Multi-Objective Neuro-Evolution with a focus on maximising input data efficiency

Shane Acton
Computer Science
University of Cape Town
Cape Town South Africa
shaneacton745.sa@gmail.com

Abstract

Many Deep Learning problem domains are restrictive in terms of availability of labeled data. It is thus crucial to offer solutions to allow for better training on limited data. Regularisation and Data Augmentation (DA) can help in this respect, but the number of different ways they can be combined and applied is immense. Neuro-Evolution (NE) has proven to be a good black box optimiser, and could thus be a suitable candidate for finding optimal combinations of these methods. Using NE to optimise Artificial Neural Networks topologies as well as Regularisation and DA schemes with the objectives of maximising accuracy and minimising dependance on large training datasets could help to alleviate problems caused by limited data availability.

1 Introduction

Recently Deep Learning (DL) has proved to be one of the more powerful ML techniques available, accomplishing many incredible feats. Specifically in the field of computer vision Deep Convolutional Neural Networks commonly referred to just as CNN's have become the state of the art across many domains starting with image classification [24]. Creating DL solutions however comes with its own set of challenges, namely availableness of quality Data; accessibility to large computing power; and model selection and tuning [D]. Of these computing power is the simplest requirement, it is obvious that computing with the maximum amount of resources at your disposal is the best solution. The more complicated issues come in with Data availability and model tuning.

Since access to data is often limited to what is publically available, many problem domains can suffer due to lack of access. Large corporations may have the human resources to manually generate and label data, but it cannot be overstated how expensive and laborious this process can be in practice. An adult human who had never seen a cat or dog before could learn the difference between the two with potentially as little as a single example, leveraging embedded knowledge about the world and using superior computing power and neural architectures. To date even using concepts like transfer learning [E] which is able to leverage predefined knowledge about the way objects look, it would still take at best hundreds of examples of dogs and cats to teach a DL model the difference. Clearly solutions need to be explored which make networks more efficient with the amount of input data they need to correctly generalise concepts. We will mention some briefly, and put a specific focus on Data Augmentation.

DL solutions can sometimes take long periods of time to train- even on powerful computers companies can dedicate weeks to training to production quality. There is however one process which can eclipse training in terms of its impact on the time it takes to perfect a model - hyper parameter tuning. This is the process of iteratively experimenting with hyper-parameters such as the topology of the network; the training speed; the activation functions used; the types of layers; ect. While extensive experience and understanding of Deep Artificial Neural Networks (Deep ANN's) can help to speed up hyper parameter tuning - but due to the black box nature of ANN's it can still take months to get this crucial step right [A]. It is the tedious and time consuming nature of this process which has motivated the study of the field of AutoML.

AutoML is using a process which given a well defined problem will aim to automate the process of hyper parameter tuning, be it by searching the hyper parameter space or by using embedded knowledge to try solve the problem heuristically. Although this introduces the trade off between computing intensity and human labor - in certain situations it can still save significant time [B]. We will be focusing on the branch of AutoML which aims to search hyperparameter space for optimal solutions. One of the most promising approaches to this is Neuro-Evolution - which leverages evolutionary algorithms to effectively search and evaluate the hyperparameter space.

2 Deep Learning with small data

In the last few years DL has solved some of the most difficult problems in AI, and has beaten the best approaches of older more heuristic based methods in almost every area of ML imaginable from computer vision and natural language processing to game agents and protein folding [C]. This is mostly due to DLs ability to leverage huge amounts of data and computing power by being highly scalable.

DL can help to solve what were once incredibly difficult tasks given sufficient data and computing power. Luckily computing power is not going anywhere, and while some problems faced today would require more computing power than we can reasonably offer, for example global climate modeling - many problems can be solved satisfactorily given sufficient available computing resources. So if the problem

faced when trying to create a DL solution is computational power, a simple solution is to acquire more, or even just wait longer for training. With data availability however, certain problems which are architecturally and computationally solvable can be crippled by a lack of quality labeled data and short of hiring farms of humans to manually label massive sets of data - there might not be much which could be done to remedy the issue.

Aside from reinforcement learning which leverages computational power to generate its own training data via interactions with a simulated environment - most forms of supervised and unsupervised DL are vulnerable to a lack of available data be it labeled or unlabeled. It is thus important to try and create solutions which can offer results with as little data as possible. To this end there are many studied approaches which can aid in increasing the data efficiency of trained networks - which is how well a network can be trained on limited datasets.

2.1 Regularisation

One approach to improving data efficiency is called regularisation, which in its essence is about constraining the network at train time. Overfitting occurs when the network being trained has enough complexity to fit around the noise of the data it is shown. A less complex network can overfit less. Regularisation is a set of approaches which all aim to limit the complexity of the network being trained either in its weights, or in the information that flows through it. Some examples of regularisation include weight constraining which bound weight values to a range; activity regularisation which penalises the network for large node activation values; dropout which stochastically loses a proportion of the information in a layer before passing it onto the next, thereby reducing the interdependence of neurons[6]; and batch normalisation which tries to limit the variation of the magnitudes of intermediary layer outputs - this is to limit a factor called covariate shift, which is at a high level the issue arising from the fact that later layers are dependant on earlier layers, but earlier layer weights change throughout training, so the dependencies learned by the later layers waste time having to constantly retrain on the changing earlier network[7]. Regularisation leads to faster generalisation which in some cases could allow networks to train on fewer data points - however the effect on data efficiency is limited, and not as well studied as its effects on training time given a fixed dataset size.

2.2 Data augmentation

Much of creating and training neural network classifiers involves balancing overfitting and underfitting of the model to the training set. The perfect classifier learns to generalise to understand the underlying differences between the classification classes, and disregards all of the noise and overly specific information in the training set. The obvious problem then is knowing what parts of the training data are useful and what parts are not - this is not trivial. The most obvious and arguably the most effective solution is to increase the size of the training set since the general concepts will pervade a training set of any size, but patterns of useless information will only exist in subsets - and so the larger the training set the better the network can distinguish between useful general patterns and useless specificities and noise.

Consider the problem of identifying a cat. Given a small set of examples where each cat picture has a blue sky in the background for example, the trained network may grow to believe that a blue sky is an integral part of being a cat and fail to recognise cats which are indoors or pictures of cats taken at night. It is clear here that increasing the training set size to include more cats some of which are in different background settings will force the network to disregard its hypothesis and edge it one step closer to generalising to the most abstract concept that includes all cats, and excludes all non cats.

Unfortunately increasing the training set size is not always possible, one may be working with limited data, or have a corpus of unlabeled data that must be manually annotated. In any event, other solutions need to be explored to make networks more efficient with the information they use. There are a handful of well established methods which aid in generalisation and can theoretically allow networks to generalise on fewer data points.

Data Augmentation(DA) adds a new dimension to data efficiency aside from regularisation. DA aims to increase the utility of each training sample by using a concept called label preserving transformations. A label preserving transformation is any data transformation which does not change the class of the data point [8]. So for example when training on an image of a cat, one may rotate, flip, crop, add noise, colour inverse, ect and still expect that human viewers would consider the result to be a cat. This is what is meant by label preserving - a transformation the result of which should still be classified as the same class were it an unseen data point. Should a transformation change the input data so much that one would not like the result to be considered the same class as its input - this transformation would not be desirable as a data augmentation scheme, as you would train this undesirable association to the class into the network.

In the case of image classification there are many label preserving transformations, some of which were mentioned in the above paragraph. Each one of these should add to the effectiveness of a classifier if used correctly [8]. To use a data augmenter your network should pick a subset of the training data to be augmented, pass these data points through the transformation, label these new data points the same as the input which was used to generate them, and finally train the network on both the augmented and unaugmented (datapoint, label) tuples. One can either augment before training or during. Augmenting before more is memory intensive as the new data points must be stored in memory along with the original; and augmenting during training adds an extra time overhead as each augmentation must be performed multiple times, but saves on memory as the augmented data can be disregarded once used in a training cycle [9].

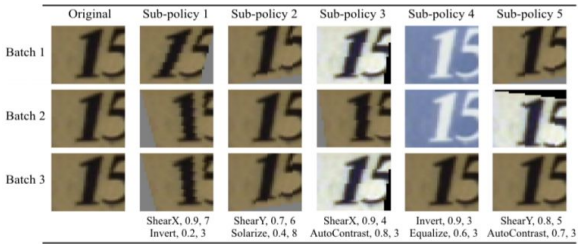

Figure 1: results of Google's AutoAugment algorithm for DA [13]

Figure 2: 10 Different Style transfer filters used as a DA scheme [10]

Image Data Augmentation can be divided up into two main categories - namely geometric and photometric. Geometric DA's alter the geometry of the image, this is to say that any label preserving transformation that reshapes, rotates, mirrors, crops, or applies any other affine transformation to the image is a geometric DA. The benefits of this are an increased ability to generalise the underlying model to be invariant to these changes [8]. Consider a training set where all items of a class share an orientation, this is clearly a limitation as the model trained may become dependant on unseen items having this same orientation. In this case applying various geometric orientation augmentations to the training data can ensure the network is able to recognise the class regardless of orientation. The second type of image DA is photometric, which is any label preserving transformation applied to the pixels. Some photometric augmentations include adding noise, hue changes, colour flipping, and style transfer. The power of DA is making your model invariant under the transformations used in your DA scheme, and so for example adding noise as a DA can help to make your classifier robust in the face of noisy test data [8].

Clearly the space of photometric transformations is much larger than geometric, as it allows for any conceivable pixel transformation, however it is also more complex and generally more computationally expensive. There have been papers comparing typical geometric and photometric transformations DA schemes - in paper [8], it was found that the set of used geometric DA's were not only computationally cheaper but also more effective than the photometric DA's used. This however does not rule out the usefulness of photometric methods as there is evidence to suggest these results are more due to the limitations of the photometric transformations used. In paper [13] google demonstrates that combining these methods in intelligent ways can yield incredible results. Using DA similar to that seen in figure 1 Google was able to beat state of the art results in the most competitive image classification benchmarks such as ImageNet and CIFAR-100.

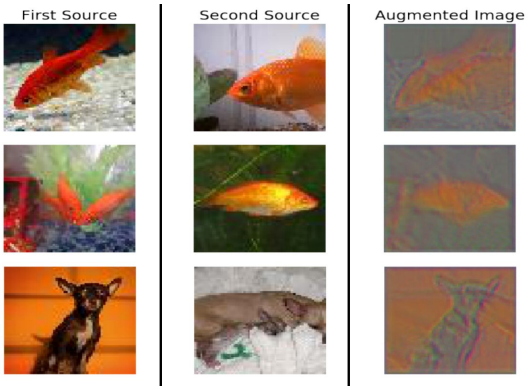

Figure 3 : results of CNN combiner as a DA scheme [9]

2.2.1 Style Transfer as Data Augmentation

In a paper investigating claims that deep CNN based image classifiers overly rely on textures, ignoring object shapes to a large degree. It was found that using a method known as style transfer as a data augmentation scheme forced classifiers to learn more rich and general understandings of the shapes of the objects [10]. The paper demonstrated that this highly abstract and conceptually complex photometric transformation was effective, adding to the idea that there is still much we can learn about photometric transformations as DA schemes. Style transfer is the concept of extracting the style of one image or set of images, and applying it to the content of a target image. Figure 2 demonstrates an input image with a set of style transferred outputs. The process rests on the idea that in an abstract sense any image can be separated into content and style, and that if these two components were isolatable, you should be able to re-express any given content in any given style. Luckily CNN neural networks are very good at abstraction, and so there has been remarkable success in image style transfer in the last few years [11]. As was mentioned earlier - data augmentation makes a classifier robust in the face of the type of variation that the augmentation scheme offers. Knowing this it is very interesting to think that these novel CNN transformations could offer incredible generalisation abilities, and could possibly enable training on smaller sets of data.

2.2.2 Image interpolation as Data Augmentation

Another photometric transformation which is highly interesting is image interpolation. An ideal image interpolator is one which can take two images A and B, and any real number between 0 and 1 call it α, and produce realistic looking images which visually resemble weighted (αA + (1-α)B) interpolation of the features such that every value for α looks acceptable. With respect to DA, using an interpolator to interpolate between two images of the same class should create novel images of the same class under the right conditions but it is unclear if these novel images would be useful for training classifiers. There has been good progress in nonlinear CNN based image interpolators recently, [12,14] being two of many approaches. There appears to have been no

public attempt at using complex CNN based interpolation schemes specifically as DA to date, however there has been a paper which aimed to use CNN combined pairs of training set images as a DA scheme. Paper [9] attempted, among other things, to train a combiner CNN which takes in two training set images of the same class and produces one image such that the new image created was used for training. This combiner CNN was trained to maximise its effectiveness as a DA scheme, but on inspection it appears it converged on silhouetting the two input images and superimposing them on top of eachother, the results were mixed between experiments and the method was not deemed a success and performed worse than traditional DA methods. A sample of results is shown above in figure 3. The Authors of the paper acknowledge the limitations of their combiner CNN and as future work recommend using better suited architectures.

3 Neuro-Evolution of Augmenting Topologies (NEAT)

Neuro-Evolution of Augmenting Topologies or NEAT is a Topology and Weight Evolving Artificial Neural Network (TWEANN) approach - meaning it simultaneously searches a neural network's weight space and tries to find an optimal topology for the graph itself, or in other words - it searches the networks topology space. This is done by evolving the number of neurons or nodes as well as the connections between these nodes - then optimising the weights of the connections on each iteration of the graph [1].

NEAT speciates or groups networks based on topology to protect newly formed topologies with unoptimised weights from being eliminated by older already optimised topologies. Instead topologies or species compete within themselves to optimise their weights before being compared to the rest of the population of networks This allows new topologies to compete fairly, having controlled for weight optimisation [1].

NEAT improved on older Neuro-Evolution methods by growing topologies from minimal graphs so that only necessary additional complexity gets added, and since less complex graphs have their weights optimised - significantly fewer generations are needed to converge [1].

3.1 DeepNEAT

As the name suggests deep NEAT is an extension to NEAT made to create deep neural networks. The principal difference is that the nodes in the chromosomes no longer represent neurons as in NEAT but instead represent layers of neurons, typically either fully connected, recurrent or convolutional. The node contains only the hyper parameters of the layer - importantly not the weights. This is because DeepNEAT, unlike NEAT, does not optimise the weight space of the graphs it creates, instead it is simply concerned with the network topology and its hyper-parameters - included in which are options such as activation function.

The weights in deepNEAT are optimised by backpropagation not evolutionary search, this offers speed up for training as

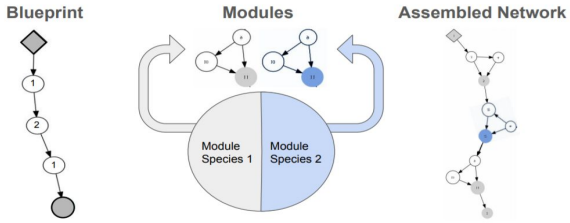

Figure 4: A diagram depicting the process of parsing a Blueprint into a module graph [3]

weights are changed in a deterministic and useful manner as opposed to randomly as in weight space neuroevolution, meaning fewer weight vectors need to be evaluated for fitness. Evaluation is done by training the given topology for a fixed number of epochs using classic supervised training.

In deepNEAT the graph being evolved is no longer directly a neural network - connections between nodes are not weighted, instead only indicate how layers pass outputs to each other. Instead the graph is an abstracted neural network, and as such must be parsed at each training interval into a real neural network.

Deep NEAT has been criticised as it produces complex and unstructured final graphs which contrasts to most of the successful DL architectures which have been developed over the last few years [2].

3.2 CoDeepNeat

CoDeepNEAT was developed to combat the problems which arise in DeepNeat. It was designed to allow for modularity in the graphs it generates, as it was observed that successful commercial and academic DL models often make use of repeated structures.

To achieve modularity CoDeepNEAT co-evolves two distinct populations, namely a population of Blueprints and of Modules. Modules are simple graphs which are akin to the graphs evolved in DeepNEAT where each node in the graph represents a layer of neurons, as such these modules can be parsed into neural networks. Blueprints are the novelty added by CoDeepNEAT. Blueprints are graphs where each node is a pointer to a species of module. Each Blueprint graph can be parsed into a graph of connected modules (shown in figure 4) which in turn can be parsed into a functional neural network. Since multiple nodes in a Blueprint can point to the same module - a module can be repeated multiple times in the final neural network - and only be evolved once, this is the co-evolution of modules and blueprints.

As in DeepNEAT - CoDeepNEAT speciates modules. To begin with, a random set of simple modules is initiated. Then they are clustered together into species based on a similarity metric, from then onwards, the aggregate fitness of each species is used to determine how large of a population that species can maintain - so more fit species of modules are allowed to grow to greater proportions of the total population.

The variation inside of the species of modules is leveraged by Blueprints - when it is time to parse a Blueprint into a graph of modules, each node in the blueprint uses its pointer to a module species population to select one specific module from that species - with the caveat that once a given blueprint has selected a module from some species, if another node in that blueprint points to the same species then instead of random selection - the same exact module is used. In other words, no single blueprint instance can contain more than one distinct module from the same species.

Due to the random selection from module species - any given Blueprint can be parsed into many distinct module graphs, and therefore many distinct final neural networks. Because of this Blueprints do not need to be speciated, instead the set of all possible instances of a single Blueprint is akin to a species population.

To evaluate fitness of a Blueprint or Module inside of a given generation, the average fitness of every final neural network which makes use of that Blueprint or Module is used [2].

4 Multi-Objective Optimisation

At their core evolutionary algorithms are optimisers which search a parameter space. They seek to find solutions to a problem which are deemed as having the highest fitness, in other words - they maximise fitness. In simple cases fitness can be some measure of performance such as the accuracy of a neural network classifier, or the speed of an evolutionary agent solving a maze. In more complex cases however one may want to find solutions which satisfy more than one independent criterion. This is where multi-objective evolution comes into play.

4.1 Multi-Objective Optimisation Approaches

4.1.1 Plain Aggregation

Consider having an implementation constraint on a neural network which meant that you needed to find solutions which used as little memory as possible - ie: as few connections in the graph as possible. A naive solution would be to measure accuracy and measure graph connections and combine the two into one fitness measurement to be maximised, for example Fitness = Accuracy/Connections. This is akin to converting your multiple objectives into a single objective, which while still guaranteed to generate a solution, may not satisfactorily search your parameter space [4]. This is known as the plain aggregation approach as has the problem of requiring the user to weight the importance of each objective which can be difficult. It is also limited by this manual weighting in that it poorly explores the trade off space between the objectives due to the fixed nature of the importance of each respective objective[5].

4.1.2 Population-based Multi-Objective Optimisation

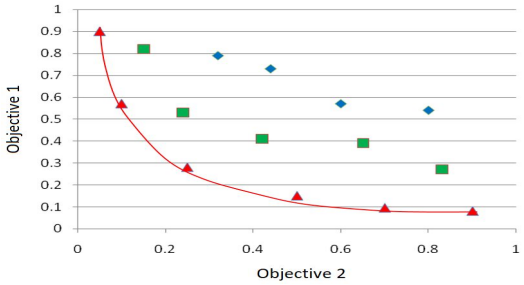

Figure 5: Three Fronts, red being the Pareto Front

Another approach is to use what is known as a population based method. In population based multi-objective optimisation (MOO) separate populations are evolved for each objective, and then at each generation - the populations are combined together genetically [5]. These solutions suffer from their own downfalls due to the limitations of the optimisation being used. In a hypothetical case where the solution properties required to satisfy two objectives simultaneously is disjoint from the sets of properties found to satisfy each objective in isolation - the method would fail as learning how to solve the objectives individually is not useful for solving the multi-objective problem (MOP).

4.1.3 Pareto-based Multi-Objective Optimisation

Currently the most promising approach to multi-objective optimisation is to use a Pareto-based algorithm in the selection process of your genetic algorithm. Under the Pareto-based approaches you evolve just one population as in plain aggregation. There is a notable exception however, in that Pareto-based methods produce a set of solutions which are all considered to be globally equivalently fit, but may differ in their fitness along any one local objective metric - that is to say that the set of Pareto solutions aims to be the set of those solutions which are all globally optimal, and distinct enough from each other that they fully explore the trade-offs between optimising each one of the individual objectives [5].

Pareto based approaches are named as such because they use what is known as the Pareto Front as a tool to ensure a good trade off between exploration and exploitation. They aim to ensure that the solution space is explored fully without converging on a single trade-off ratio between the objectives - as is often the case with non Pareto-based methods. To do this the method requires a function or metric for each objective, these functions are called objective functions or objective metrics - which are in essence quantitative values describing how well a given solution is satisfying the respective objective. These metrics are related to but distinct from fitness since there is no assumed way of weighing them against each other. Instead of dimensionally reducing the objective metrics to a single fitness value - solutions are sorted into Fronts as seen in figure 5 then inside of these fronts solutions are sorted by some implementation specific criterion which generally tries to ensure diverse sets of solutions.

To explain what a Pareto Front of a set is, it is first necessary to explain the concept of domination in the context of MOP's. If a solution is referred to as being dominated by another - it means that there exists a second solution which is more optimal than it for every objective metric under consideration. Conversely a solution being non-dominated simply means that no other single solution is more optimal along every objective measurement simultaneously - but there still may be solutions which are more optimal on a subset of the objectives. Knowing this - the Pareto Front is simply the set of all solutions which are non-dominated [5]. Take each of the red triangle points in figure 5 which together form the Pareto Front - note that when the two axis are normalised - the manhattan distances of all the points on the Pareto Front from the origin are very similar, and that if there were another solution with a significantly smaller manhattan distance from the origin - it would be dominating the red triangles, and thus the red triangles would no longer be a Pareto Front. So by definition - the points on the Pareto Front are an optimal subset of the set of solutions being considered.

Since the points on the Pareto Front are an approximation of the optimal trade-offs between the objectives - having a set of solutions which is disperse along the Front means that you have both found optimal solutions and you have explored the trade off space in a meaningful way [5]. This is the goal of Pareto-based selection, exactly how this is done is up to the implementation of the specific algorithm, and will be discussed in the next section

4.2 Multi-Objective Neuro-Evolution Implementations

4.2.1 Non-Dominated Sorting Genetic Algorithm (NSGA)

NSGA is one of the most popular multi-objective search algorithms used today. Beginning with NSGA in 1995 and being built on with NSGA-II and NSGA-III, many of the desired properties of MOO can be provided with these algorithms. Its success is largely due to it using a powerful non-dominated sorting algorithm, which is simply a sorting algorithm which sorts tuples based on the Pareto dominance principle.

4.2.1.1 NSGA

NSGA-I was one of a few MOO approaches developed in the mid-1990's which focused on two important criterion, namely: 1) using non-dominated sorting to select the fittest members of a population and 2) encouraging diversity of solutions produced on the Pareto front [15] Satisfying these two properties meant NSGA was successful as a multi-objective optimiser. However a number of technical problems were raised by researchers who attempted to use it. The largest problems were in essence efficiency problems, as the nondominated sorting algorithm used had a computational complexity of O(MN^3) where M is the number of objectives and N is the population size[15]. Furthermore at the time of its development there had been little work in incorporating Elitism into Multi-Objective Genetic Algorithms.

Elitism at a high level is the property of a genetic algorithm whereby it passes on the fittest members of a parent generation to the subsequent child generation. This is useful in that it ensures that the benchmark set by the parent generation is upheld in the next generation, without it - it would not be uncommon for children generations to underperform compared to their parent generations, especially when the genetic algorithm was getting close to converging on optimal solutions. It is well known that Elitism helps genetic algorithms converge faster for single-objective optimisation[15,16]. Paper [17] which investigates multiple factors which can affect the success of MOO, found that Elitism played a large role in the success and convergence speed of various MOO algorithms. It is thus a major shortfall of NSGA-I that it cannot support Elitism.

4.2.1.2 NSGA-II

In the widely cited paper [15], NSGA-II is introduced to combat some of the shortcomings of NSGA-I. The paper claims NSGA-II eliminates the need for a sharing variable, which was used in NSGA for ensuring population diversity and created a host of issues, leading some researchers to develop dynamically adjusting sharing values, which only in part fixed the problem. Another impressive improvement is reducing the computational complexity of nondominated sorting from O(MN^3) in NSGA-I to O(MN^2) in NSGA-II.

The authors of NSGA-II decided to implement elitism into the algorithm. Some of the previous attempts they considered included Zitzler and Thiele's [18] approach of maintaining a separate population of all non-dominated solutions found along the generational iteration. At each generation, the current population is combined with this separate non-dominated population. This combined population is then used for all genetic operations from evaluation to reproduction. After each generation, the non-dominated population is updated from the previous combined population. This non-dominated population acts as the elite group, as if no better solutions are found in a new generation the optimal solutions from the previous generations will be carried forward.

Paper [19] implemented a similar albeit more complex version of Zitzler and Thiele's [18] approach whereby parent solutions are compared via domination. If a parent or child was dominated by the other - the non dominated one would be carried forward into the next generation. In the case where neither parent or child dominates the other the decision as to which one to bring forward into the next generation goes to a stored archive of solutions which is similar to the non-dominated population from Zitzler and Thiele's [18] approach. If the child dominates any member of the archive - it replaces the dominated solution in the archive, and is chosen over the parent to be brought into the next generation. Finally if the offspring does not dominate any member of the archive, a crowding mechanism is used to decide whether the child or parent is in a less crowded

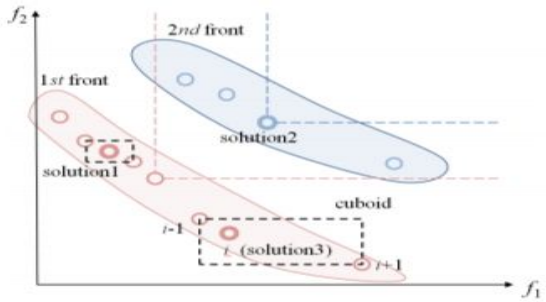

Figure 6: Ranking using non-dominated and crowding distance sorting [4]

area of the archive in terms of the objective metric space. The less crowded between the two is carried forward and is also added to the archive.

The non-dominated sorting solution chosen by the creators of the NSGA-II [15] algorithm was to separate solutions into fronts as seen in figure 6 and to consider fitness to be firstly which front the solution is in, so fronts closer to the Pareto front are prioritised over fronts further away. Solutions inside of the same front were assigned fitness based on a crowding metric. A density estimation is performed on each solution point, this estimation is obtained by going through each objective metric axis and averaging the distances between the nearest two neighbors of the solution being estimated on that axis. The nearest-two-neighbours values for each objective metric are combined to give each solution an estimate for how crowded the area it inhabits in the objective metric space is. Less crowded solutions inside the same front are considered more fit [15]. This is the mechanism used to ensure exploration of the pareto-optimal space, in contrast to the criticised method of sharing used in NSGA-I.

Elitism is introduced in NSGA-II by applying selection to the combined parent and offspring populations at each generation. More formally, at each step t, parent population Pt and child population Qt are combined to create population Rt which has a size of 2n where n is the population size of parent and child generations, such that Rt = Pt $\cup$ Qt. The non-dominated sorting described above is then used on population Rt. And finally the next parent generation P(t+1), which will be used to create Q(t+1), is created by taking the n highest priority solutions in Rt. Non-dominated solutions from Pt which are also non-dominated in Qt will therefore be non-dominated in Rt, thus globally non-dominated solutions can carry forward from Pt to P(t+1) [15]. This solution is elegant in that it does not require a separate population to be maintained, and instead needs only the parent and child populations to ensure an Elitism property is present.

4.2.1.3 NSGA-III

Proposed in paper [21] NSGA-III was developed as an extension to NSGA-II. NSGA-II and other prominent MOO's have been critised for scaling poorly with the number of objectives, thus a new problem class was defined called many-objective optimisation, which is defined as an

Algorithm 3 Multiobj CoDeepNEAT Pareto Front Calculation

1. Given list of individuals I, and corresponding objective fitnesses X and Y
2. Sort I in descending order by first objective fitnesses X
3. Create new Pareto front PF with first individual I_0
4. For each individual I_i, $i > 0$
  5. If Y_i is greater than the Y_j, where I_j is last individual in PF
    6. Append I_i to PF
7. Sort PF in descending order by second objective Y (Optional)

Figure 7: A summary of the pareto front selection algorithm used for MOO in CoDeepNEAT [3]

Algorithm 2 Multiobj CoDeepNEAT Module/Blueprint Ranking

1. Given population of modules/blueprints, evaluated primary and secondary objectives (X and Y)
2. For each species S_i during every generation:
  3. Create new empty species $\hat{S}_i$
  4. While S_i is not empty
    5. Determine Pareto front of S_i by passing X_i and Y_i to Alg. 3
    6. Remove individuals in Pareto front of S_i and add to $\hat{S}_i$
  7. Replace S_i with $\hat{S}_i$
  8. Truncate $\hat{S}_i$ by removing the last fraction F_l
  8. Generate new individuals using mutation/crossover
  9. Add new individuals to $\hat{S}_i$, proceed as normal

Figure 8: A summary of the non-dominated sorting algorithm used for prioritising solutions for MOO in CoDeepNEAT [3]

optimisation problem with four or more objectives, as opposed to multi-objective which is just more than one. Paper [20] demonstrates a case where NSGA-III is superior over NSGA-II with three objectives, and paper [21] furthers this by showing NSGA-III outperforms its predecessor on objective numbers ranging from three to fifteen. Paper [22] however casts doubt on the universal superiority of NSGA-III and suggests that for certain problems NSGA-II is preferable, even in the class of many-objective optimisation.

4.2.2 Multi-Objective Optimisation used in Lamarckian Evolutionary algorithm for Multi-Objective Neural Architecture DEsign (LEMONADE)

The platform named LEMONADE is a full neuro-evolutionary genetic algorithm specification designed for resource efficiency. Lemonade requires the user to specify which metrics require computationally expensive evaluations and which only require cheap evaluation. It leverages this information by evaluating populations based on the cheap evaluations, creating a subset of the highest performers, then evaluating only these cheap-evaluation-high-performers using the expensive methods. This is to minimise the number of expensive evaluations done throughout the evolutionary process. This has the side effect of treating cheap evaluation objectives as primary objectives, and since accuracy is often an expensive metric as it requires training an ANN, it is treated as a secondary objective which can result in a trade-off between shortened training time and lowered accuracy in some cases[23].

The 'Lamarckian Evolution' is in reference to the evolutionary theory that children inherit properties which are

influenced by the lifetime experiences of their parents. In LEMONADE the mutations are chosen such that the weights of the trained and evaluated parent ANN's are passed down to their children who may have different topologies. To ensure the children topologies are compatible such that they can accept the weighting values of their parents, special mutation operators are applied that the authors call 'network morphism operators'. These inherited weights provide the new networks with a so called 'warm start' allowing them to start with a performance similar to that of their trained parents and then train for shorter periods to improve. This process is referred to as Lamarckian Weight inheritance. Combined with the factors mentioned in the previous paragraph, Lamarckian Weight inheritance allows for significantly faster convergence that other Neuro-Evolutionary approaches [23].

4.2.3 Multi-Objective Optimisation used in CoDeepNEAT

CoDeepNEAT uses a simple algorithm for its non-dominated sorting. As shown above in figure 8 the algorithm requires the specification of a primary and secondary objective, the authors of the 2019 CoDeepNEAT paper [3] suggest that the primary objective would typically be the performance of the ANN being generated. By definition of domination the solution in a set of solutions which has the most optimal value in the primary, or any, objective is non-dominated. This is why I0 (figure 7) can be assumed to be non-dominated and added to PF, as it has the most optimal primary objective metric. From this starting point the remaining solutions are iterated over and each of the solutions which are also in the Pareto front are added to PF. As is also seen in NSGA-II, the primary sorting criterion in CoDeepNEAT is which front a solution belongs to, however as shown in the algorithm described by figure 7 and figure 8 the secondary sorting criterion is simply the ordering of the primary objective metric results as opposed to a crowding based sort. Thus while the algorithm successfully ensures solutions will converge onto the pareto front, there is no crowding or other mechanism which ensures solution diversity, so it is not ensured that final solutions will be well distributed along the front.

Elitism is introduced in CoDeepNEAT by simply ensuring a fixed proportion of the most fit (according to CoDeepNEAT's non-dominated sorting) is propagated to the next generation. In CoDeepNEAT this process is done species-wise as seen in figure 8, this means that a proportion of the most fit of each species is propagated forward without being mutated, and the remaining solutions in the next generation are generated via genetic crossover and mutation.

5 Conclusion

Image classification is widely used as a benchmarking tool for DL. This is due to many reasons including the intuitiveness of the classification to humans, the fact that it has historically been a challenging field of computing and the availability of image data. Its industry applicability is also large ranging from self driving cars to automatically scanning documents. Due to the applicability of DA schemes which naturally fit into image classification we will be framing our analysis in trying to maximise the accuracy of image classifiers, as well as using MOO to achieve other ANN related goals.

Many approaches for altering ANN's to generalise with less data have been shown to be successful. The variety in these approaches leads to its own problem for ANN designers, which is knowing which of these approaches to use and where. Designers can use experience to narrow down the choices but are often still left with the task of manually tuning these hyper parameters over many test iterations.

AutoML approaches such as Neuro-Evolution can help to automate away the process of hyper parameter tuning. CoDeepNEAT has shown to be able to create state of the art DL architectures, but doesn't encourage solution diversity. NSGA-II describes a non-dominated sorting algorithm which does not require a primary objective and can ensure diversity, which may work well as a replacement for the unnamed non-dominated sorting algorithm used in CoDeepNEAT.

Previous approaches have included regularisation schemes in their evolvable hyper parameter set [2,3]. Allowing DA schemes to be evolved alongside topologies and regularisation schemes may help answer questions about the limits of DA in its effect on data efficiency. Allowing the evolution of arbitrary combinations of DA schemes in training sets may also result in optimal DA schemes which could be difficult to implement manually.

Modern Neuro-evolution approaches geared towards deep learning often make use of backpropagation to train networks designed by the evolutionary process. This is in contrast to older methods which optimised weights as well as topologies via evolution. While backpropagation used in supervised learning can lead to faster convergence than evolution based weight tuning, it can still be a slow process. Lamarckian weight inheritance has been shown to decrease training time and therefore convergence time in LEMONADE by offering new topologies 'warm starts'. Implementing this in CoDeepNEAT may offer similar speed ups.

Adapting the CoDeepNEAT genetic algorithm to include the NSGA-II non-dominated sorting algorithm may improve CoDeepNEAT's exploration of the Pareto optimal tradeoffs between different objectives. With Accuracy and data efficiency as objectives, this better exploration may aid analysis to further the understanding of what properties allow networks to generalise on less data. By studying the properties which emerge in networks as they become more data efficient and less accurate, perhaps guidelines could be developed to help network design in cases where there is limited data availability. Including both regularisation and data augmentation schemes in the evolvable set of hyper parameters may allow for optimal use of these methods and speed up training.

References

[1] Stanley, K.O. and Miikkulainen, R., 2002. Efficient evolution of neural network topologies. In Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600) (Vol. 2, pp. 1757-1762). IEEE.

[2] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N. and Hodjat, B., 2019. Evolving deep neural networks. In Artificial Intelligence in the Age of Neural Networks and Brain Computing (pp. 293-312). Academic Press.

[3] Liang, J., Meyerson, E., Hodjat, B., Fink, D., Mutch, K. and Miikkulainen, R., 2019. Evolutionary Neural AutoML for Deep Learning. arXiv preprint arXiv:1902.06827.

[4] Kim, Y.H., Reddy, B., Yun, S. and Seo, C., 2017. Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. In ICML 2017 AutoML Workshop.

[5] Abbass, H.A., Sarker, R. and Newton, C., 2001. PDE: a Pareto-frontier differential evolution approach for multi-objective optimization problems. In Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546) (Vol. 2, pp. 971-978). IEEE.

[6] Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R.R., 2012. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

[7] Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[8] Taylor, L. and Nitschke, G., 2018, November. Improving Deep Learning with Generic Data Augmentation. In 2018 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1542-1547). IEEE.

[9] Perez, L. and Wang, J., 2017. The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621.

[10] Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F.A. and Brendel, W., 2018. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. arXiv preprint arXiv:1811.12231.

[11] Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X. and Yang, M.H., 2017. Universal style transfer via feature transforms. In Advances in neural information processing systems (pp. 386-396).

[12] Bojanowski, P., Joulin, A., Lopez-Paz, D. and Szlam, A., 2017. Optimizing the latent space of generative networks. arXiv preprint arXiv:1707.05776.

[13] Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V. and Le, Q.V., 2018. Autoaugment: Learning augmentation policies from data. arXiv preprint arXiv:1805.09501.

[14] Berthelot, D., Raffel, C., Roy, A. and Goodfellow, I., 2018. Understanding and improving interpolation in autoencoders via an adversarial regularizer. arXiv preprint arXiv:1807.07543.

[15] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.A.M.T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation, 6(2), pp.182-197.

[16] Vasconcelos, J.A., Ramirez, J.A., Takahashi, R.H.C. and Saldanha, R.R., 2001. Improvements in genetic algorithms. IEEE Transactions on magnetics, 37(5), pp.3414-3417.

[17] Zitzler, E., Deb, K. and Thiele, L., 2000. Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary computation, 8(2), pp.173-195.

[18] Zitzler, E. and Thiele, L., 1998, September. Multiobjective optimization using evolutionary algorithms—a comparative case study. In International conference on parallel problem solving from nature (pp. 292-301). Springer, Berlin, Heidelberg.

[19] Knowles, J. and Corne, D., 1999, July. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In Congress on Evolutionary Computation (CEC99) (Vol. 1, pp. 98-105).

[20] Ciro, G.C., Dugardin, F., Yalaoui, F. and Kelly, R., 2016. A NSGA-II and NSGA-III comparison for solving an open shop scheduling problem with resource constraints. IFAC-PapersOnLine, 49(12), pp.1272-1277.

[21] Deb, K. and Jain, H., 2014. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. IEEE Transactions on Evolutionary Computation, 18(4), pp.577-601.

[22] Ishibuchi, H., Imada, R., Setoguchi, Y. and Nojima, Y., 2016, July. Performance comparison of NSGA-II and NSGA-III on various many-objective test problems. In 2016 IEEE Congress on Evolutionary Computation (CEC) (pp. 3045-3052). IEEE.

[23] Elsken, T., Metzen, J.H. and Hutter, F., 2018. Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution. arXiv preprint arXiv:1804.09081.

[24] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

Additional Reading

[A] Rodriguez, J. (2019). Understanding Hyperparameters Optimization in Deep Learning Models: Concepts and Tools. [online] Towards Data Science. Available at: https://towardsdatascience.com/understanding-hyperparameters-optimization-in-deep-learning-models-concepts-and-tools-357002a3338a [Accessed 1 May 2019].

[B] Hwang, Y. (2019). What is AutoML? Promises vs. Reality | IoT For All. [online] IoT For All. Available at: https://www.iotforall.com/what-is-automl-promises-vs-reality auto/ [Accessed 1 May 2019].

[C] Senior, A., Jumper, J. and Hassabis, D. (2019). AlphaFold: Using AI for scientific discovery | DeepMind. [online] DeepMind. Available at: https://deepmind.com/blog/alphafold/ [Accessed 1 May 2019].

[D] Brownlee, J. (2019). What is Deep Learning?. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/what-is-deep-learning/ [Accessed 1 May 2019].

[E] Brownlee, J. (2019). A Gentle Introduction to Transfer Learning for Deep Learning. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/transfer-learning-for-deep-learning/ [Accessed 1 May 2019].