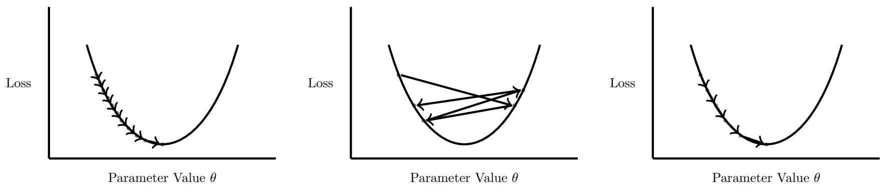
Literature Review

Christopher MacKinnon

November 2020

1 Hyperparameters

Hyper-parameter optimisation (HPO) is traditionally considered a black-box optimisation problem as the problem is generally analysed only in terms of inputs and outputs without consideration for the inner operation. Although the inner operation of Machine Learning (ML) systems can be observed, it is generally considered too complex and therefore functionally equivalent to a black-box. While some more recent approaches have attempted to move away from this black-box perspective [29], [38], this styling of the problem is still very prevalent. HPO problems can also be considered noisy in cases where the parameter weights within a network are initialised with random values, due to the variance in performance this can cause. This is common in practice to improve validation results [48]. Hyperparameters describe the settings of a model or algorithm. These "meta-parameters" define how a type of algorithm or model is expressed. Hyper-parameters can be continuous, discrete or categorical variables, requiring flexible approaches to find optimal solutions across a wide range of architectures and algorithms. The collection of hyperparameters for a particular algorithm, describes the hyperparameter space of a problem or model. This is the search space over which an optimisation algorithm is run. An intuitive example of a hyperparameter in the context of neural networks is learning rate. This is a parameter of many common optimisers and controls the amount by which a network parameter (i.e. the weight or bias of a connection between two nodes) changes by during one optimisation step. The value of this hyperparameter can significantly impact not only the convergence time but also overall model performance. Figure 1 shows the effects of learning rate over a simple single parameter example. In figure 1a an overly cautious value is applied, in this case the model does converge, however, the number of training steps required is very large resulting much higher computational cost. Figure 1b shows the opposite scenario with a highly aggressive value for the learning rate. In this case the model is unable to converge due to the large changes in model behaviour at each training step. Figure 1c shows the training loss in the ideal case. Here the model converges effectively without an excessive number of training iterations. While this trivial example shows the effects of a single hyperparameter, in practice modern machine learning systems can have tens or hundreds of hyperparameters. This results in high-dimensional search spaces which are both expensive to traverse and difficult for human experts to visualise. This has led to a drive for efficient automated systems which can produce high performance models without requiring extreme computational or man-hour costs.



(a) Overly Conservative Learning Rate (b) Overly Aggressive Learning Rate (c) Ideal Learning Rate

Figure 1: Effects of learning rate value on training loss over time.

A manual search combing a grid search and expert knowledge has been the standard approach to this problem in the past, with model designers relying on experience and intuition to guide the optimisation process. Grid search is a procedure that involves evaluating model performance at equidistant points over the search space. It was shown that a random search could perform as well as, or better than, grid search in the same time [7] and this type of search has become a minimum benchmark or starting point for many hyper-parameter optimisation techniques [24][11]. 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, with the SOTA in some application requiring thousands of GPU days of computation [27], [29], [40], [43].

There has been an increase in interest in Neural Architecture Search (NAS) since 2015 as these methods began to outperform human designed models [43][40]. NAS can be considered a subset of hyperparameter search which focuses on the networks topology, in particular in the context of Convolutional neural networks and Long Short-Term Memory networks (LSTM). These systems are able to design at scale creating intricate topologies for large, deep networks, while discovering novel variants in the components used to build these networks [26] [33]. In the context of computer vision this problem has been investigated significantly over the last four years, with a range approaches from genetic algorithms to reinforcement learning being applied to the problem. NAS systems often constrain the search space to a set of cells stacked in a predefined structure. These structures are based upon successful past implementation in these domains [16], [28], [43], [44]. The design and restriction of the of the search space over which a method implemented can have a very significant impact on the success of a search procedure. This generally involves contraining the search space to conform to a known succussful area or creating sub-modules which tend towards generating the repeating style of model structure which often successful in practice [33] [35].

2 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[4] [26] and can often produce more targeted, specialised architectures that deviate from the generalised templates that are 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 [26]. Due to tendency of evolutionary systems to become trapped in local optima, care must taken to maintain diversity in the population. This is a key problem faced by modern approaches to GAs to avoid early, sub-optimal convergence. 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 conversely, is a process which contracts the population based on some criterion. This is commonly some fitness evaluation on the objective function.

2.1 Selection

The selection process in a GA defines much of the character of the overall system. Truncation Selection, also known as 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. This process is shown in figure 2. While this method is common in real-world selective breeding, more sophisticated methods are generally used for GA applications in practice. Tournament selection is a less greedy approach to selection where the population is randomly sampled for a batch of members S. An evaluation is then done within this sub-set with an approach similar to elitist selection where the highest performing member of the sample is used to create an offspring which replaces the lowest performing. As the sample size grows so does the overall selection pressure of the process as samples are more likely to contain higher performing individuals. This system is described in Algorithm 1. A variant of tournament selection was used by [40], in which the population is selected against based upon the “age” of the network. This approach was used to set a new SOTA, performing at scale on ImageNET. While this method was able to achieve a highly impressive level of performance, the computation costs of this experiment were extremely high requiring thousands of GPU days of processing power. This method maintains a population of networks along with a list which retains the age of each network ranking each from oldest to youngest based on when they were added to the population, as shown in Figure 3. At each iteration a sample is draw from the population and the highest performing model is used to generate a new network. This new network becomes the youngest in the population and replaces the oldest network which is removed. This promotes exploration, reducing the likelihood of the system getting trapping local optima.

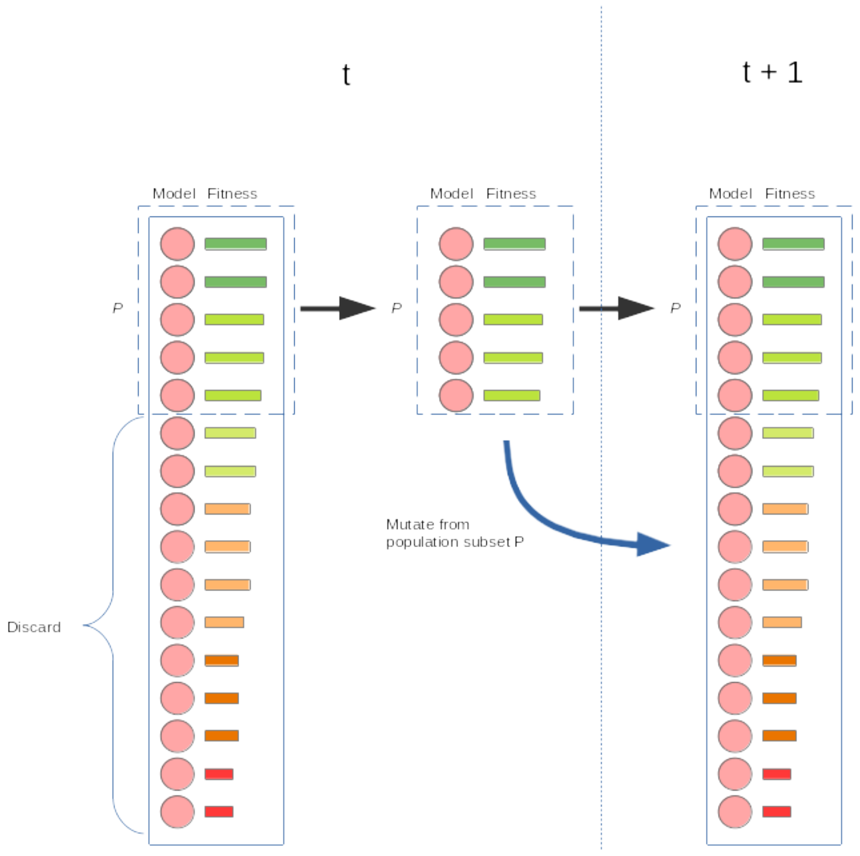


Figure 2: Example of one step of elitist selection applied to a population.

Algorithm 1: Tournament Selection Algorithm

```
Data: Samples size S, number of iterations C

for $i$ = $1$ to $C$ do
    while $\|sample\| < S$ do
        $sample$ ← random element from $population$;
    end
    $parent$ ← best evaluation score in $sample$;
    $offspring$ ← MUTATE($parent$);
    $population$ ← TRAINANDEVALUATE($offspring$);
    remove worst evaluation score in $sample$ from $population$;
end
Return $best\ evaluation\ score\ in\ population$;
```

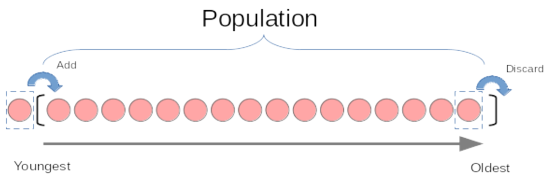


Figure 3: Ageing tournament selection. This method takes the same approach as tournament selection however, rather than discarding the lowest performing model in the sample, the oldest network on the right of the age array is discarded.

2.2 Speciation

NeuroEvolution of Augmenting Topologies (NEAT) [4] 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 (This is discussed in Section 2.4). While NEATs success is limited to smaller networks, however, there have been several extensions to NEAT, adapting it for larger, deeper networks [5][26]. One of the important contributions of NEAT was the use of Speciation. This method is used as a way to protect innovation in evolutionary systems by dividing the population into species which compete internally rather than with the population at large. This protects innovation by allowing new solutions to be explored and reach maturity without having to compete with older, more optimised networks. In practice this creates a less greedy algorithm supporting a greater diversity of solutions and, in theory, improving performance on highly multi-modal fitness functions. Models are divided into species based on genetic distance, δ. Equation 1 describes the process for finding the genetic distance. E and D refer to *Excess* and *Disjointed* genes respectively, which is a component of the NEAT encoding system which is explained in Section 2.6. These are connections which are not common to both of the networks being compared, with C_1 and C_2 being weight hyperparameters that can be used adjust the impact of these attributes. $\bar{w}$ denotes the difference in the parameter weights across common connections, with C_3 again a coefficient controlling the importance. $($N) is simply a normalisation factor equal to the number of genes in the larger network. Using this metric, networks are placed into species based on a defined compatibility threshold, δ_t. At each iteration, a network is chosen at random from each species to represent the genotype of that species. New networks are then sequentially added to the first species with which $\delta < \delta_t$, if there are no compatible species, a new species is created.

$$\delta = \frac{C_1 E}{N} + \frac{C_2 D}{N} + C_3 \bar{w} \tag{1}$$

The number of offspring allocated to a species is based on the shared fitness of the species with respect to the entire population. Equation 2 describes the adjusted fitness, the sum of which is fitness for a species. This restricts one species from dominating the entire population. Within a sub-population, elitist selection is then performed before repopulating the species.

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))} \tag{2}$$

$$sh(\delta(i,j)) \begin{cases} 1, & \text{if } \delta(i,j) < \delta_t \\ 0, & \text{else} \end{cases} \tag{3}$$

2.3 Multi-Objective Selection

In multi-objective optimisation rather than maximising the solution according to a single criterion, there are a set of criterion $f = \{f_i\}_{i=1}^{N}$, with respect to which the solution must be optimised. Often there is not a single solution which maximizes all objective functions f_i, this particularly true is cases where the f contains functions which are in conflict rather than complimentary. In this scenario, there are often a range of Pareto optimal solutions with different characteristics [2]. A naive example of this could be in the design of an electric car. Where the maximisation battery life and acceleration is of interest. These two criterion are in conflict with the increase in battery size decreasing the acceleration. There would therefore not be a solution in this case which maximizes both objectives simultaneously. A pareto set would therefore likely occur across the different combinations of these criteria. One example of conflicting objects which are both desirable in the context of neural networks would be model performance and complexity, which are a common set of criterion used in multi-objective approaches to HPO. While the desirability of performance is intrinsic to the problem domain and self evident, model simplicity also carries advantages. While there are more niche scenarios where this is a direct requirement, such as embedded systems were compute power is limited, more simple models are generally quickly to train reducing the computational load of the optimisation process overall. There is also a inherent benefit to solution diversity in promoting a range of models varying in complexity.

2.3.1 Pareto Efficiency

A system is said to be Pareto efficient when no improvement can be made to any of the criterion f_i without causing a deterioration in at least one other criterion. This leads to a collection of solutions across the objective function space which are Pareto optimal, these are known as a Pareto set or Pareto front and are said to be non-dominated. A solution is non-dominated when there is no solution which is better by one criterion and at least equal in all others. More formally this is given by equation 4 for a two dimensional problem. These non-dominated solutions collectively form the Pareto set for a collection of solutions. Figure 5 shows an example of a Pareto front in two dimensional space.

$$\begin{aligned}(f_1(x) \geq f_1(\{x\}_{i=1}^N) \ \boldsymbol{and} \ f_2(x) > f_2(\{x\}_{i=1}^N))\\ \boldsymbol{or}(f_1(x) > f_1(\{x\}_{i=1}^N) \ \boldsymbol{and} \ f_2(x) \geq f_2(\{x\}_{i=1}^N))\end{aligned} \tag{4}$$

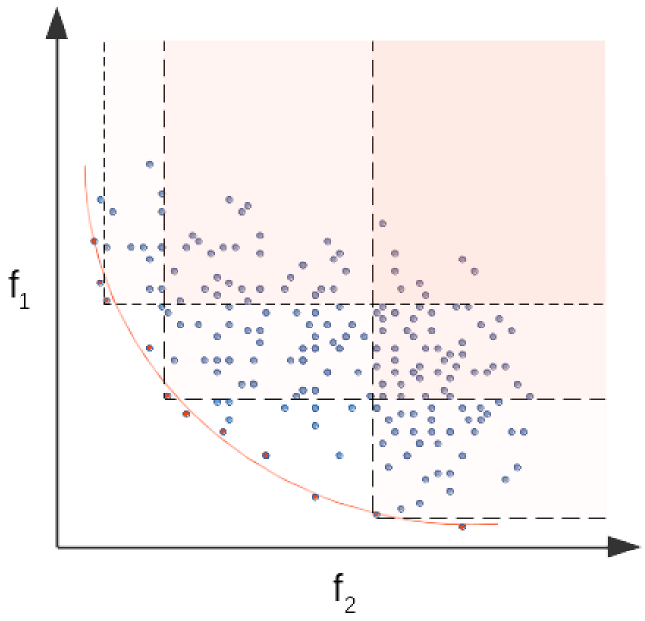


Figure 4: Two dimensional optimisation problem. The red line denotes the Pareto front of the problem, with the red dots showing non-dominated, Pareto optimal solutions. The black dotted lines and shaded area show the area of objective function space which is dominated by the solutions at the origin of the line, note that this is only shown for a subset of the non-dominated points for clarity

2.3.2 LEMONADE and NSGA-Net

As mentioned in Section 2.3, a common secondary objective function in NAS or hyper-parameter optimisation to combine with performance is model complexity. However, there are a number of metrics by which model complexity can be evaluated, such as, inference time, active network nodes, the number of model parameters or FLOPs (floating point calculations) in a forward pass of the network [39]. LEMONADE [36] is multi-objective system which uses the model parameters as a measurement of complexity and leverage's the cheap evaluation cost in the selection process.

This method uses Kernel Density Estimation (KDE) to build a distribution of networks with respect to complexity to select candidates for reproduction in sparse areas of the objective space (an example of KDE in a different application is shown in figure 7). The distribution generated by equation 5 is used to sample the initial set of offspring, where equation 6 is the normalisation constant. This process is then repeated sub-setting the offspring with equation 7, where $\hat{c}$ is again the normalisation constant similar to equation 6. it is important to note that N refers to the subset of solutions that make up the Pareto front of non-dominated solutions.

$$P_p(N) = \frac{c}{P_{KDE}(f_{cheap}(N))} \tag{5}$$

$$c = \left(\sum_{N \in P} \frac{1}{P_{KDE}(f_{cheap}(N))} \right)^{-1} \tag{6}$$

$$P_p(N^c) = \frac{\hat{c}}{P_{KDE}(f_{cheap}(N^c))} \tag{7}$$

NSGA-Net [39] is another multi-objective optimisation system for NAS which is built upon NSGA-II algorithm[3] for selection and incorporates Bayesian Optimisation methods. Rather than creating a distribution, NSGA-II iteratively allocates points into Pareto sets. This involves finding the initial set of non-dominated points assigning them to first layer of solutions and discounting them from the overall objective function space. This process is then repeated to fill out subsequent layers until there are no points remaining. These layers are then added to the next generation from the first layer down until the population limit is reached.

2.4 Genetic Encoding

Genetic encoding is a way of representing a network topology, parameters and hyper-parameters with the goal of allowing effective storage, crossover and transformation. Encoding methods can be broken down into two main categories, direct and indirect encoding. Indirect encoding refers to a system where rather than explicitly defining the structure of a network, a series of rules and structural motifs are defined from which the network is reconstructed. This allows for compact representations, often with repeating structures which has been shown to be successful in hand designed deep and convolutional networks [21] [18]. This type of encoding takes inspiration from biological systems with implementations based on composition pattern-producing networks (CPPNs) [19] having a foundation in embryonic development.[41] Direct encoding conversely, explicitly defines the attributes and topology of a system. This means the encoding (genotype) maps directly to the structure (phenotype).

The encoding scheme applied in NEAT is a direct encoding approach which maintains a list of connection and node genes. The connection gene contains the input node, output node, connection weight, innovation number and the enable bit. The node gene simply contains the innovation number and the type of node (i.e. input, output or hidden).

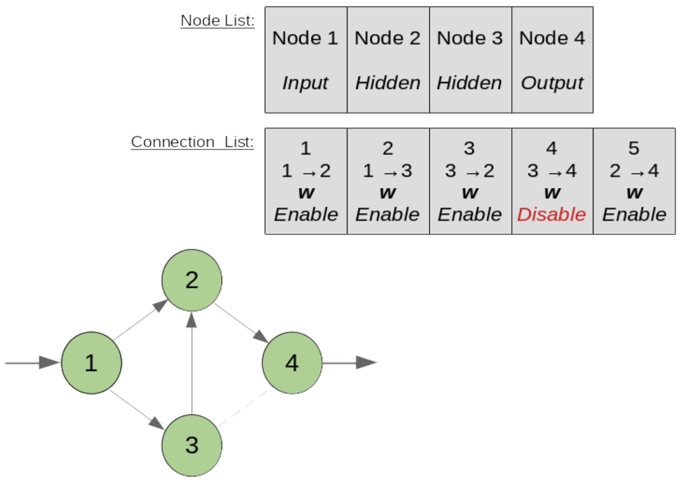


Figure 5: Example of the NEAT encoding scheme on a four node network. The connection list showing from top to bottom the innovation number, the input/output nodes, w denoting the weight, and the enable bit. In this example the connection from node 3 to 4 is disabled, while this connect with no longer be active in the network, it remains important as a historical marker for computing the genetic distance and performing crossover.

A significant issue that is important to consider when designing a genetic encoding methods, in particular with respect to crossover, is the problem of competing conventions or the permutation problem[1][4]. This occurs when there are multiple genomes which refer to the same phenotype. This can lead to loss of information, inefficient allocation of computational resources and irregularities during crossover operations.

NEAT makes use of an innovation number which is a core component to the overall NEAT system. The innovation number tracks the appearance of specific genes within the system. When a unique mutation occurs, it is allocated an innovation number, this number is incremented globally each time this occurs. Any subsequent occurrences of the same mutation are given the same innovation number. It is important to note that NEAT is not initialised randomly and rather all networks begin as a single connection between the input and output node, thus all networks effectively have a common ancestor and innovation numbers do not need to be assigned during initialisation.

2.5 Hierachical Model Design

CoDeepNEAT [26] is an expansion to NEAT for use with deep neural networks. CoDeepNEAT is based on a bi-level optimisation approach to NAS and hyper-parameter optimisation problems, implementing a hierarchical approach in which “blueprints” and modules are evolved independently. This method moves away from the neuroevoluationary approach of NEAT for learning and instead uses gradient descent to optimise the weights of the model.

A blueprint in this context is a graph of nodes which can be considered similar to the genotypes in NEAT. However, the nodes are replaced with smaller DNNs referred to as modules. These are combined together to produce a full network for evaluations. 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. These two components are evolved independently as separate populations. This method was also applied to LSTM networks, where it was able to produce an LSTM variant using skip connections, which outperform the standard LSTM cell.

Another similar approach was introduced in [33] used a nested system for NAS which is similar to the method of CoDeepNEAT and employs an N-level hierarchy rather than a bi-level hierarchy. In this system, primitive operations (i.e. convolutional cells, linear cells, etc) 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. Importantly, this method does not rebuild networks for each evaluation as is done in CoDeepNEAT, rather the networks are mutated in a more traditional process. The process was also able to achieve competitive results via a random search of this architecture space, implying that the representation and design of the search space may have played a major role in the methods success.

2.6 Crossover

Crossover is the recombination of two successful networks into a “child” network which is used in certain GA approaches to NAS. [37][4][39] This can be implemented through a wide variety of mechanisms based on the network encoding method which is used. Ablation tests on NEAT showed that while crossover can result in an overall improvement if carried out correctly, it was less significant that other aspects such as speciation [39][4]. Figure 6 shows an example of crossover in NEAT. In this process genes are classified as *Common*, *Disjointed* or *Excess*. Genes common to both parents are selected between randomly, whereas disjointed or excess genes are selected from the fittest parent. *Disjointed* refers to genes which are unique to one parent but occur before the most recent innovation in the other parent, this implies less shared history between the genotypes.

Excess genes are unique to one parent but occur after the last innovation in the other parent.

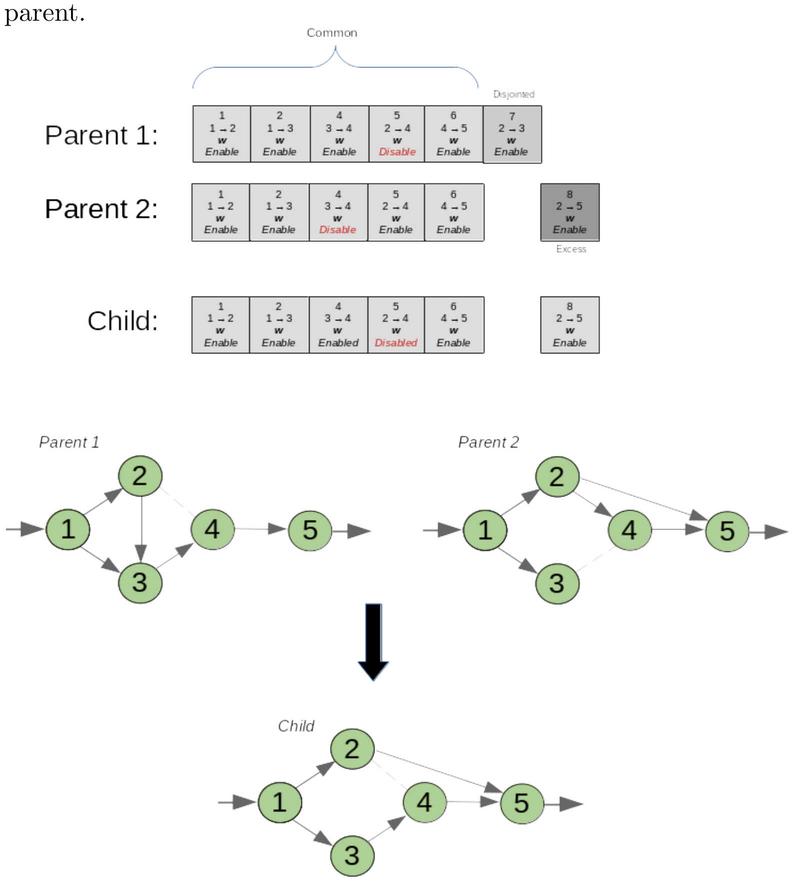


Figure 6: Example of the crossover in NEAT.

2.7 Mutation

Mutation is evolutionary systems is the process of perturbing or transforming some parameter or structure of a phenotype, generally without a pre-guided directive. This process must manage the balance between innovation, which is required to find new solutions and maintain diversity in the population, and preserving the internal functions and transforms of the model with which it achieved success. For numerical, continuous hyper-parameters a common approach is to permute the value according to a Gaussian distribution. For discrete variables it is common to employ bit-flipping [24], [26]. Topological mutation is generally implemented via a set of operations which can be applied to the network, although bit-flipping can be used in binary encoded implementations[39]. The NEAT style mutation operators involve adding either a connection or a node to the network. In this case a node is added by disabling the connection where the node is to be placed, adding the node and two connections to the genotype, the first new connection having a weight equal to one, the second having the weighting of the old, now disabled connection. While the implementation differ most architecture mutations follow a similar procedure, having a operation which randomly selects a node or connection, then removes, adds or changes the operation of that component. These system commonly employ a bank of operations from which a new operation can be selected [3], [33].

A similar approach which takes a slightly different direction is the use of *Network Morphisms* as described in LEMONADE. These are again transforms which are applied to the network, however, the goal of the operation is to deepen or wider the network while preserving the network function. A network morphism is a transform T, applied to a network N, which satisfies equation 8 for $x \in X$. This means that the network function *N(x)*, will remain intact after the transform is applied, maintaining the performance of the network and acting as a form of warm-starting. This allows for the topology of the network to be expanded non-destructively with respect to the network function. An example of a transform which conforms to this criterion be the addition of a layer to a network which is initialised to the identity function across all its parameter weights.

$$N^w(x) = (TN)^w(x) \tag{8}$$

This type of network transform has an obvious advantage for the efficient use of weight inheritance also know as Lamarckian Inheritance. This process has similarities with some neuroevolutionary approaches, as the network weights are maintained through the process of mutation and the generation of offspring. This be considered a form of warm-starting as networks are effectively pre-trained with their previous structure.

Another approach to weight inheritance is Population Based Training (PBT) introduced by [24]. This is a GA which takes inspiration from bandit approaches to hyper-parameter optimisation. 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[14] 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. This method of exploiting successful networks produces a schedule of hyper-parameters which change over the course of the training process rather than a static set. While this does offer the potential for a more dynamic training system, the size of the search space is increased dramatically as the number of hyper-parameter schedules is combinatorially larger than the original number of hyper-parameter sets.

3 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 [31] [6]. 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 Sequential Model-Based Optimisation (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, in contrast to a bandit style framing (Section 4) where each set of hyperparameters would be considered an independent variable. The system endeavours to use all of the available information to produce useful points for evaluating, x, by exploiting a model of the hyperparameter space constructed based upon a set of function query-observations pairs $D = \{(x_n, y_n)\}_{n=1}^N$ where $y_n \sim \mathcal{N}(f(x), \sigma_n^2)$.

3.1 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.

3.1.1 Gaussian Process

The most common model used in hyper-parameter optimisation is a Gaussian Process (GP)[9]. The Gaussian process is non-parametric model which is widely used due to its flexibility and simplicity, allowing many common acquisition functions (discussed in Section 3.2) 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 any finite 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 function, $m(x)$ and covariance function $K(x, x')$ shown in equation 9. It is common to use $m(x) = 0$ as the Gaussian process is generally robust to an arbitrary mean given sufficient data, which gives an equation for the GP as 10.

$$f(x) \sim \mathcal{N}(m(x), K(x, x')) \tag{9}$$

$$f(x) \sim \mathcal{N}(0, K(x, x')) \tag{10}$$

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*, given in equation 11. The covariance between two points is a function of their separation scaled by the hyper-parameter l.

$$K(x,x') = \sigma^2 exp\left(-\frac{|x-x'|^2}{2l^2}\right) \tag{11}$$

$$K(x,x') = \sigma^2 exp\left(\sum_{i=1}^{k} -\frac{|x_i-x'_i|^2}{2l_i^2}\right) \tag{12}$$

$$\Sigma(x,x') = K(x,x') + I\sigma_y \tag{13}$$

Equation 12 shows the kernel function extended to multidimensional problem. for k dimensions there are k+3 hyper-parameters. l is the scale length or the horizontal scaling, effectively how quickly the correlation between two points decays for each dimension. σ is the vertical scaling. σ_y, shown in equation 13, 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 often 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 [9] 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.

$$p(f|\theta, D) = \mathcal{GP}(0, K(x,x')) \tag{14}$$

$$p(y|\theta, D) = \mathcal{GP}(0, K(x,x') + I\sigma_y) \tag{15}$$

Equations 14 and 15 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 16. An estimate of y at *x'* is simply *y(x')* conditioned on *y(**x**)*, shown in equation 17

$$p(y(\boldsymbol{x}), y(x')) \sim \mathcal{N}\left(\begin{pmatrix}0\\0\end{pmatrix}\begin{pmatrix}K(\boldsymbol{x},\boldsymbol{x}) & K(\boldsymbol{x},x')\\ K(\boldsymbol{x},x')^T & K(x',x')\end{pmatrix}\right) \tag{16}$$

$$p(y(x')|y(\boldsymbol{x})) = \frac{p(\boldsymbol{x},x')}{p(\boldsymbol{x})} \sim \mathcal{N}(\mu', \sigma') \tag{17}$$

$$\mu' = K(\boldsymbol{x},x')K(\boldsymbol{x},\boldsymbol{x})^{-1}y(\boldsymbol{x}) \tag{18}$$

$$\sigma' = K(x',x') - K(\boldsymbol{x},x')K(\boldsymbol{x},\boldsymbol{x})^{-1}K(\boldsymbol{x},x')^T \tag{19}$$

Equations 18 and 19 show the computational issue with Gaussian processes, 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 scale, 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.

3.1.2 Tree-Structured Parzen Estimator

The Tree-Structured Parzen Estimator (TPE) is a model introduced by [6] 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 [6]. BOHB[31] 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” [47]. 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 7. Equation 20 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.

$$p(x|y) \begin{cases} l(x), & \text{if } y > y* \\ g(x), & \text{if } y \leq y* \end{cases} \tag{20}$$

Kernel Density estimation is a method for building a probability density function of a variable. This approach generates a function from variable samples, adding probability ”mass” at the location of each sample value. Equation 21 gives the density function of a set of samples n. K in this formula denotes the kernel. This describes how the probability mass allocated for each sample point. Figure 7 shows an example of this using an approximate Gaussian kernel. h is referred to as the bandwidth and scales the kernel distribution along the sample axis.

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K \left(\frac{x - x_i}{h} \right) \tag{21}$$

Query point acquisition which is discussed in section 3.2, is also simple with TPE, with equation 22 describing the process. The ratio of l(x) (higher performing models) over g(x) (lower performing models) is maximised for the argument x, this was shown to be equivalent to maximizing EI [6] (Section 3.2.1).

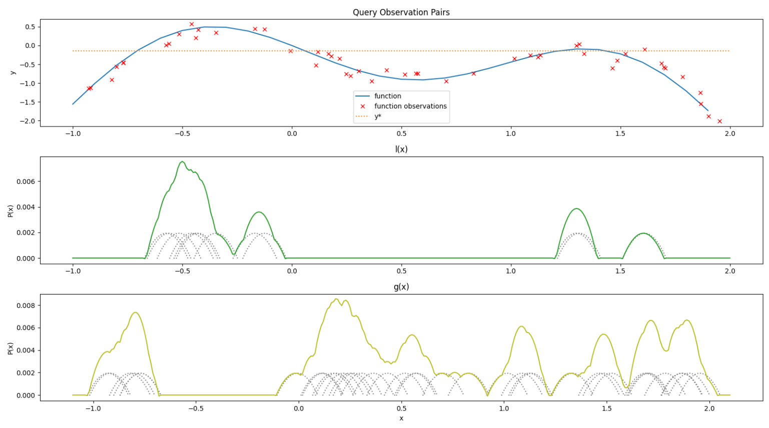


Figure 7: Example of the distributions created by KDE during TPE on a 1-dimensional maximisation problem. Top shows noisy observations of a function with y* set so that l(x) contains the upper quartile of evaluation scores. The middle and lower plots show KDE with a Gaussian kernel shown by the dotted lines.

$$x_{next} = argmax\frac{l(x)}{g(x)} \tag{22}$$

3.1.3 Bayesian Neural Networks

Another method for modelling the distribution over a function is with the use of a Bayesian neural network. DNGO [17] 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 [22] 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 neural network 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 [6]) to the system to optimise its own hyper-parameters.

3.2 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 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. [45][25][6]

3.2.1 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 23 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 24 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 25. This hyper-parameter limits the greediness of the criterion by devaluing the posterior mean relative to uncertainty, promoting exploration.

$$EI_n(x) = \mathbf{E}\ max(f(x) - f(x*), 0) \tag{23}$$

$$EI_n(x) = [\Delta_n(x)]^+ \sigma_n(x) \varphi\left(\frac{\Delta_n(x)}{\sigma_n(x)}\right) - |\Delta_n(x)| \Phi\left(-\frac{|\Delta_n(x)|}{\sigma_n(x)}\right) \tag{24}$$

$$\Delta_n(x) = (\mu(x) - f(x*) - \xi) \tag{25}$$

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 8 shows that even in simple problems EI can be multi-modal in nature.

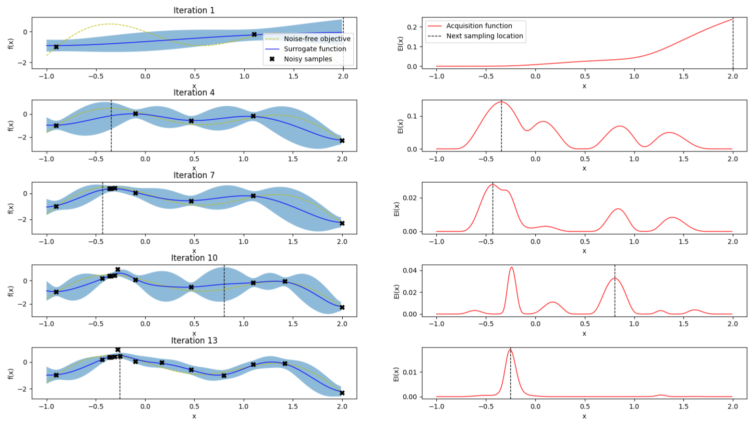


Figure 8: Expected improvement optimising a simple one-dimensional problem over a GP for 13 iterations. Left shows the Gaussian process modelling the posterior. Right shows the value curve at selected iterations under the EI criterion.

3.2.2 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 [9]. In a 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 26.

$$x_{next} = argmax\ \widehat{EI}(x; D, \theta, \{x_j\}) \tag{26}$$

3.2.3 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 trapped in local optima. Simply attempting to maximizing the information gain at each iteration, a greedy approximation of which is given by 27, 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 3.2.4. Equation 28 describes GP-UCB [10], 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 moreso than other acquisition functions requires an accurately calibrated measure of uncertainty which encapsulates f entirely.

$$x_t = argmax\ \sigma_{t-1}(x) \tag{27}$$

$$UBC(x) = \mu(x) + B_n \sigma(x) \tag{28}$$

3.2.4 Batched Upper Confidence Bound (BUCB)

GP-BUCB is an extension to UCB introduced by [8] which allows for parallel query point acquisition. This approach builds on the observation from the previous section on equation 27 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 19, 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 proposed solution to this is to use highly conservative confidence bounds to ensure that the true function is captured.[8]

3.2.5 Local Penalisation

One intuitive solution to the problem of parallelisation is the application of local penalization to an acquisition function in order to create batches of query points as shown in [13]. 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 2 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 2: Local Penalisation for BO

```
**Data:** observation history $D = \{(x_n, y_n)\}_{n=1}^{N}$, batch size  J
 **Result:** $B_J = \{x_0, ..., x_J\}$

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

However, this method can be consider as "doubly greedy" [15] in cases where a greedy acquisition function is used as the basis for local penalisation.

3.2.6 Thompson Sampling

Thompson Sampling (TS) is one method that has been applied to the problem of BO Parallelisation[45][25]. 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 9, shows an example of how TS can produce diverse sampling locations in parallel. Thompson Sampling was shown in [25] 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 [45], 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.

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

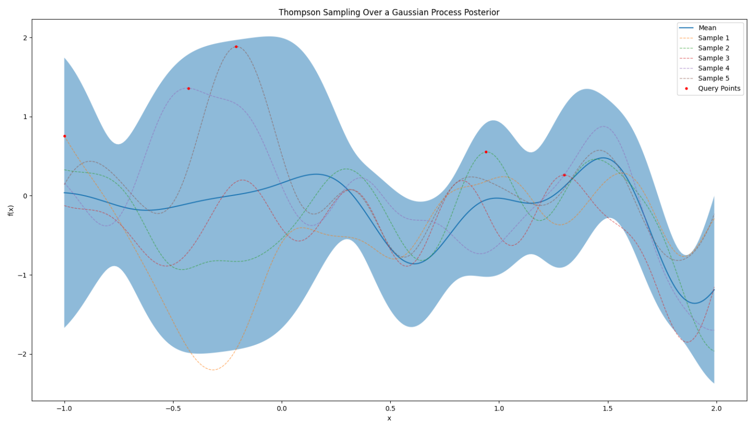


Figure 9: Synchronise query point acquisition for a batch of five points with Thompson Sampling under a GP.

with each action while also attempting to maximise the total reward. Figure 11 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. In particular HPO can be considered a infinitely armed bandit problem as in practice the number of possible actions (hyperparameter sets) is far larger than the possible number of models that can be trained to a meaningful level [12].

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 bandit problems it is common to measure the success of an approach based on the regret associated with the set of actions taken. This is what is known as cumulative regret and is the summation of the difference in utility or value between the set of actions taken and the optimal set actions. This approach however, is less useful in the context of HPO where it is simply the final hyper-parameter recommendation that is of interest. Therefore, what is known as simple regret, the difference between the highest utility taken action and the utility of the optimal action, is employed as the evaluation metric.

$$l_{\theta,t} = \frac{1}{|VAL|} \sum_{i \in VAL} loss(f_{\theta,t}(x_i), y_i) \tag{29}$$

Equation 29 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

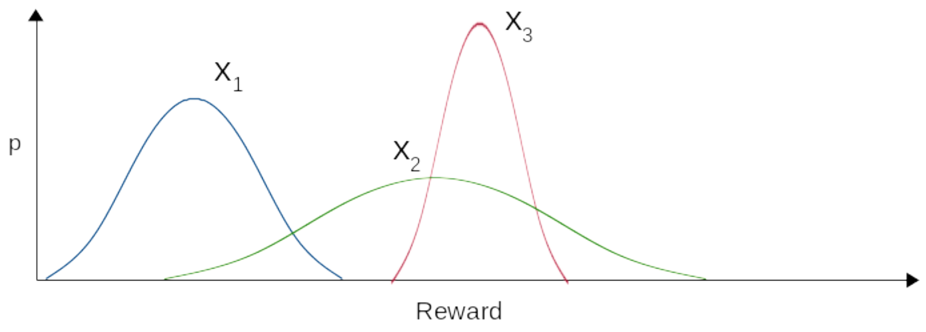
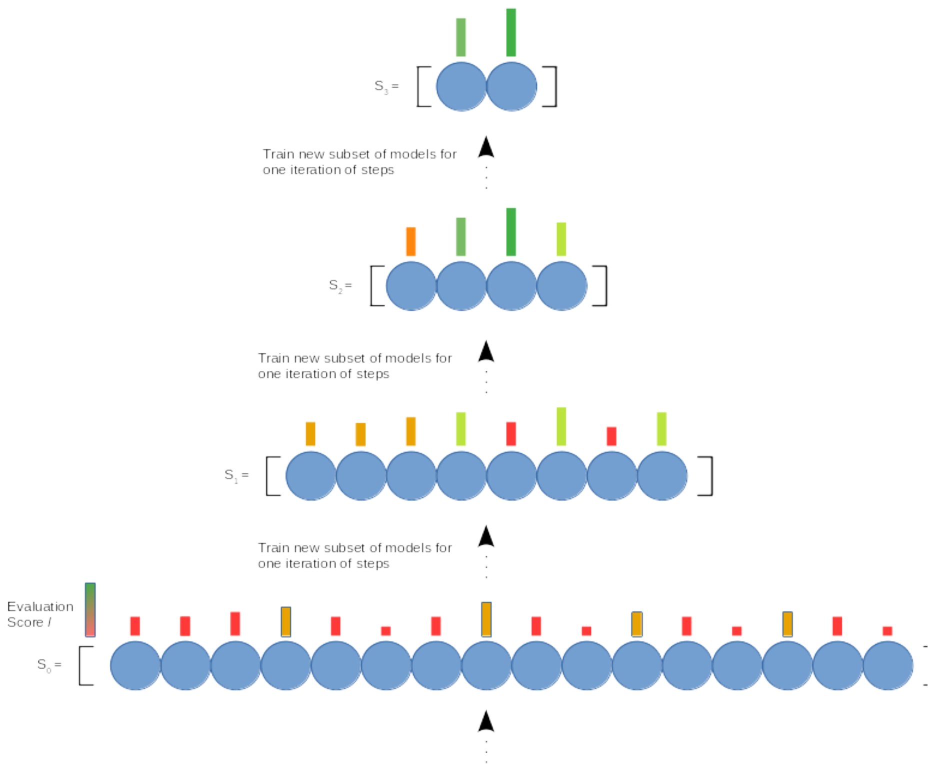


Figure 10: Bandit problem example with three actions and associated reward distributions denoted as x_1,x_2 and x_3.

θ

4.1 Successive Halving

Successive Halving is an algorithm first introduced in [11] for stochastic bandit problems and later adapted for the non-stochastic domain of hyper-parameter optimisation [14]. 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 29. The set of S_0 is then subset to half its size based on this metric. Figure 11 shows an a series of iterations of Successive Halving which is more formally described in Algorithm 3.



S_3 =
Train new subset of models for one iteration of steps
S_2 =
Train new subset of models for one iteration of steps
S_1 =
Train new subset of models for one iteration of steps
Evaluation Score l
S_0 =

Figure 11: Four steps of the Successive Halving algorithm.

Algorithm 3: 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
        $TrainS_{k,j}\ for\ \frac{B}{|S_k| log_2(n)} steps$;
        $l_k = \{l_k, \frac{1}{|VAL|}\sum_{i \in VAL} loss(S_{j,k}(x_i), y_i)\}$ ;
    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. [24], [31], [47] Asynchronous Successive Halving Algorithm (ASHA) [47] 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.

4.2 HyperBand

HyperBand [32] 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 [14], 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 4: HyperBand

```
Data: R, $\eta$

Initialise: $s_{max} = [log_\eta(R)], B = (s_{max} + 1)R$, Model evaluation
  scores $l = \{\}$
for $s = \{s_{max}, s_{max} - 1, ..., 0\}$ do
    $n = \left[\frac{B}{R}\frac{\eta^s}{(s+1)}\right]$;
    $r = R\eta^{-s}$;
    $S_0 \leftarrow$ randomly_generate_hyperparameter_sets($n$);
    $l \leftarrow SucessiveHalving * (S_0, r, \eta)$;
end
Return max(l)
```

Algorithm 4 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 the factor by which the number of models is reduced by 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}.

5 Reinforcement Learning

Reinforcement learning is an approach to NAS which became popular with the introduction of RL-NAS [29] which utilised a controller agent trained via reinforcement learning to generate model architectures for image classification and natural language processing problems. This approach was able to set a new state of the art in CIFAR-10, outperforming human hand-designed systems. Figure 12 shows an example of the controller network. While this approach was able to achieve very strong performance, the computational budget required was extremely large, requiring 22,400 GPU days of training.

The extreme computational cost of RL-NAS lead to the development of Efficient Neural Architecture Search (ENAS). ENAS [34] was an expansion upon RL-NAS which was able to reduced the computational cost by a factor of more than 1000 and achieve slightly improved results setting a new SOTA on CIFAR-10. This was done by training a single acyclic graph within which the entire search-space was contained. This type of approach is now referred to as One-Shot NAS. This type of weight sharing can be considered an aggressive form of transfer learning. The key assumption here is that the performance of a network using a set of weights generated on a *supernetwork* will correlate with true performance of the network. The importance of drop-out in the training of the supernetwork was shown in [30]. This method has also seen success recently with what is known as Few-Shot NAS [50]. In this approach the search space is split into a number of sub-*supernetworks*. This allows the for a middle ground between One-Shot NAS and RL-NAS.

Another approach to this problem is to generate the network weights through

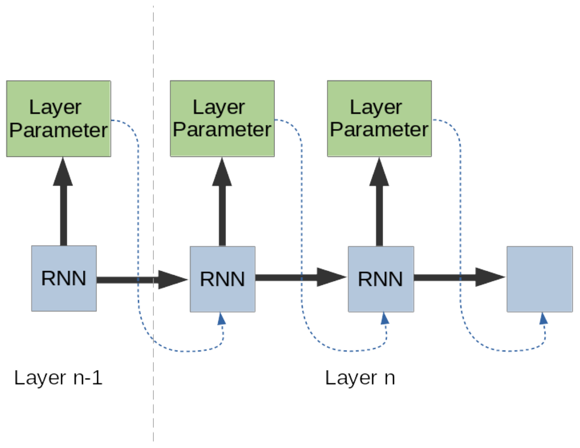


Figure 12: RNN controller in RL-NAS. The controller is a recurrent neural network (RNN), which at each time-step outputs a parameter for the current layer (i.e. number of node, filter size, dropout, ect). The number and order of these parameters is regular and the process simply repeats for each layer.

the use of a *HyperNetwork* [20]. The concept of a *HyperNetwork* involves training a smaller network to generate the weights of a larger network. In particular this has interesting applications in RNNs where the weights can be generated dynamically at each time-step. SMASH [23] is an application of this method to the NAS problem which was able to achieve some success.

6 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 [42]. 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[49]. 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.[46][49]

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. Miettinen, *Nonlinear Multiobjective Optimization*, ser. International Series in Operations Research & Management Science. Springer US, 1999, ISBN: 9780792382782. [Online]. Available: https://books.google.co.uk/books?id=ha%5C_zLdNtXSMC.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

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

[10] 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, Apr. 2012, ISSN: 1557-9654. DOI: 10.1109/tit.2011.2182033. [Online]. Available: http://dx.doi.org/10.1109/TIT.2011.2182033.

[11] Z. Karnin, T. Koren, and O. Somekh, *Almost optimal exploration in multi-armed bandits*, 2013.

[12] A. Carpentier and M. Valko, *Simple regret for infinitely many armed bandits*, 2015. arXiv: 1505.04627 [cs.LG].

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

[14] K. Jamieson and A. Talwalkar, *Non-stochastic best arm identification and hyperparameter optimization*, 2015. arXiv: 1502.07943 [cs.LG].

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

[16] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[17] 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].

[18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, 2015. arXiv: 1512.00567 [cs.CV].

[19] ——, *Rethinking the inception architecture for computer vision*, 2015. arXiv: 1512.00567 [cs.CV].

[20] D. Ha, A. Dai, and Q. V. Le, *Hypernetworks*, 2016. arXiv: 1609.09106 [cs.LG].

[21] K. He, X. Zhang, S. Ren, and J. Sun, *Identity mappings in deep residual networks*, 2016. arXiv: 1603.05027 [cs.CV].

[22] 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.

[23] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, *Smash: One-shot model architecture search through hypernetworks*, 2017. arXiv: 1708.05344 [cs.LG].

[24] 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].

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

[26] 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].

[27] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, *Large-scale evolution of image classifiers*, 2017. arXiv: 1703.01041 [cs.NE].

[28] S. Zagoruyko and N. Komodakis, *Wide residual networks*, 2017. arXiv: 1605.07146 [cs.CV].

[29] B. Zoph and Q. V. Le, *Neural architecture search with reinforcement learning*, 2017. arXiv: 1611.01578 [cs.LG].

[30] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *International Conference on Machine Learning*, 2018, pp. 550–559.

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

[32] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, *Hyperband: A novel bandit-based approach to hyperparameter optimization*, 2018. arXiv: 1603.06560 [cs.LG].

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

[34] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, *Efficient neural architecture search via parameter sharing*, 2018. arXiv: 1802.03268 [cs.LG].

[35] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, *Learning transferable architectures for scalable image recognition*, 2018. arXiv: 1707.07012 [cs.CV].

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

[37] 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].

[38] H. Liu, K. Simonyan, and Y. Yang, *Darts: Differentiable architecture search*, 2019. arXiv: 1806.09055 [cs.LG].

[39] 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].

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

[41] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[42] 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.

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

[44] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter, *Nas-bench-101: Towards reproducible neural architecture search*, 2019. arXiv: 1902.09635 [cs.LG].

[45] G. D. Ath, R. M. Everson, and J. E. Fieldsend, *Asynchronous epsilon greedy bayesian optimisation*, 2020. arXiv: 2010.07615 [cs.LG].

[46] 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.

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

[48] L. Li and A. Talwalkar, *Random search and reproducibility for neural architecture search*, R. P. Adams and V. Gogate, Eds., Tel Aviv, Israel, 22–25 Jul 2020. [Online]. Available: http://proceedings.mlr.press/v115/li20c.html.

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

[50] Y. Zhao, L. Wang, Y. Tian, R. Fonseca, and T. Guo, *Few-shot neural architecture search*, 2020. arXiv: 2006.06863 [cs.LG].