, by exploiting a model constructed based upon a set of function query, observations pairs $D = \{(x_n, y_n)\}_{n=1}^{N}$ where $y_n \sim \mathcal{N}(f(x), \sigma_n^2)$.

5.1 Posterior Model

A posterior model uses the observation history D often in combination with some prior to make estimation about the function f. These models generally model the value of f across the input space, either directly or indirectly, while also having some measure of uncertainty.

5.1.1 Gaussian Process

The most common model used in hyper-parameter optimisation is a Gaussian Process (GP)[8]. The Gaussian process is non-parametric model which is widely used due to its flexibility and simplicity, allowing many common acquisition functions to be described in a closed form, while having a well calibrated measure of uncertainty.

A GP can be considered a generalisation of a multivariate Gaussian distribution to an infinite number of variables. In our case this denotes all possible values within the bounded region X. A Gaussian process is fully defined, analogously to a Gaussian distribution, by a mean, $m(x)$ and covariance function $K(x, x')$, it is common to use $m(x) = 0$ as the Gaussian process is generally robust to the mean function given sufficient data.

$$f(x) \sim \mathcal{N}(m(x), K(x, x')) \tag{5}$$

$$f(x) \sim \mathcal{N}(0, K(x, x')) \tag{6}$$

The covariance function or kernel function is used to generate the covariance matrices in a GP. The kernel is responsible for how both the prior and posterior are expressed. The most commonly used kernel function is the *Squared Exponential Kernel*. The covariance between two points is a function of their separation and a hyper-parameter l.

$$K(x, x') = \sigma^2 exp\left(-\frac{|x - x'|^2}{2l^2}\right) \tag{7}$$

$$K(x, x') = \sigma^2 exp\left(\sum_{i=1}^{k} -\frac{|x_i - x_i'|^2}{2l_i^2}\right) \tag{8}$$

$$\Sigma(x, x') = K(x, x') + I\sigma_y \tag{9}$$

equation 8 shows the kernel function extended to multidimensional problem. for k dimensions there are k+3 hyper-parameters. l is the scale length, effectively how quickly the correlation between two points decays or the horizontal scaling for each dimension. σ is the or the vertical scaling. σ_y, shown in equation 9, is a representation of noise in the evaluations, this maintains some uncertainty around evaluation points. In hyper-parameter optimisation and other applications with noisy evaluations, Gaussian noise is added to the covariance matrix to avoid over-fitting. These hyper-parameters have a significant effect of the expression of the model and can be key to the final result, in particular l. Because of this, these settings are normally dealt with automatically rather than hand tuned. One approach used in [8] is to integrate over the hyper-parameters using Monte Carlo estimates, however this can be computationally expensive. Another common approach is to use a marginal likelihood estimates and optimise these settings via gradient descent.

$$p(f|\theta, D) = \mathcal{GP}(0, K(x, x')) \tag{10}$$

$$p(y|\theta, D) = \mathcal{GP}(0, K(x, x') + I\sigma_y) \tag{11}$$

equations 10 and 11 show a definition of Gaussian processes over an noiseless and noisy function respectively. For a point of interested, y(x'), the Gaussian process can be considered a joint distribution over the y(x') and the query observation pair history D. Using the marginalisation property of gaussians this can be restructured as equation 12. An estimate of y at x' is simply $y(x')$ conditioned on $y(\boldsymbol{x})$

$$p(y(\boldsymbol{x}), y(x')) \sim \mathcal{N}\left(\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}K(\boldsymbol{x}, \boldsymbol{x}) & K(\boldsymbol{x}, x')\\ K(\boldsymbol{x}, x')^T & K(x', x')\end{pmatrix}\right) \tag{12}$$

$$p(y(x')|y(\boldsymbol{x})) = \frac{p(\boldsymbol{x}, x')}{p(\boldsymbol{x})} \sim \mathcal{N}(K(\boldsymbol{x}, x')K(\boldsymbol{x}, \boldsymbol{x})^{-1}y(\boldsymbol{x}), K(x', x') - K(\boldsymbol{x}, x')K(\boldsymbol{x}, \boldsymbol{x})^{-1}K(\boldsymbol{x}, x')^T) \tag{13}$$

$$p(y(x')|y(\boldsymbol{x})) = \frac{p(\boldsymbol{x}, x')}{p(\boldsymbol{x})} \sim \mathcal{N}(\mu', \sigma') \tag{14}$$

$$\mu' = K(\boldsymbol{x}, x')K(\boldsymbol{x}, \boldsymbol{x})^{-1}y(\boldsymbol{x}) \tag{15}$$

$$\sigma' = K(x', x') - K(\boldsymbol{x}, x')K(\boldsymbol{x}, \boldsymbol{x})^{-1}K(\boldsymbol{x}, x')^T \tag{16}$$

Equation 13 shows the computational issue with Gaussian processes with the inversion of the posterior covariance matrix, $K(\boldsymbol{x}, \boldsymbol{x})$. This is a (N, N) matrix which means the computational complexity scales cubically $O(n^3)$ with the number of observations. This issues is further exacerbated in high dimensional search spaces causing GPs to become intractable on large high dimensional problems. Due to the expensive function evaluations in hyper-parameter optimisation, these computational expenses have, until recently, been easily justifiable. However, with the increasing ubiquity of parallel and cloud computing in ML, this limitation has lead to interest in other, more scalable models.

5.1.2 Tree-Structured Parzen Estimator

The Tree-Structured Parzen Estimator (TPE) is a model introduced by [5] based on Parzen Window Density or Kernel Density estimation. This method was also claimed to outperform both random and GP based Bayesian hyper-parameter optimisation on the MNIST data-set [5]. [17] has more recently used a similar method to great success, leading to its use as the underlying system in the popular Auto-ML tool "HyBandSter" [27]. TPE takes an alternative approach to the GP, modelling $p(x|y)$ and $p(y)$ rather than $p(y|x)$ directly. This approach makes use of Kernel Density Estimation (KDE) to build two distribution of hyper-parameter settings which is shown in Figure 3. equation 17 describes the criterion for discriminating between evaluations for division into either $l(x)$ or $g(x)$ in a maximization problem. Here $y*$ is set to be some quantile of the evaluations y rather than the maximum, this creates two probability distributions one of high performing hyper-parameter combinations and another of lower performing combinations.

$$p(x|y)\begin{cases} l(x), & \text{if } y > y* \\ g(x), & \text{if } y \leq y* \end{cases} \tag{17}$$

Query point acquisition which is discussed in section 5.2, is also simple with TPE. As shown in equation 18, the ratio of l(x) over g(x) is maximised, this was shown to be equivalent to maximizing EI [5] (Section 5.2.1).

$$x_{next} = argmax\frac{l(x)}{g(x)} \tag{18}$$

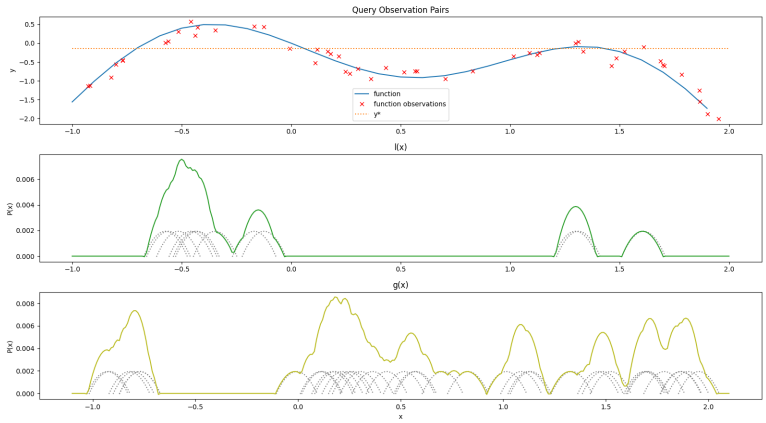

Figure 3: Example of the distributions created by KDE during TPE on a 1-dimensional maximisation problem. Top shows noisy observations of a function with y* set so that l(x) contains the upper quartile of evaluation scores. The middle and lower plots show KDE with a Gaussian kernel shown by the dotted lines.

5.1.3 Bayesian Neural Networks

Another method for modelling the distribution over a function is with the use of a Bayesian neural network. DNGO [12] is an implementation of this that uses a Deep Neural Network in combination with a Bayesian linear regressor to create an adaptive basis regression to model the posterior. This approach has the advantage of scaling linearly, in terms of computation, with the number of observations rather than cubically as with a GP. This method was able to achieve parity with SOTA in hyper-parameter optimisation on CIFAR-10. BOHAMIANN [13] is another implementation of Bayesian neural networks which was able to out perform DNGO on a number of hyper-parameter optimisation tasks while supporting native parallelisation.

One significant issue with NN based BO methods is the reliance of these techniques on well tuned hyper-parameters of their own to achieve optimal performance. This resulted in DNGO applying the more robust hyper-parameter optimisation system *spearmint* (based on [5]) to the system to optimise its own hyper-parameters.

5.2 Acquisition Function

The acquisition function is used in BO to select the next query point in hyper-parameter space at which to evaluate, x_{next}, based on the posterior model. This can be considered the criterion against which the latent value of a point in

the search space is evaluated. The exploit/explore problem is managed by this component of the system, balancing the use of known high value regions with information gain from areas with large uncertainty. Parallelisation is another important design feature of the acquisition function. Many popular and effective methods are natively sequential in nature and therefore require modification or extension to accommodate parallel query point acquisition. There are asynchronous implementations used as a solution to this problem, which commonly involve updating the posterior model as a worker completes an evaluation, producing a new query point from the updated posterior model and re-dispatching the worker. [25][15][5]

5.2.1 Expected Improvement

The most common of acquisition function, in particular when paired with a GP, is Expected Improvement (EI) due to the fact it is considered robust for most problems and does not require complex hyper-parameter tuning of its own. Equation 19 defines EI, which can be described as the expectation that a query point x will improve upon the current best evaluation in the next step, leading to a greedy choice of query points. Equation 20 shows how EI can be evaluated in closed form under a GP. From this the framing of the exploration/exploitation problem in terms of Δ_n (predicted mean - current best estimate) and predicted variance σ_n can be seen. EI is high when Δ_n is large and largest when both Δ_n and σ_n are high, with the former having a much larger effect. The exploration coefficient, ξ, is shown in equation 21. This hyper-parameter limits the greediness of the criterion by devaluing the posterior mean relative to uncertainty, promoting exploration.

$$EI_n(x) = \mathbf{E}\ max(f(x) - f(x*), 0) \tag{19}$$

$$EI_n(x) = [\Delta_n(x)]^+ \sigma_n(x)\varphi\left(\frac{\Delta_n(x)}{\sigma_n(x)}\right) - |\Delta_n(x)|\Phi\left(-\frac{|\Delta_n(x)|}{\sigma_n(x)}\right) \tag{20}$$

$$\Delta_n(x) = (\mu(x) - f(x*) - \xi) \tag{21}$$

Maximizing EI can be done with a number of approaches. One simplistic approach is simply to perform a grid search over the function. However, one of the beneficial attributes of EI under a GP is the often at least ones differentiable (conditional on the kernel) allowing for gradient descent to be used. it is important to note that EI tends to produce a function which is highly multi-modal, this means in practice gradient descent is restarted from a number of random locations in-order to find the global optimum. Figure 4 shows that even in simple problems EI can be multi-modal in nature.

5.2.2 EI-MCMC

EI-MCMC is an extension of this function which allows for parallelisation based on Monte Carlo Markov Chain estimates of the acquisition function [8]. In a

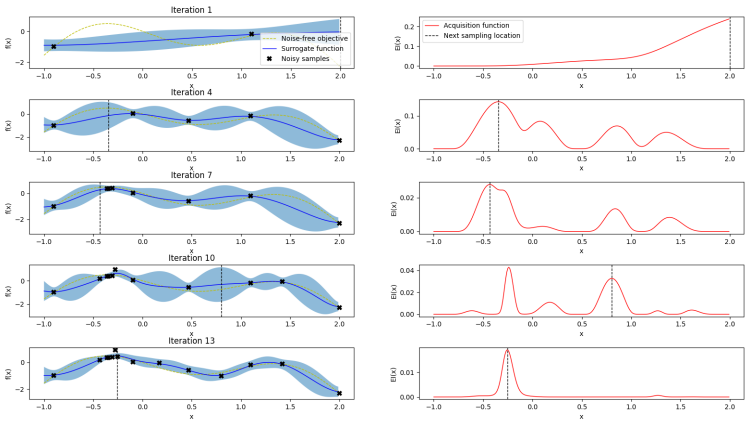

Figure 4: Expected improvement optimising a simple one-dimensional problem over a GP for 13 iterations. Left shows the Gaussian process modelling the posterior. Right shows the value curve at selected iterations under the EI criterion.

case where there are a set of J points being currently evaluated in parallel, EI cannot simply be re-evaluated on the same posterior as it would produce a redundant evaluation. This approach makes a Monty Carlo estimates via slice sampling at the ongoing evaluation points $\{x\}_{j=1}^{J}$ of the evaluation outcomes, $\{y\}_{j=1}^{J}$. One set of these outcomes is referred to as a 'fantasy'. Optimally, EI would be computed across all possible values for $\{y\}_{j=1}^{J}$ and the next point selected by integrating over the EI outcomes. In practice this is calculated over a finite number of fantasies with the x_{next} being the argument maximizing the result of integrating over the acquisition function outputs, described in 22.

$$x_{next} = argmax\ \widehat{EI}(x; D, \theta, \{x_j\}) \tag{22}$$

5.2.3 Upper Confidence Bound (UCB)

Upper Confidence Bound (UCB) is an approach to query point acquisition which is optimistic towards uncertainty. This, ideally, results in a system which is less greedy, and consequently less likely to become trapping in local optima. Simply attempting to maximizing the information gain at each iteration, a greedy approximation of which is given by 23, gives a criterion for quickly reducing the global uncertainty in the posterior. In practice however, this is not efficient as not only will it produce query points where $\mu(x)$ is known to be poor, but it fails to utilize function evaluations, $y(x)$, entirely. This observation is an important aspect of GP-BUCB, discussed in Section 5.2.4. Equation 24 describes GP-UCB [9], which rather than attempting to reduce across the entire function, attempts

to reduce uncertainty in the proximity of the maxima. B_n is a constant which is used to scale the dependence on the uncertainty. UCB more-so than other acquisition functions requires an accurately calibrated measure of uncertainty which encapsulates f entirely.

$$x_t = argmax\ \sigma_{t-1}(x) \tag{23}$$

$$UBC(x) = \mu(x) + B_n\sigma(x) \tag{24}$$

- figure
- guarantee of convergence more robust than EI

5.2.4 Batched Upper Confidence Bound (BUCB)

GP-BUCB is an extension to UCB introduced by [7] which allows for parallel query point acquisition. This approach builds on the observation from the previous section on equation 23 that the variance of the posterior model is independent of the function evaluation and dependant only on the evaluation location. This is shown in equation 16, however, this also highlights the computational bottleneck of this process as the posterior covariance matrix inversion is still required. In order to be feasible this required a surrogate method for calculating the variance. Another issue is over confidence of the posterior model when the variance has been updated with unevaluated queries, as UCB relies heavily on this uncertainty. One solution to this is to use highly conservative confidence bounds to ensure that the true function is captured.

- computational issues
- over confidence

5.2.5 Local Penalisation

One intuitive solution to this problem is the application of local penalization to an acquisition function in order to create batches of query points as shown in [10]. This approach iteratively applies a penalty to the acquisition function directly for each query point in a batch. The goal of this method is to produce a set of query points which explore separate modes of the acquisition function. Algorithm 3 describes this process, where $\varphi_j(x)$ is the local penalisation function and $\widetilde{a_j}(x)$ is the transform of the acquisition function.

Algorithm 3: Local Penalisation for BO

Data: observation history D = $\{(\mathrm{x}_n, y_n)\}_{n=1}^{N}, batchsize\ J$
Result: $\mathrm{B}_J = \{x_0, ..., x_J\}$

$\mathrm{x}_{next,0} = a(x)$
$\widetilde{a_0}(x) = \leftarrow \widetilde{a}(x)\varphi_0(x)$
for *j =1 to J* **do**
  $\widetilde{a_j}(x) \leftarrow \widetilde{a}(x)\varphi_j(x)$
  $\mathrm{x}_{next,j} = \widetilde{a_j}(x)$
end

However, this method can be consider as "doubly greedy" [11] in cases where a greedy acquisition function is used as the basis for local penalisation.

5.2.6 Thompson Sampling

Thompson Sampling (TS) is one method that has been applied to the problem of BO Parallelisation[25][15]. TS has performed well when compared with other parallel BO methods on a number of hyper-parameter optimisation tasks including the CIFAR-10 data-set. Thompson sampling is an approach to the exploitation/exploration problem in which an assumption is made about the value of a variable based on a sample from a probability distribution. In the context of BO and more specifically in application under a GP, TS involves sampling across the posterior distribution to create a sample function, g. The next query point can then simply be evaluated as $x_{next} = argmax\ g(x)$. Figure **??**, shows an example of how TS can produce diverse sampling locations in parallel. Thompson Sampling was shown in [15] to perform only marginally worse in parallel implementations, either synchronous or asynchronous, when compared with a sequential variation with the same number of total evaluations.

A similar approach which utilizes Thompson Sampling, AEGis, introduction by [25], combines this technique with periodic random selection of query points. This method was able to outperform other Parallel BO methods on synthetic function optimisation, however, it was unable to compete with vanilla TS on hyper-parameter optimisation tasks. When compared with TS, AEGiS also performed worse as the number of workers increased.

6 Reinforcement Learning

Short description of reinforcement learning Pros: Strong performance with high enough computational budget Incorporates meta-learning, moving away from a black-box perspective Cons: Very High computational cost

1. [50] Brought RL-NAS to the mainstream producing neural architectures that outperformed human designed SOTA on cifar-10, this required 800 gpu days or something idk

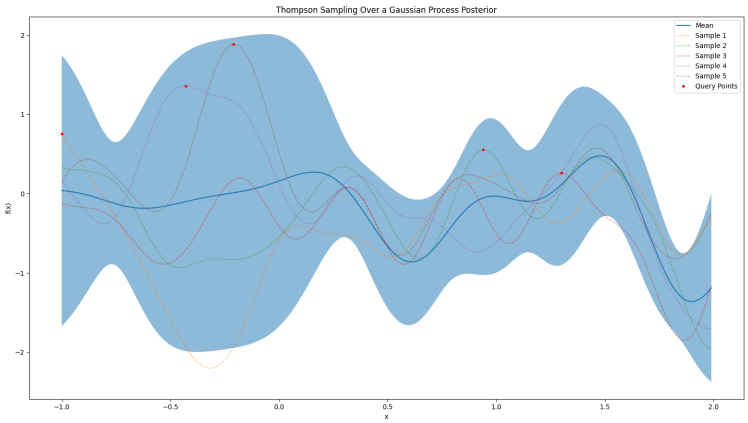

Figure 5: Synchronise query point acquisition for a batch of five points with Thompson Sampling under a GP.

6.1 One-shot Architecture Search

2. ENAS [52] was an expansion upon RL-NAS which was able to improve efficiency by over 1000x and achieve slightly improved results. This was done by training a single acyclic graph within which the entire search-space was contained. 3. [51] was able to show the importance of dropout in the one-shot method described in [52]

7 Bi-Level/Gradient Descent

[11] STNs - gradient descent for hyper-parameter optimisation
[53] Darts - gradient descent for cell architecture search
Networrks without training

8 Meta-Learning

Meta-learning has gained traction as a possible path away from the black-box perspective of hyper-parameter optimisation. This involves the application of machine learning to configuration selection. This has been implemented in various forms with success, such as a type of transfer learning in network embedding applications [23]. This has also been implemented as an interface which allows experts to effectively warm start an auto-ML pipeline with knowledge of effective configurations for similarly structured problems[28]. Another approach has been to train an agent via reinforcement learning to select configurations[28]. These applications of meta-learning are effective at reducing the convergence

time however, rarely achieve significantly superior final performance than the methods they are supplementing.[26][28]

“

9 Research Opportunities

one-shot/hypernetwork applied to GAs or BO

References

[1] P. J. B. Hancock, “Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification,” in *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, 1992, pp. 108–122. DOI: 10.1109/COGANN.1992.273944.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002. DOI: 10.1109/4235.996017.

[3] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[4] K. O. Stanley, D. B. D’Ambrosio, and J. Gauci, “A hypercube-based encoding for evolving large-scale neural networks,” *Artificial Life*, vol. 15, no. 2, pp. 185–212, 2009. DOI: 10.1162/artl.2009.15.2.15202.

[5] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyper-parameter optimization,” in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[6] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.

[7] T. Desautels, A. Krause, and J. Burdick, *Parallelizing exploration-exploitation tradeoffs with gaussian process bandit optimization*, 2012. arXiv: 1206.6402 [cs.LG].

[8] J. Snoek, H. Larochelle, and R. P. Adams, *Practical bayesian optimization of machine learning algorithms*, 2012. arXiv: 1206.2944 [stat.ML].

[9] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, “Information-theoretic regret bounds for gaussian process optimization in the bandit setting,” *IEEE Transactions on Information Theory*, vol. 58, no. 5, pp. 3250–3265, May 2012, ISSN: 1557-9654. DOI: 10.1109/tit.2011.2182033. [Online]. Available: http://dx.doi.org/10.1109/TIT.2011.2182033.

[10] J. González, Z. Dai, P. Hennig, and N. D. Lawrence, *Batch bayesian optimization via local penalization*, 2015. arXiv: 1505.08052 [stat.ML].

[11] A. Shah and Z. Ghahramani, *Parallel predictive entropy search for batch global optimization of expensive objective functions*, 2015. arXiv: 1511.07130 [cs.LG].

[12] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams, *Scalable bayesian optimization using deep neural networks*, 2015. arXiv: 1502.05700 [stat.ML].

[13] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter, "Bayesian optimization with robust bayesian neural networks," in *Advances in neural information processing systems*, 2016, pp. 4134–4142.

[14] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, *Population based training of neural networks*, 2017. arXiv: 1711.09846 [cs.LG].

[15] K. Kandasamy, A. Krishnamurthy, J. Schneider, and B. Poczos, *Asynchronous parallel bayesian optimisation via thompson sampling*, 2017. arXiv: 1705.09236 [stat.ML].

[16] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, *Evolving deep neural networks*, 2017. arXiv: 1703.00548 [cs.NE].

[17] S. Falkner, A. Klein, and F. Hutter, *Bohb: Robust and efficient hyperparameter optimization at scale*, 2018. arXiv: 1807.01774 [cs.LG].

[18] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, *Hierarchical representations for efficient architecture search*, 2018. arXiv: 1711.00436 [cs.LG].

[19] T. Elsken, J. H. Metzen, and F. Hutter, *Efficient multi-objective neural architecture search via lamarckian evolution*, 2019. arXiv: 1804.09081 [stat.ML].

[20] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, *Evolutionary neural automl for deep learning*, 2019. arXiv: 1902.06827 [cs.NE].

[21] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, *Nsga-net: Neural architecture search using multi-objective genetic algorithm*, 2019. arXiv: 1810.03522 [cs.CV].

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, *Regularized evolution for image classifier architecture search*, 2019. arXiv: 1802.01548 [cs.NE].

[23] K. Tu, J. Ma, P. Cui, J. Pei, and W. Zhu, "Autone: Hyperparameter optimization for massive network embedding," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 216–225.

[24] M. Wistuba, A. Rawat, and T. Pedapati, *A survey on neural architecture search*, 2019. arXiv: 1905.01392 [cs.LG].

[25] G. D. Ath, R. M. Everson, and J. E. Fieldsend, $Asynchronous\,\epsilon-greedybayesianoptimisation$, 2020. arXiv: 2010.07615 [cs.LG].

[26] Y. Huan, F. Wu, M. Basios, L. Kanthan, L. Li, and B. Xu, "Ieo: Intelligent evolutionary optimisation for hyperparameter tuning," *arXiv preprint arXiv:2009.06390*, 2020.

[27] K. H. Kouassi and D. Moodley, *Automatic deep learning for trend prediction in time series data*, 2020. arXiv: 2009.08510 [cs.LG].

[28] A. Souza, L. Nardi, L. B. Oliveira, K. Olukotun, M. Lindauer, and F. Hutter, *Prior-guided bayesian optimization*, 2020. arXiv: 2006.14608 [cs.LG].