On the Application of Continuous Deterministic Reinforcement Learning in Neural Architecture Search

by

Keith George Mills

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Architecture evaluation is a major bottleneck of Neural Architecture Search (NAS). Recent trends have seen a shift in favor of weight-sharing networks capable of superimposing all possible candidate architectures in a search space. Nevertheless, this technique is not beyond reproach, and has already encountered significant criticism. Of these is the ability of weight-sharing supernets to accurately represent the characteristics of a single discrete architecture when they are purposefully designed to mimic the behaviour of many.

As the cost of NAS evaluation decreased, the complexity of search algorithms has grown. In this thesis, we explore the application of Reinforcement Learning (RL) in the problem space of weight-sharing NAS. Specifically, we focus on the usage of deterministic agents operating in a continuous action space. First, analogous to gradient-based optimization, we train both the supernet and agent simultaneously and interface them accordingly. Our agent consists of an actor-critic framework, where the actor generates architectures based on the teachings of the critic. Rewards are calculated to encourage the selection and further improvement of high-performance architectures.

Next, we refine the efficiency of our weight-sharing supernet, while decoupling optimization with the RL agent. These reforms lower the resource cost during architecture search and remove unhelpful biases the supernet may have imposed on the agent. We adapt the RL agent to these changes by redefining the state as statistical representation of the best architectures observed. Finally, in order to focus on only the most high-performance architectures, we incorporate the check loss into the critic.

Experimental results on DARTS show that our first scheme is capable of generating architectures that achieve over 97% test accuracy on CIFAR-10 and 81% test accuracy on CIFAR-100. Findings indicate that the agent of our second approach is capable of state-of-the-art test performance on NAS-Bench-201. Additionally, architectures generated by our second approach achieve over 97.4% test accuracy on CIFAR-10 and 75% top-1 accuracy on ImageNet.

To my late grandfather, Keith Maynard Mills

Acknowledgements

I would first express deep appreciation for my supervisor, **Dr. Di Niu**. I thank him for inducting me into his research group and giving me the opportunity to further my passion for learning and expand the frontiers of my expertise, both as a newly graduated engineering student and budding research scientist. Dr. Niu has challenged and pushed me, not only to become a stronger, more competent researcher, but an ambitious, yet impartial author. Most of all, Dr. Niu has helped me to better define the boundary between sticking with what you know, or know will happen, and when it it best to put all your heart towards a leap into the unknown.

Second, I would be amiss to not recognize the researchers at the Huawei Canada Edmonton Research Centre, such as Mohammad Salameh, Seyed Saeed Changiz Rezaei, Fred X. Han and Hengshuai Yao, that I, as an Associate Intern, worked alongside to create the results I am now presenting. This thesis is being published in the year A.D. 2020; a year that lies marred beneath the shadow of the pernicious Novel Coronavirus, COVID-19. Although the danger we face has confined us to our homes, we have not allowed ourselves to become divided nor has our progress been halted.

Third, I recognize Jean-Pierre De Villers, Sorin Nita, Matthew Morin and Neil O'Donnell. In the 2013-2014 academic year, a time that now seems like a lifetime ago, these talented gentlemen taught me during the first year of my engineering degree at Keyano College. Without their guidance, I may have never finished my undergraduate degree, much less even started my Masters.

Finally, I acknowledge and give thanks to my closest family members. I thank my mother, Lori Lee Wright, for her unerring and invaluable support. I also thank my father, Brian Keith Mills, for his support and encouragement. With gratitude, I acknowledge several of my paternal relatives, such as my grandparents, Keith Maynard Mills and Heather Mills, and Aunt, Brenda Mills. During my journey as a Master's Student they showed me the half of my lineage I could never truly appreciate as a child.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

All Neural Architecture Search (NAS) methods consist of three core components (Elsken et al., 2018b). The first of these is the search space, which describes what operation and topology choices exist when crafting network architectures. For example, the choice of what activation function to use in a Multi-Layer Perceptron (MLP), or what type of pooling to use in a Convolutional Neural Network (CNN), are both examples of operation search spaces. Conversely, the number of neurons in the hidden layers of an MLP, or the choice and placement of skip-connections in a CNN, describe topology search spaces. The overall search spaces used in modern NAS (Ying et al., 2019; Dong and Yang, 2020; Siems et al., 2020) involve optimizing both the choice and placement of operations, as well as the topological structure of a network.

The search algorithm, an optimization routine responsible for traversing the search space, is second. Search algorithms come in a variety of different types. Reinforcement Learning (RL) (Zoph and Le, 2017), Evolutionary Algorithms (EA) (So et al., 2019) and Random Search (RS) (Li and Talwalkar, 2020) frameworks have all been implemented for NAS. The choice of search algorithm is mostly limited by the nature of the target search space. For example, NAS-Bench-101 (Ying et al., 2019) is the first publically available NAS-focused dataset, yet due to the nature of how architectures are represented within it, many search algorithms cannot be applied to it (Dong and Yang, 2020). Additionally, Differentiable Architecture Search (DARTS) (Liu et al., 2019) was truly made possible with the introduction of weight-sharing (Bender et al., 2018; Guo et al., 2020).

Finally, the performance evaluation strategy rounds out the trio of NAS components. This is the mechanism by which the search algorithm gauges candidate architectures from the search space. Years ago, this was done by training and evaluating individual architectures from scratch, and thus formed a costly, resource-intensive bottleneck (Real et al., 2019) in NAS. However, much like how the introduction of weight-sharing has enabled the use of gradient-based methods, so too has it lessened the resource costs of performance evaluation.

Weight-sharing condenses an entire search space into a single dynamic neural network, typically called a *oneshot model* or *supernet*. From a practical point of view weight-sharing has made research in NAS more accessible to scientists who may not have access to hundreds of GPUs at a time (Zoph and Le, 2017). Conversely, from a theoretical standpoint, a key advantage of weight-sharing is the relaxation of search spaces from discrete domains to continuous landscapes.

At a lower resource cost, the number of algorithms implementing weight-sharing techniques has grown at a rapid pace in recent years (Bender et al., 2018; Liu et al., 2019; Xie et al., 2018; Cai et al., 2019; Elsken et al., 2018a; Pham et al., 2018; Wang et al., 2020; Chen et al., 2019; Dong and Yang, 2019; Chu et al., 2019; Guo et al., 2020; Jin et al., 2019; Stamoulis et al., 2019; Wu et al., 2019; Xu et al., 2019; Cai et al., 2020; Li et al., 2020; Wan et al., 2020; Dai et al., 2020). This heightened interest has not been without merit. Competing algorithms seek recognition by claiming the much sought after title of *state-of-the-art*, specifically on the benchmark datasets of CIFAR-10 (Krizhevsky, 2009) and ImageNet (Deng et al., 2009). In terms of small-scale models with 10 million parameters of fewer, this crown has rapidly changed hands in the past few years.

1.1 Problem Motivation

Attention entails critique, and NAS is no exception. As the number of schemes employing the use of weight-sharing has grown, so too have the number of criticisms pointing out flaws, whether they pertain to the workings of individual algorithms (Li and Talwalkar, 2020; Zela et al., 2020; Chen and Hsieh, 2020) or the use of weight-sharing altogether (Yu et al., 2020b; Shu et al., 2020; Yu et al., 2020a).

Despite these concerns, NAS architectures found by weight-sharing have proved superior next to manually handcrafted ones. However, credit for these gains cannot be exclusively awarded to their corresponding search algorithms. The aptly titled paper "NAS Evaluation is Frustratingly Hard" (Yang et al., 2019) reveals a reliance on 'tricks' in the evaluation phase of an experiment as a means to boost performance.

The use of expert knowledge during formal evaluation is not a significant problem in isolation. However, it is clear that the destination - high performance metrics, has taken precedence over the journey - how the model that produced said metrics is found. This conceals a deeper, more fundamental issue with NAS worth exploring: *To what extent is the proposed search algorithm working?*

1.2 Solution Motivation

The continuous relaxation of the search space (Elsken et al., 2018b), as is done by DARTS (Liu et al., 2019) and other succeeding gradient-based NAS approaches (Xie et al., 2018; Chen et al., 2019; Dong and Yang, 2019; Jin et al., 2019; Xu et al., 2019; Chen and Hsieh, 2020; Li et al., 2020; Xu et al., 2020; Wu et al., 2019; Cai et al., 2020), has incurred a few issues, especially when the final evaluation is inevitably performed on discrete architectures derived from the supernet. Since supernets are trained to minimize the loss as a whole instead of maximizing the performance of individual architectures, a discretization error, or optimization gap, exists between the performance of an architecture acting as part of a larger supernet and its true performance when evaluated individually (Yu et al., 2020b; Xie et al., 2020).

Moreover, as differentiable methods, these algorithms are fundamentally subject to the limitations of gradient-descent-based optimization techniques when applied to NAS. While it is true that differentiable methods have been remarkably effective for unconstrained neural network optimization (Wilson et al., 2017), it is unclear whether this will naturally extend to optimizing architecture parameters, which can lie in constrained, non-Euclidean domains (Li et al., 2020). In fact, Zela et al. (2020) found the DARTS algorithm unstable. They show that DARTS is unable to generalize to different search spaces, some of which contain dummy operators that a search algorithm should be able to discriminate against. Additionally, Zela et al. (ibid.) propose regularization to delay the instability and early stopping to terminate search before the inevitable occurs.

Another common problem facing differentiable NAS approaches, whether by design or consequence, is an inherent lack of exploration. In the former case, gradient-based methods seek the closest, nearest local loss minima as quickly as possible. Therefore, they do not incorporate the same degree of robust exploration as previous NAS frameworks based on reinforcement learning or evolutionary algorithms (So et al., 2019). Consequently, gradient-based NAS approaches are reported to be driven towards regions of the search space where the supernet can train rapidly. This results in wide and shallow architectures being selected over deeper and narrower topologies (Shu et al., 2020). Clearly, there is a motivation to develop more advanced schemes than gradient descent for differentiable NAS.

Reinforcement Learning is a less favorable solution compared to other means, like gradient descent, because RL is not a precise match for NAS (Preiss et al., 2020). Nonetheless, like evolutionary approaches (So et al., 2019), reinforcement learning solutions for NAS can be treated as a double-blind black box. That is, the search algorithm submits an architecture to the search space, which returns information that serves to improve the algorithm and influence the shape of future architectures. The method by which the search space generates information using the architecture is unknown to the algorithm. Likewise, the method by which the algorithm uses the information returned to it in order to generate new architectures, is unknown to the search space. This is not the case when gradient-based optimization is used for NAS as the search space and search algorithm are tightly coupled by losses.

Decoupling search space and search algorithm allows for different optimization methods to be implemented on both. For example, optimization of a weight-sharing supernet can be done by random discrete sampling that is slower but better represents the behaviour of architectures used for benchmark evaluation. Meanwhile, the algorithm policy can be trained using a continuous, gradient-based method that learns at a rapid pace.

It has been shown that differentiable NAS as originally proposed (Liu et al., 2019), does not generalize to alternative search spaces (Zela et al., 2020), datasets (Yang et al., 2019) or publicly available benchmarks (Dong and Yang, 2020). Modularizing the search space and algorithm allows the performance of the latter to be more readily tested on different spaces and data. RL algorithms typically present performance metrics on a variety of different tasks (Mnih et al., 2013; Lillicrap et al., 2016; Schulman et al., 2017; Fujimoto et al., 2018; Haarnoja et al., 2018).

The application of RL for NAS is not in itself novel (Zoph and Le, 2017; Pham et al., 2018; Wang et al., 2020; Bender et al., 2020). However these methods operate in a discrete action space using stochastic policies. Architectures are constructed sequentially as NAS search spaces are too large for whole architectures to be selected all at once. NAS-Bench-101 (Ying et al., 2019) and NAS-Bench-201 (Dong and Yang, 2020) are small search spaces yet contain over 400k and 15k architectures, respectively. By contrast one of the first continuous RL algorithms, Deep Deterministic Policy Gradient (DDPG), was originally proposed to operate in problem spaces where discrete policy optimization "is likely intractable" (Lillicrap et al., 2016).

1.3 Scope of Research

This thesis presents the evolution of a continuous RL scheme applied to the problem space of weight-sharing NAS. We base our research in one of the most popular (Xie et al., 2020) NAS search spaces, DARTS (Liu et al., 2019). Weight-sharing supernets are trained discretely (Li and Talwalkar, 2020) in order to minimize the error incurred when architectures are sampled at the end of an experiment (Chen and Hsieh, 2020).

Actions are produced by our RL agent, then mapped to discrete architectures to be used by the supernet. We present two algorithms, Deep Deterministic Architecture Search (DDAS) and Continuous Action-Discrete Architecture Mapping (CADAM), respectively.

First, we provide a review of relevant NAS methods in Chapter 2. Specifically, we list details on different NAS objectives and methods and how they relate to DDAS and CADAM. Then we provide background information on Differentiable Architecture Search.

Chapter 3 introduces DDAS, an RL scheme that can be adapted in place of a gradient-descent search algorithm. We warm-up the weight-sharing supernet for a short period of time before using a continuous RL search policy to direct further optimization. The agent learns alongside the warmed-up supernet and is designed accordingly. We focus our efforts on how to use Pareto frontiers to incorporate computational constraints without explicitly modifying the supernet loss or RL reward functions.

CADAM is elaborated on in Chapter 4. We separate training of the search space and search policy so that the former is fixed while the latter is optimized. We further modify the search algorithm to better suit the problem objective through the use of a quantile regression loss function. In place of resource constraints and Pareto frontiers, we describe an agent-environment pair that provides metrics regarding the composition of high-performing architectures in a given search space.

Finally, we summarize and compare our findings in Chapter 5. We then provide a brief discussion regarding future research directions, including applications and methods for further refinement.

Chapter 2

Background

Before describing our methods of applying continuous reinforcement learning, we will first establish a necessary context through a review of recent advancements in neural architecture search. Additionally, we provide a mathematical description of the search spaces used throughout this thesis.

2.1 Review of Recent Literature

Originally, NAS is a resource-intensive task, since every candidate architecture must be fully trained to execute performance evaluation, which is used to guide the search algorithm. For instance, NASNet (Zoph et al., 2018) and AmoebaNet (Real et al., 2019) spend over 2,000 GPU days to find the best architecture. To reduce this cost, succeeding works like ENAS (Pham et al., 2018) adopt a weight-sharing scheme by training a supernet containing all possible operations and connections. Each architecture then inherits the corresponding weights from the supernet.

Weight-sharing NAS techniques mainly branch into two categories, ones that rely on random search, and ones that attempt to incrementally learn a distribution of the best architectures. Under the first category, Bender et al. (2018) train a oneshot model once and then sample architectures from a fixed distribution for performance estimation and search. Guo et al. (2020) sample a single path uniformly from a supernet to train the shared weights. Similarly, Li and Talwalkar (2020) randomly sample a child network and only update a selected set of shared weights.

In comparison, DARTS (Liu et al., 2019) uses gradient descent to try and learn the architecture distribution parameters during supernet training. This method has given rise to numerous NAS schemes built upon differentiable architecture search. P-DARTS (Chen et al., 2019) breaks the search procedure into different stages, where the number of cells in the supernet grows while the search space is pruned to bridge the optimization gap. PC-DARTS (Xu et al., 2019) searches on ImageNet by using partial channel connections to decrease the memory cost of backpropagation. This same technique also serves to regularize any bias towards parameterless operations, such as skip-connections and pooling layers. Similarly, our supernets are based on DARTS in Chapter 3 and as well as PC-DARTS in Chapter 4. However, we modify the supernet training procedure using random sampling inspired by Li and Talwalkar (2020) to ensure that the architectures used during training match those that can be selected for evaluation.

Many attempts have focused on identifying and potentially correcting the flaws of gradient-based NAS methods. GDAS (Dong and Yang, 2019) and SNAS (Xie et al., 2018) help bridge the optimization gap by using Gumbel Softmax techniques to sample a single operation per edge during the forward pass. ProxylessNAS (Cai et al., 2019) incorporates a binary gating mechanism into the search procedure. Zela et al. (2020) propose early stopping based on the validation loss in DARTS. GAEA (Li et al., 2020) modify the weight update equation using a simplex projection to ensure better convergence. SDARTS (Chen and Hsieh, 2020) aim to reduce the discretization error by forcing the supernet weights to generalize to a broader range of architecture parameters. Yu et al. (2020b) demonstrate that the DARTS policies perform similarly to random search and are heavily dependent on the initial random seed. By contrast our schemes presented in Chapters 3 and 4 do not use gradient-descent as our search algorithm, but a continuous RL agent that falls into the actor-critic framework, which can still be trained efficiently using gradient ascent.

Hardware constraints, such as FLOPS, model size and inference time, are considered by a number of NAS schemes (Tan et al., 2019; Wu et al., 2019; Stamoulis et al., 2019). Using an Evolutionary Algorithm, Elsken et al. (2018a) approximates the Pareto frontier of architectures under multiple objectives. SNAS (Xie et al., 2018) and ProxylessNAS (Cai et al., 2019) handle hardware-friendly objectives, i.e., latency, to tailor the search for specific devices, i.e., CPU, GPU, or Mobile, by adding loss regularizers. We confront these issues in Chapter 3 with DDAS. The key difference of our approach is that instead of introducing penalty terms, which necessitate repeated search runs, DDAS generates the Pareto frontier in a single search by judiciously striking a balance between exploration and exploitation. A similar oneshot Pareto frontier search scheme is presented in Cai et al. (2020), which decouples supernet training and search, and uses a progressive shrinking trick to combat interference between child models. In contrast, DDAS solves the supernet training and architecture search as a holistic problem, relying on the ability of DDPG to discover and train important architectures on the Pareto frontier in a continuous search space.

Shu et al. (2020) show that weight-sharing NAS algorithms that train supernets and the policies that operate on them in parallel are biased towards selecting wide, shallow cell architectures. Yu et al. (2020b) claim that architecture rankings obtained using weight-sharing techniques do not properly reflect the true architecture rankings during formal evaluation. We incorporate exploration strategies in the DDPG frameworks of both DDAS and CADAM to avoid over-exploitation. Additionally, we separate supernet training from architecture search in Chapter 4, which helps to reduce bias toward shallow architectures.

Last, but certainly not least, a number of RL-based NAS methods, including, but not limited to, NASNet (Zoph et al., 2018) and MNASNet (Tan et al., 2019), have been proposed. ENAS (Pham et al., 2018) is the first reinforcement learning scheme to use weight-sharing. By use of a discrete agent, TuNAS (Bender et al., 2020) shows that guided policies decisively exceed the performance of random search on vast search spaces. Like Zoph and Le (2017), most of these schemes use a simple agent trained by REINFORCE (Williams, 1992) to select architectures parameters or Proximal Policy Optimization (PPO) (Schulman et al., 2017) to learn to sample child networks in a stochastic manners. One exception is AlphaX (Wang et al., 2020) that uses Monte Carlo Tree Search to balance exploration with exploitation during the search. Nevertheless, all of these methods operate in a discrete space where the RL state keeps track of the partial architecture that is built over many steps. By contrast, DDAS and CADAM select an entire architectures at every step. Since DDAS trains the supernet alongside the agent in parallel, the state records the previously-selected architecture. By contrast, CADAM uses the RL state to keep track of the distribution of high performing architectures in the search space.

2.2 Differentiable Architecture Search

The Convolutional Neural Network architectures used by DARTS (Liu et al., 2019) consist of two types of cells, normal and reduction, that are stacked to form neural networks. The majority of network cells are normal cells where the output data dimension - channels, height and width - match the input data dimension. Each network contains two reduction cells that double the number of channels while halving the height and width of input data.

DARTS adopts a continuous relaxation of searchable operators, which leads to a *supernet* that represents the behavior of all possible architectures for both types of cells. Specifically, each cell is represented by a directed acyclic graph (DAG) with 2 input nodes, $|N|$ ordered intermediate nodes, $|E| = \sum_{i=1}^{|N|}(i+1)$ edges and an output node. The intermediate nodes are latent data representations, while the edges perform a weighted sum of $|\mathcal{O}|$ candidate operations, from a predefined operation space denoted by $\mathcal{O}$. Given input data x_i, the computation performed at each edge (i, j) where $i < j$, is defined as,

$$f_{i,j}(x_i) = \sum_{o \in \mathcal{O}} \alpha_{(i,j),o} o(x_i),$$

where $\alpha \in \mathbb{R}^{|E| \times |\mathcal{O}|}$ are architecture distribution parameters. The output of a cell is the channel-wise concatenation of the outputs of each intermediate node x_j.

PC-DARTS (Xu et al., 2019) further extends this framework with the inclusion of partial channel connections to overcome the memory inefficiency of DARTS. This involves masking out certain channels based on a masking distribution to bypass the mixed operations computation, thus allowing for larger batch sizes during training. It also includes additional architecture distribution parameters, $\beta \in \mathbb{R}^{|E|}$, which can help overcome instability caused by sampling channels. PC-DARTS performs a weighted summation of all edges directed into a given node. Specifically, for node $j \in N$,

$$x_j = \sum_{i<j} \beta_{(i,j)} f_{i,j}(x_i).$$

During the search phase, the architecture distribution parameters exist in a *continuous* domain, and are updated alongside the model weights, w, using gradient descent. At the end of the search phase, the architecture distribution parameters are used to determine a single *discrete* cell architecture that can be retrained from scratch to perform a formal evaluation. The process consists of the following steps: Each node in N will receive input from only two of the directed edges that can feed into it. Each of these selected edges shall perform a single operation. The choice of which edges are chosen, and which operators will occupy these edges are determined using the magnitude of their architecture distribution parameters, where higher is better. Therefore, during the search, a cell of the DARTS supernet contains $|N|$ intermediate nodes and no less than $|E| = \sum_{i=1}^{|N|}(i+1)$ edges, while the discretized cell for evaluation contains no more than $2|N|$ edges.

DARTS and PC-DARTS excludes the 'none' operation from being selected regardless of the architecture distribution parameters. Additionally, P-DARTS (Chen et al., 2019) places limits on the number of 'skip-connect' operations that may be selected.

These discrepancies, pertaining to the number of edges or allowed operations, leads to a performance loss or *discretization error* (Xie et al., 2020) when deriving the architecture from the continuously relaxed supernet for final evaluation.

Chapter 3

Parallel Optimization of Supernet *and* Policy

In this chapter, we use reinforcement learning to efficiently explore an architecture search space and propose Deep Deterministic Architecture Sampling (DDAS), a weight sharing NAS algorithm. Based on Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016), a continuous, off-policy RL agent, DDAS is capable of efficiently generating a Pareto frontier of architectures in terms of the accuracy and FLOPS, or accuracy and number of parameters, to find the best architecture in a single run. Specifically, we make the following contributions.

First, we model NAS as a continuous control problem in a high-dimensional search space. Similar to DARTS (Liu et al., 2019), SNAS (Xie et al., 2018) and Proxyless-NAS (Cai et al., 2019), we parameterize the search space using a set of continuous weights of operations that connect latent vectors. However, instead of updating these weights of operations with gradient descent or bi-level optimization, we rely on the ability of DDPG to explore and sample them in an actor-critic framework. Empirical evidence shows that DDPG performs well in high-dimensional control tasks with continuous actions, e.g., robotic control (Duan et al., 2016).

Second, previous reinforcement learning schemes proposed for NAS, e.g., ENAS, mainly use a stochastic policy to sample architectures for training over a sequence of steps. In contrast, we use a deterministic policy in DDAS, which can generate one whole architecture in a single step. Additionally, the deterministic policy gradient can be estimated much more efficiently than the usual stochastic policy gradient, as is shown in DPG (Silver et al., 2014) and DDPG (Lillicrap et al., 2016).

Third, we judiciously design the reward and strike a balance between exploitation and exploration when updating the actor and critic networks in DDAS, such that the agent will maintain a high reward while being allowed to explore a potentially large space of candidate architectures, which gradient-based optimization methods fail to fully explore. In fact, in the pursuit of a higher validation performance, gradient-based optimization schemes may often converge to a single large architecture, which is repeatedly selected for training, preventing these schemes from sampling other architectures in the search space and generating the Pareto frontier. In DDAS, exploration over diverse architectures in the search space is achieved by the use of a combination of noise-based exploration schemes in a continuous space.

Fourth, the problem of NAS is further complicated by the need for hardware-friendly architecture search, e.g., by budgeting the number of floating point operations used to forward pass a single data sample (FLOPS), inference time, or the total number of parameters that a model can have. Schemes such as SNAS (Xie et al., 2018), RC-DARTS (Jin et al., 2019) and ProxylessNAS (Cai et al., 2019) address the problem by introducing constraints or penalty into their optimization formulations. In practical deployment, however, the need for depicting a Pareto frontier of architectures necessitates repeated executions of these algorithms under different constraints, which is costly. Therefore, our final contribution is the design of an algorithm capable of generating a Pareto frontier that features adequate coverage of many regions of the search space, both within the context of a single experiment and without the use of explicit regularizers in the reward or loss functions.

A series of experimental results on CIFAR-10 and CIFAR-100 (Krizhevsky, 2009) suggest that the Pareto frontiers generated by DDAS show a clear superiority over that of random search over a randomly warm-started supernet (Li and Talwalkar, 2020). In the meantime, the test accuracy of the best architectures found by DDAS is comparable to various state-of-the-art NAS methods that rely on weight sharing.

3.1 Methodology

In this section, we describe the detailed mechanisms of our proposed scheme, DDAS. First, we present our RL agent and environment, followed by a description of the training procedure and methods to balance exploitation and exploration.

3.1.1 The DDAS Agent

In reinforcement learning, at time-step t, an agent in state s_t interacts with an environment by executing an action a_t. The environment in turn returns a reward, r_t, and the agent observes the next state, s_{t+1}. The goal of the agent is to maximize the return $R = \sum_{t=0}^{T-1} \gamma^t r_t$ over T time steps subject to a discounting factor $\gamma \in [0, 1)$.

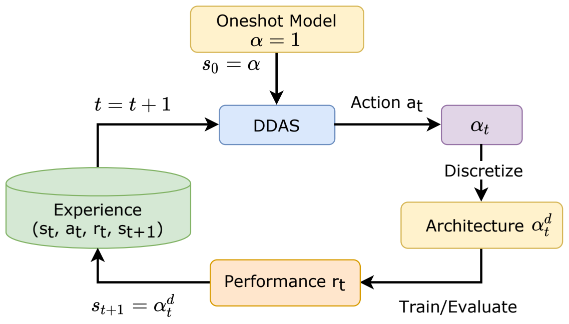


Figure 3.1: An illustration of one DDAS step.

DDAS is based off DDPG, which adopts an actor-critic framework. The actor $\mu(s_t)$ is a neural network that takes a state s_t as its input, and produces an action

$$a_t = \mu(s_t) + Z_t, \tag{3.1}$$

where Z_t is a noise added to the actor's output to encourage exploration of architectures.

The critic $Q(s_t, a_t)$ is a neural network that is trained to maximize the return by predicting the action value of a state-action pair (s_t, a_t). On the other hand, the actor learns the optimal policy necessary to maximize the return.

A replay buffer is used to store interactions with the environment, the supernet, in a form of experience tuples (s_t, a_t, r_t, s_{t+1}). *Experiences* can be randomly sampled with replacement in order to train the actor and critic. Our actor and critic are simple MLPs with three hidden layers applied onto the vectorized inputs.

3.1.2 Environment Interaction

We now describe the interaction of the DDAS agent with the environment through the action, state and reward. We split all the available training data into two non-overlapping sets, the training data $\mathcal{D}_T$ and validation data $\mathcal{D}_V$; the first is used for training the supernet weights w while the latter is used to evaluate the performance of a given architecture. The DDAS procedure is illustrated in Figure 3.1.

We first initialize the environment by setting every element of α to one to obtain the supernet with all the operations present. Then, we *warm up* the supernet with several epochs of training (Bender et al., 2018; Guo et al., 2020). The accuracy of the warmed-up one shot model OS is denoted by *Acc(OS)*.

The DDAS agent interacts with the environment in an iterative process. In particular, the actor network will output the action $a_t = \alpha_t$, where α_t represents the α generated at time step t. This action a_t will be evaluated by the environment according to Algorithm 1 to obtain r_t and s_{t+1}.

Algorithm 1 Architecture Sampling and Evaluation

```
1: Input: $\alpha_t$ given by the DDPG actor
2: $\alpha_t^d \leftarrow$ Discretize($\alpha_t$)                # Algorithm 2
3: for $M$ minibatches do
4:     Sample a minibatch $m$ from $\mathcal{D}_T$
5:     Update supernet weights selected by $\alpha_t^d$ using SGD and $m$
6: end for
7: Compute the loss $\mathcal{L}_V(\alpha_t^d)$ and the accuracy Acc($\alpha_t^d$) of the selected architecture
   on $\mathcal{D}_V$ using weights inherited from the supernet.
8: Return $r_t$ by Equation 3.3 and $s_{t+1} = \alpha_t^d$.
```

Algorithm 2 Map Continuous α to Discrete Architecture (DARTS)

```
1: Input: $\alpha \in \mathbb{R}^{|E|\times|\mathcal{O}|}$
2: Output: $\alpha^d \in \{0,1\}^{|E|\times|\mathcal{O}|}$
3: Start $= 0, n = 1$
4: $\alpha^d = 0^{|E|\times|\mathcal{O}|}$                # Initialize the $\alpha^d$ matrix to zero
5: for $k = 0, 1, .., |N| - 1$ do
6:     End = Start + $n$                  # $|N|$ is the number of intermediate nodes
7:     $A = \alpha$[Start : End, :]           # Rows of a specific intermediate node
8:     $(i_1, j_1) = \arg\max_{(i,j)} A_{ij}$
9:     $(i_2, j_2) = \arg\max_{(i,j):i\neq i_1} A_{ij}$
10:    $\alpha^d$[Start + $i_1, j_1$] = 1
11:    $\alpha^d$[Start + $i_2, j_2$] = 1
12:    Start = End + 1
13:    $n = n + 1$
14: end for
15: Return $\alpha^d$
```

Having generated α_t at time step t, we discretize α_t to obtain $\alpha_t^d \in \{0,1\}^{|E|\times|\mathcal{O}|}$. Given an intermediate node, we select the top two edges with the highest operation weights incoming from all its predecessor nodes. Then we discretize the two edges by setting the index of the operation with highest weight on each edge to one and the rest to zero, i.e.,

$$\alpha_t^d = \text{Discretize}(\alpha_t), \tag{3.2}$$

where a detailed definition of Discretize is given in Algorithm 2. In fact, α_t^d corresponds to a single deterministic architecture, with a controlled complexity, whose corresponding weights in the supernet will be updated using SGD. This architecture is then evaluated on $\mathcal{D}_V$ to calculate the reward.

We define the reward at time step t , i.e., r_t, in terms of the selected architecture's

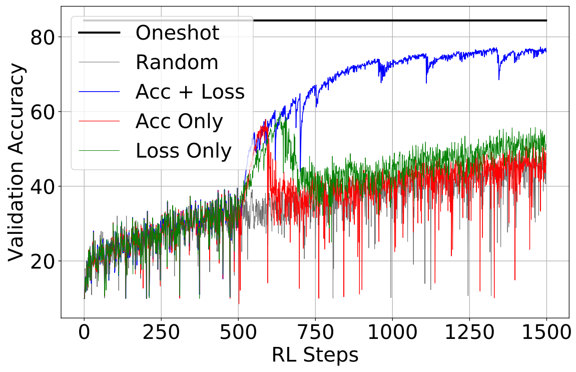


Figure 3.2: Comparison of DDAS performance on CIFAR-10 with (blue) the full reward function, without the loss term (red) and without the accuracy term (green). DDAS becomes nearly indistinguishable from Random Search (grey) when either the loss or accuracy terms are removed.

validation accuracy, $\mathrm{Acc}(\alpha_t^d)$, and the validation loss $\mathcal{L}_V(\alpha_t^d)$ as

$$r_t = \frac{1}{2}(\mathrm{Acc}(\alpha_t^d) - \mathrm{Acc}(\mathrm{OS})) + \frac{1}{2}(-\mathcal{L}_V(\alpha_t^d) + \mathcal{L}_V(\alpha_{t-1}^d)). \tag{3.3}$$

The accuracy term encourages the DDAS agent to select well-performing architectures, while the validation loss term, e.g., cross-entropy loss in the case of classification, encourages the agent to constantly improve. Moreover, the addition of loss in the reward is empirically critical to addressing concerns raised by Yu et al. (2020b). As Figure 3.2 shows, without a loss component, the actor policy eventually degenerates into random search.

Finally, the agent sets the next state to the selected architecture, i.e., $s_{t+1} = \alpha_t^d$, and continues the process to find a better architecture.

3.1.3 Exploration and Exploitation

The goal of DDAS is to generate a Pareto frontier of architectures in terms of accuracy and FLOPS through a single run of the algorithm. Intuitively speaking, we can also obtain the Pareto frontier by warming up the supernet and then applying random search or evolutionary algorithms over architectures that inherit weights from the supernet. In contrast, optimization schemes such as DARTS, SNAS, etc., are not capable of depicting the Pareto frontier in one run, as gradient descent will drive these schemes to train a single or a few large architectures fully in order to minimize the validation loss of selected architectures.

The ability of DDAS to discover a better Pareto frontier in one run critically depends on a balance between exploration and exploitation processes. Since DDPG is *off-policy*, it benefits from the use of an experiential replay buffer. DDPG splits exploration and exploitation into two sequential phases. In the first phase, neither the actor nor the critic is used or updated. Instead, the agent accumulates a diverse collection of state transitions in its replay buffer by sampling actions from a random distribution. In the second phase, i.e., the exploitation-centered phase, actions are generated by the actor using Equation 3.1. The agent samples a random batch B_R of experiences from its replay buffer and uses them to update the critic network according to the loss,

$$\mathcal{L}_{Critic} = \frac{1}{|B_R|} \sum_{i \in B_R} (r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1})) - Q(s_i, a_i))^2, \tag{3.4}$$

where Q' and μ' are the *target networks* used to aid in the training procedure; refer to Lillicrap et al. (2016) or Mnih et al. (2013) for further details. The actor network is then updated using a sampled policy gradient from the critic,

$$\mathcal{L}_{Actor} = \frac{1}{|B_R|} \sum_{i \in B_R} Q(s_i, \mu(s_i)). \tag{3.5}$$

One caveat of Equation 3.4 is that given our definition of the reward, the actor will learn to sample the same architecture repeatedly regardless of the state in order to train this architecture fully to increase the validation performance. In DDAS, we introduce exploration in architecture sampling through the use of two types of noise.

First, we introduce exploration during the exploitation phase by adding a Gaussian noise to the actor output in every step, as in Equation 3.1. However, it may not be strong enough to completely randomize the actions. Rather, it perturbs α such that when discretized into α^d, it is in the same neighborhood of the actor's output. If the agent samples from a small neighborhood repeatedly, the validation performance will be guaranteed to improve, as the shared weights are repeatedly updated.

To further encourage exploration in DDAS, we introduce a new phase to follow the normal exploitation phase, where we replace the Gaussian noise by the Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930), which is more effective than Gaussian noise at overwriting actions (Lillicrap et al., 2016). Thus, we do not add Ornstein-Uhlenbeck process to the actor output in every step. When the agent detects that α has been stagnant, measured by observing minimal changes from step-to-step, for a number of steps T_{stag}, the new noise is added to the actor's output for the next T_{stag} steps. When the noise is off, the agent will focus on a small number of architectures that the critic deems worthwhile and continuously train their weights, driving up the validation performance. On the other hand, when the Ornstein-Uhlenbeck process is temporarily introduced, the newly selected architectures will become radically different, yet still having a few shared weights overlapped with previously selected architectures. This overlap of shared weights can be used to boost the performance of the newly selected architectures.

Through a combined use of the above two types of noise, the DDAS agent can switch attention to seldom sampled architectures including the smaller architectures, so that the Pareto front in terms of validation accuracy and FLOPS can be uplifted.

3.2 Experiments and Discussion

We perform our experiments on two datasets, CIFAR-10 and CIFAR-100 (Krizhevsky, 2009). Both datasets have 60k images each, of dimension size 32×32, with ten classes for CIFAR-10 and one hundred classes for CIFAR-100. Architecture search is performed on a data split similar to DARTS, resulting in a training set $\mathcal{D}_T$, validation set $\mathcal{D}_V$, and test set with sizes 25k, 25k, and 10k samples, respectively. Further evaluation of the best architectures found involves training on the official CIFAR-10 and CIFAR-100 splits that partition the data into 50k training samples and 10k testing samples.

3.2.1 Architecture Search

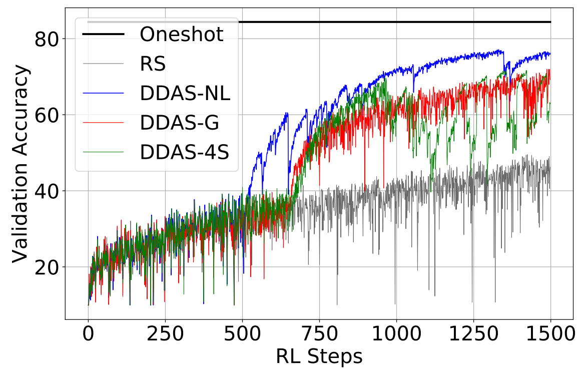
Warmed-up Supernet: We warm up all our architecture search experiments by training a 6-cell oneshot supernet for 75 epochs on $\mathcal{D}_T$ with all elements of α set to one. Supernet training typically takes less than six hours.

Architecture Sampling with DDAS: We initialize DDAS with the warmed up supernet and start the architecture sampling process. For every sampled architecture, the supernet is trained for 25 batches on $\mathcal{D}_T$ to fit the supernets weights to the new architecture configuration. We compare three different DDAS configurations against a random search baseline. Each experiment runs for 1,500 steps and takes around 24 hours to finish.

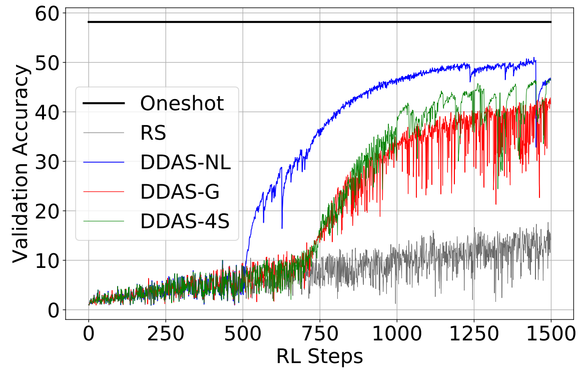
1. *Random Search (RS)*: Following Li and Talwalkar (2020), we train the supernet by randomly sampling architectures from $Uniform(0,1)^{|E|\times|\mathcal{O}|}$.

2. *Noiseless (DDAS-NL)*: After an initial 500 steps of exploration, DDAS enters an almost purely deterministic exploitation phase where $Z_t = U(-10^{-5}, 10^{-5})$.

3. *Gaussian (DDAS-G)*: Same as *DDAS-NL*, however we engender further exploration during the exploitation phase by disrupting the actor's output a_t with a noise sampled from a Normal distribution, $Z_t = \mathcal{N}(0, 0.05)$.

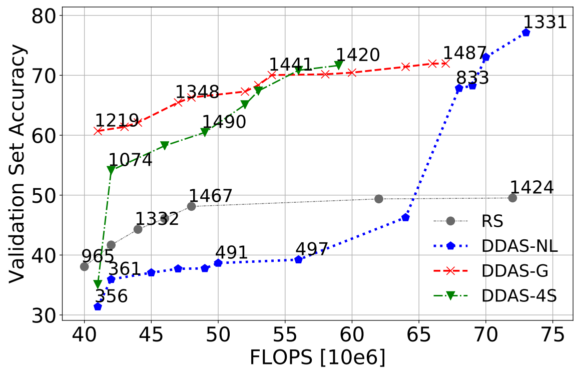


(a) CIFAR-10 validation curves

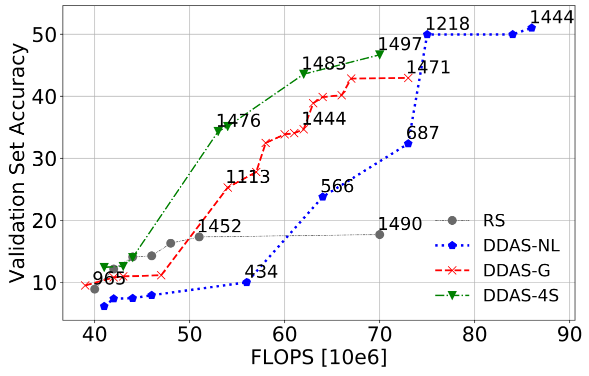


(b) CIFAR-100 validation curves

Figure 3.3: Search validation curves on CIFAR-10 and CIFAR-100 as a function of RL steps. The 'Oneshot' line represents the best validation accuracy the warmed-up supernet obtained prior to architecture search.



(a) CIFAR-10 FLOPS Pareto frontiers

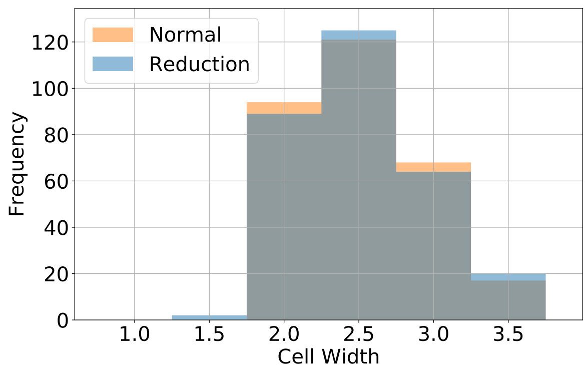


(b) CIFAR-100 FLOPS Pareto frontiers

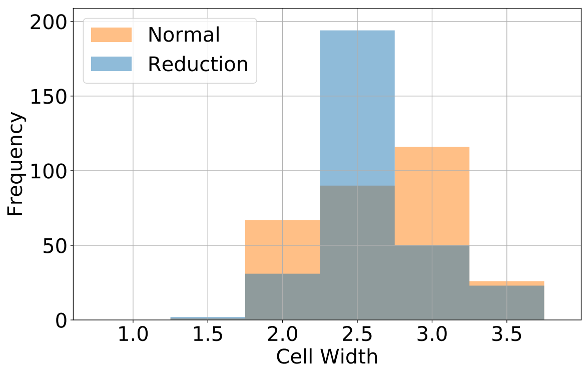
Figure 3.4: Search Pareto frontiers in terms of validation accuracy and FLOPS. Each line is generated by one of the four search schemes. Numerical annotations denote the step t where an architecture was sampled. Any step below 500 is guaranteed to be generated from a uniform random distribution, $U(0,1)^{|E|\times|\mathcal{O}|}$.

4. *4-Stage (DDAS-4S)*: Behaves like *DDAS-G* for the first 500 steps of the exploitation phase. In the last 500 steps of the experiment, the additive noise is turned off by default, $Z_t = 0$, then re-enabled sporadically. The key difference is that the agent keeps track of selected architectures. The agent considers two architectures to be similar if less than 6 of the 16 activated operation-edge pairs between the normal and reduction cells are different. If the agent detects that it has been selecting a similar architecture for $T_{stag} = 32$ steps in a row, then a large, (Uhlenbeck and Ornstein, 1930) noise will be added to the actor output for the next T_{stag} steps.

For each experiment we obtain the best architecture found by DDAS with the



(a) CIFAR-10 Cell Width Histogram



(b) CIFAR-100 Cell Width Histogram

Figure 3.5: Search histograms of normal and reduction cell width distributions on CIFAR-10 and CIFAR-100 across architectures in the top 5% accuracy percentile. Width is defined as the average number of operation-edge pairs originating from the two input nodes, and can take values between 0.5 and 4.

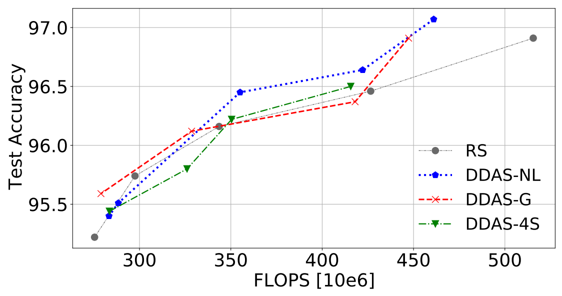
highest validation accuracy on a given dataset. For every architecture sampled by DDAS, we calculate both the number of FLOPS and the number of parameters in a model assuming it is instantiated on a 6-cell network. We construct the Pareto frontier from the validation accuracy of each sampled architecture on the supernet, constrained by the number of FLOPS/parameters on the 6-cell network.

In the second half of our experiments, we forwarded many of the architectures found on the FLOPS Pareto frontiers for further evaluation on larger models for 600 epochs each. The number of cells used were 10 and 20 for CIFAR-10 and CIFAR-100, respectively.

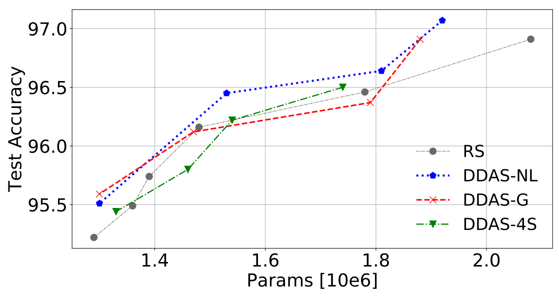
Lastly, we took the absolute best performing architecture from each experimental setting and compared their test accuracies against those of several related NAS algorithms. For comparisons on CIFAR-10, we re-trained these architectures using 20 cell models in order to perfectly match the hyperparameter choices of Liu et al. (2019).

3.2.2 Evaluation and Comparison

Search validation curves for all experiments are illustrated by Figure 3.3. All variants of DDAS demonstrate a clear superiority over random search. The performance of *DDAS-NL* is the quickest to rise following the initial exploration steps, while *DDAS-G* and *DDAS-4S* take a few hundred additional steps before they surpass random search. Additionally, dips and rises in the plots of *DDAS-4S* clearly denote the time steps where a large noise is added to the actor output. The validation Pareto frontiers found by our search experiments, in terms of FLOPS, are presented in Figure 3.4.

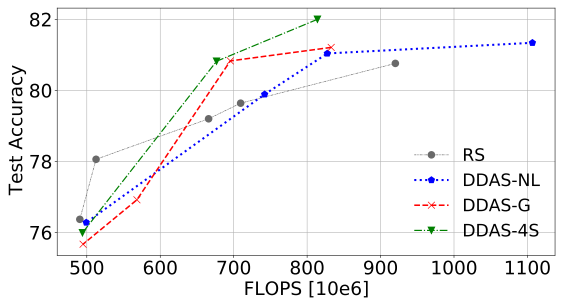


(a) FLOPS Pareto frontiers

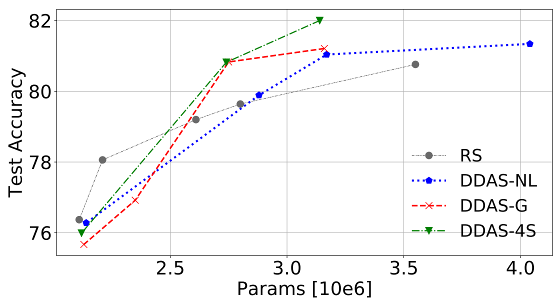


(b) Parameter Pareto frontiers

Figure 3.6: Test set evaluation Pareto frontiers for CIFAR-10. Points correspond to architectures present on the search Pareto frontier for a given search scheme. All models were trained using 10 cells.



(a) FLOPS Pareto frontiers

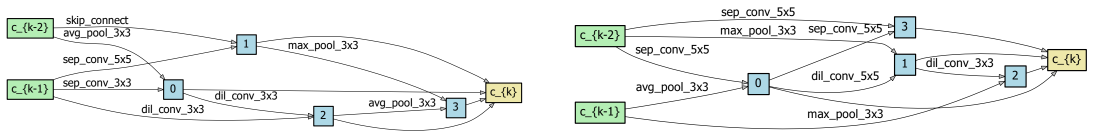


(b) Parameter Pareto frontiers

Figure 3.7: Test set evaluation Pareto frontiers for CIFAR-100. Points correspond to architectures present on the search Pareto frontier for a given search scheme. All models were trained using 20 cells.

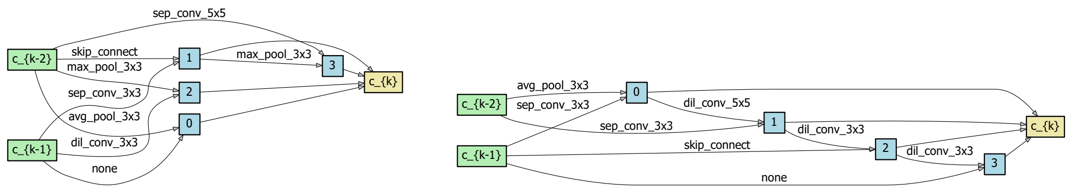
Architectures on these curves were selected for further evaluation through larger models. Note that while *DDAS-NL* appears to outperform all other methods in terms of validation performance over time, Pareto frontier regions corresponding to smaller FLOPS are dominated by *DDAS-G* and *DDAS-4S*.

We adopt the definition introduced by Shu et al. (2020) for measuring the width and depth of NAS cells and exclude the 'none' operation from these calculations. Figure 3.5 displays the histograms of cell widths for cells in the top 5% accuracy percentile for all experiments on both datasets. The distribution of architectures for both datasets resembles that of a Gaussian distribution centered around 2.5; corresponding to 2-3 edges per input node, in the case of CIFAR-10. For CIFAR-100, the distribution of normal cells more closely resembles a uniform distribution bounded between 2 and 4. Reduction cell widths follow a narrow Gaussian centered around 2.5. Regardless of the distribution, it is clear that respectable accuracy metrics can be found across a spectrum of cell widths—high accuracies are not limited to a narrow



(a) Normal Cell, 2.5c, 4 (b) Reduction Cell, 2.5c, 4

Figure 3.8: Best set of cells found on CIFAR-10. These cells were found using the noiseless configuration (DDAS-NL) at step 833.



(a) Normal Cell, 3c, 3 (b) Reduction Cell, 2c, 5

Figure 3.9: Best set of cells found on CIFAR-100. These cells were found using the four-stage configuration (DDAS-4S) at step 1483.

range of cells with large widths. These findings corroborate our claim that NAS algorithms should incorporate a higher degree of exploration and avoid being biased toward a specific type of topologies.

Test set Pareto frontiers, in terms of both FLOPS and total number of parameters, on both datasets, are given by Figures 3.6 and 3.7, respectively. By test set accuracy, the Pareto frontiers of all three DDAS configurations are higher than those of *RS* in at least one region. This reflects the search curves where their architectures were chosen from. *DDAS-NL* is the sole exception to this observation. *DDAS-NL* produced the highest test score on CIFAR-10 and the highest validation scores on both datasets. According to Figure 3.4, the only architectures *DDAS-NL* chose that had a small number of FLOPS were sampled during the initial 500-step exploration phase, or shortly afterward. When comparing *DDAS-G* to *DDAS-4S* we observe that their evaluation Pareto frontiers almost identically match the ones generated during the search. On CIFAR-10, *DDAS-G* is better at sampling low-FLOPS architectures, but is eventually overtaken by *DDAS-4S*. Meanwhile, on CIFAR-100, the *DDAS-4S* Pareto frontiers completely dominate *DDAS-G* on both search and evaluation.

The best cell architectures found for CIFAR-10 and CIFAR-100 are given by Figures 3.8 and Figure 3.9, respectively. With exception to the normal cell for CIFAR-100, no cell has a width above 3 nor a depth smaller than 3. This demonstrates that DDAS is not prone to the same issue as cells found by state-of-the-art algorithms.

Table 3.1: Comparison of DDAS schemes with other state-of-the-art algorithms in terms of test accuracy, gigaFLOPS and millions of parameters. We manually evaluated the publicly available architectures found by DARTS first-order and second-order on CIFAR-100.

	CIFAR-10			**CIFAR-100**		
Architecture	**FLOPS**	**Params**	**Test Acc. (%)**	**FLOPS**	**Params**	**Test Acc. (%)**
DARTS 1st Order	1.022G	3.65M	97.00	1.022G	3.77M	82.37
DARTS 2nd Order	1.078G	3.83M	97.24	1.078G	3.95M	82.65
ENAS	–	4.60M	97.11	-	-	-
ProxylessNAS-G	–	5.70M	97.92	–	–	–
GDAS	–	3.40M	97.07	–	3.40M	81.62
GDAS (FRC)	–	2.50M	97.18	–	2.50M	81.87
SNAS (Mild Const.)	–	2.90M	97.02	–	–	–
SNAS (Mod. Const.)	–	2.80M	97.15	–	–	–
RS	1.024G	3.67M	97.16	0.920G	3.55M	80.76
DDAS-NL	*0.876G*	*3.23M*	*97.27*	1.106G	4.04M	81.34
DDAS-G	0.839G	3.10M	96.81	0.916G	3.47M	80.88
DDAS-4S	0.842G	3.07M	96.74	*0.814G*	*3.14M*	*82.00*

That is, the layout of the cells do not resemble a wide, shallow neural network; each input is not simply passed to each node independently before being aggregated at the output. Instead, the inputs are subject to a series of sequential operations as they are passed from one node onto the next.

Next, we compare the test performance of the best architectures found by all four of our experimental setups to those reported by several other state-of-the-art NAS algorithms using weight sharing and relying on a few GPUs. The results are given in Table 3.1.

Table 3.1 provides evidence that DDAS is superior to ENAS (Pham et al., 2018), GDAS (Dong and Yang, 2019) and SNAS (Xie et al., 2018), where the latter two employ exploration in the form of Gumbel Softmax. The only architectures whose scores are higher than DDAS are ProxylessNAS (Cai et al., 2019) on CIFAR-10 and DARTS (Liu et al., 2019) on CIFAR-100. Both methods achieve their high accuracy metrics at the cost of substantially larger model sizes.

Comparing our experimental configurations against each other, we observe the superiority of *DDAS-NL* and *RS* over *DDAS-G* and *DDAS-4S* on CIFAR-10. Both of these algorithms favored architectures with a much higher number of parameters than *DDAS-G* and *DDAS-4S*. Most notably *RS* is the more inefficient of the two. Moreover, the same situation is partially true on CIFAR-100, where *DDAS-G* and *DDAS-4S* reign supreme with fewer parameters.

DDAS-NL is most comparable to gradient-based NAS algorithms due to a low, almost negligible amount of exploration during exploitation. Conversely, *DDAS-4S* incorporates mechanisms that allow it to actively fight against the sampled policy gradient of its critic, while *DDAS-G* does not heavily depart from the original specification of DDPG given by Lillicrap et al. (2016). On CIFAR-100 *DDAS-4S* completely outperformed *DDAS-NL*, both in terms of performance and parameter efficiency. CIFAR-100 is inherently more difficult to classify than CIFAR-10 due to having the same number of samples but 10 times as many classes and therefore 10 times fewer samples per class. Thus, it can be said that *DDAS-4S* demonstrates the benefits of modifying RL algorithms beyond the scope of their original theory for use in NAS problems.

In addition, we approximated the slope of accuracy against FLOPS or parameters using linear regression. For CIFAR-10, we found that test accuracy increased at rates of 2.86% per gigaFLOPS and 2.123% per million parameters, both with linear correlations over 0.93. For CIFAR-100, these values are higher at 4.03% per gigaFLOPS and 3.196% per million parameters, linearly correlated over 0.86. These metrics quantify the small loss of accuracy entailed by downsizing model size and indicate the ability of DDAS to find resource-efficient architectures for practical deployment.

Table 3.2: Spearman correlation coefficients between validation and evaluation accuracies for Pareto front cells

Setting	**CIFAR-10**	**CIFAR-100**
RS	0.964	1.000
DDAS-NL	0.881	0.826
DDAS-G	0.886	0.810
DDAS-4S	0.886	0.600

We also computed the ranking correlation between the validation and evaluation scores of all Pareto frontier architectures. The Spearman coefficients are given in Table 3.2. This shows that the DDAS is capable of sampling architectures where the true, evaluation ranking is adequately preserved during the search phase.

Finally, the search cost of DDAS is relatively comparable to DARTS. DDAS takes approximately 24 hours to complete 1,500 steps, while the initial phase of warming up the oneshot model takes around 6 hours, for a total of 30 hours on an RTX 2080 Ti GPU. It is worth noting that DARTS, P-DARTS, and GDAS ran their search experiments four or three times with different random seeds in order to pick the best architecture according to the validation accuracy. Repeated searches are a mechanism to encourage exploration. In contrast, DDAS is designed to explore, train and identify a range of good architectures in the same search run.

3.3 Conclusions

We introduce Deep Deterministic Architecture Search (DDAS), an algorithm based on Deep Deterministic Policy Gradient (DDPG) in Reinforcement Learning (RL), to thoroughly explore an architecture search space and perform Neural Architecture Search (NAS) by sampling and training architectures on a weight-sharing supernet. Unlike prior RL schemes for NAS which use stochastic policy gradient to sample architectures, DDAS uses a deterministic policy and leverages the ability of DDPG to handle high-dimensional control in a continuous space. Coupled with a loss-based reward function, the policy of DDAS is distinct from random search and can learn to focus on important regions of the search space.

Furthermore, DDAS addresses the lack-of-exploration issue present in recent NAS frameworks via several exploration schemes. As a result, DDAS is capable of generating a Pareto frontier of architectures in a given search space for flexible deployment on target hardware. Additionally, the cells produced by DDAS are not always wide and shallow or biased toward a specific type of topologies. We performed extensive experiments on CIFAR-10 and CIFAR-100 in a wide range of experimental settings.

Experimental results have shown that DDAS is capable of generating architectures with test accuracies that are competitive with other state-of-the-art efficient, weight-sharing NAS algorithms that are based on no more than a few GPUs. In addition, in a single algorithm run for less than 1.5 GPU days, DDAS can produce Pareto frontiers that outperform random search based on a warm-started supernet, demonstrating its superior capability to automatically explore and discover important regions of a neural architecture search space.

In the next chapter, we decouple the training of the supernet and agent, and perform these tasks sequentially. We also explore what additional alterations, both to the original DDPG algorithm and DDAS, are made possible by this change.

3.4 Additional Experimental Details

In this section we further elaborate on the hyperparameter setups used for our experiments and disclose the computational resources used to perform our experiments.

3.4.1 Search Hyperparameters

Our weight-sharing search models, modified from DARTS (Liu et al., 2019), all have 6 cells; 4 normal and 2 reduction cells. Data enters through a head which applies a channel multiplier of 16 as well as a few preliminary convolution operations, before being passed on to the cells. A batch size of 64 is used at all times, and each supernet is trained over the course of 75 epochs on the 25k training set, $\mathcal{D}_T$. We followed the precedent set by DARTS and utilized a stochastic gradient descent optimizer with momentum. During oneshot supernet training, the initial learning rate is set to 2.5×10^{-2}, but is annealed down to 10^{-3} by a cosine schedule without restarts (Loshchilov and Hutter, 2017). When searching for an architecture using DDAS, we set the learning rate to a constant value of 10^{-3}. For reproducibility, all experiments are initialized with the same random seed values of 2 for search and 0 for evaluation.

3.4.2 Agent Hyperparameters

The actor and critic networks of DDAS are both MLPs with 3 hidden layers and 256 neurons in each layer. Both networks are trained using Adam (Kingma and Ba, 2014) with its default parameters of $\vec{\beta} = (0.9, 0.99)$ and learning rates of 10^{-4} and 10^{-3}, respectively. ReLU (Nair and Hinton, 2010) is used as the internal activation function for both the actor and the critic. However, the actor's final layer uses a sigmoid activation (σ) to truncate the output into the range $(0, 1)$. The critic does not utilize any final activation at all, because it produces a scalar. The target networks (see Lillicrap et al. (2016) for details) are synchronized at every step using a mixing coefficient of 10^{-3}. The replay buffer is truncated to only hold experiences from the last 500 time steps during Phase 4. The size of the buffer is 10^6 at all other times. The number of experiences, $|B_R|$, sampled from the replay buffer is always 64. The discount factor γ is set to 0.99.

DDAS uses a Gaussian noise $\mathcal{N}(0, 0.05)$ during its exploitation phase (DDAS-G) before adopting the Ornstein-Uhlenbeck (Uhlenbeck and Ornstein, 1930) process for its final, fourth stage (DDAS-4S). Unlike Lillicrap et al. (2016), the actor and critic networks are completely separate with no overlap between their parameters. We do not apply any regularization to either network. Table 3.3 lists the configurations

Table 3.3: Hyperparameters specific to different versions of DDAS.

Setting	**Explore Steps**	**Exploit Steps**	**4S Steps**	$Z_{Exploit}$	Z_{4S}
DDAS-NL	500	1,000	0	$U(-10^{-5}, 10^{-5})$	N/A
DDAS-G	500	1,000	0	$\mathcal{N}(0, 0.05)$	N/A
DDAS-4S	500	500	500	$\mathcal{N}(0, 0.05)$	(Uhlenbeck and Ornstein, 1930)

specific to different search algorithms including three different versions of DDAS. Our RL code is based off of Shangtong (2018).

3.4.3 Evaluation Hyperparameters

Once a cell architecture is found and sent for evaluation (testing), the tested network consists of 10 or 20 cells for CIFAR-10 and CIFAR-100, respectively. The channel multiplier present at the beginning of a network is increased to 36. The same cosine annealed SGD with momentum optimizer is used here, except now the learning rate is annealed down to a value of 0 over the course of every experiment, all of which lasted 600 epochs with a batch size of 96. Finally, we also made use of DARTS path dropout feature, with a probability of 0.2, and an auxiliary head with a weight of 0.4.

When further evaluating the best CIFAR-10 architectures for Table 3.1, we re-ran the evaluation experiments with 20 cells. This allowed us to directly compare our results with those of DARTS (Liu et al., 2019). In all experiments, we made use of Cutout (DeVries and Taylor, 2017) using the recommended lengths for CIFAR-10 and CIFAR-100.

3.4.4 Computing Platforms

Workstations used to run our experiments were equipped with Threadripper 2990WX processors, with two exceptions: One computer used a Ryzen 9 3900X, and the other was equipped with a Intel Core i9-9900X. All systems were equipped with dual RTX 2080 Ti GPUs.

Chapter 4

Sequential Optimization of Supernet *then* Policy

In this chapter, we propose Continuous Action-Discrete Architecture Mapping (CADAM) for NAS. Our search algorithm consists of a deterministic reinforcement learning agent based on DDPG (Lillicrap et al., 2016) and the actor-critic framework operating in a continuous space. In particular, CADAM has the following advantages.

First, CADAM produces continuous actions via a DDPG agent which undergo a many-to-one mapping into discrete architectures. This allows the RL policy to be efficiently optimized using gradient ascent and enjoy the same efficiency as other gradient-based optimization algorithms. In the meantime, CADAM incorporates several forms of exploration strategies into the DDPG framework, including the noise-based exploration mechanisms built into DDPG and ϵ-greedy, to ensure that the agent adequately explores a large search space of architectures to avoid premature convergence.

Second, in contrast to previous RL agents (Pham et al., 2018; Bender et al., 2020) in NAS that make discrete actions sequentially to build a neural network over multiple steps, the DDPG actor in CADAM generates one continuous action per step which translates into an entire architecture, and thus does not suffer from the sparse reward problem. Practically speaking, instead of learning a policy to construct a graph step by step, CADAM learns to generate the update rule for architecture parameters using continuous RL.

CADAM diverges from DDAS by training the supernet first and fixing the shared weights during optimization of the RL agent policy. Since no updates are applied to the supernet after initial training, the RL state no longer needs to represent the previously selected architecture. Therefore, we design an actor, which is a neural network, to generate such architecture parameter update rules based on a novel state

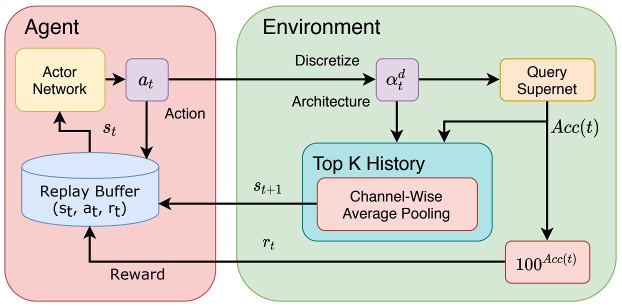


Figure 4.1: A high-level illustration of CADAM

representation that provides statistical information of high-performing architectures in search history, as well as a critic network driven by quantile regression to assess the value of different updates. Thus, CADAM can also be understood as replacing the gradient descent update rule usually used in optimization with an architecture parameter update rule meta-learned by DDPG. However, CADAM is more robust than gradient-based NAS and DDAS as it does not strictly require a supernet to operate and can perform search by quering fixed oracle performance.

We test CADAM and show that it outperforms the best published results on the public benchmark NAS-Bench-201 (Dong and Yang, 2020). Additionally, we evaluate CADAM on supernets trained using CIFAR-10 and a downsampled version of ImageNet (Deng et al., 2009). We show that our scheme produces architecture capable of achieving over 97.4% test accuracy on CIFAR-10 and 75% top-1 accuracy on ImageNet.

4.1 Methodology

We now present the operating mechanisms of the proposed algorithm, CADAM. CADAM relies on a continuous reinforcement learning agent that explores the search space and learns to generate architecture parameters that increasingly focus on high-performing architectures. Every continuous action generated by CADAM is directly discretized into an individual architecture whose reward is instantaneously examined to avoid discretization loss.

4.1.1 The CADAM Agent

A high-level overview of our scheme is illustrated in Figure 4.1. CADAM performs model search using Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016) from reinforcement learning, which operates in a continuous space. The agent interacts with an environment, e.g., a pre-trained supernet, predicted performance or oracle performance if available, over an infinite number of steps, $t = 0, 1, ...,$, by performing an action a_t at each step, receiving a reward r_t in return, followed by transitioning to a state s_t. Different from prior RL-NAS schemes, e.g., ENAS, that learn to construct a high-performing neural network by taking multiple steps of discrete actions, the goal of CADAM is to explore a large search space and learn to output actions given the current state in order to locate the top architectures within a small number of steps.

Specifically, at step t, the agent produces an action that serves as continuous architecture parameters, i.e., $\alpha_t = a_t \in \mathbb{R}^{|E|\times|\mathcal{O}|}$. Each continuous action α_t is mapped into a discrete α_t^d by Algorithm 2, which is actually equivalent to the procedure used by DARTS (Liu et al., 2019) to produce the final discrete architecture for evaluation at the end of search. Algorithm 2 produces $\alpha^d \in \{0,1\}^{|E|\times|\mathcal{O}|}$, which contains discrete entries. Therefore, each α^d corresponds to an individual architecture in the search space.

The reward of action α_t is defined as

$$r_t = 100^{Acc(\alpha_t^d)}, \tag{4.1}$$

where Acc is the accuracy of the architecture α_t^d, in its decimal form, measured by the environment.

During the search, the environment keeps track of the top-K actions seen to date, as determined by the accuracy. That is, we store a history tensor, $h_t \in \{0,1\}^{K\times|E|\times|\mathcal{O}|}$ of top-K α_t^d seen so far and define the state, $s_t \in \mathbb{R}^{|E|\times|\mathcal{O}|}$, as the channel-wise averaging of h_t.

The state s_t is meant to provide statistical information regarding the search space. Given our definition, each entry of the state matrix represents the sample probability that a specific operation is present on an edge in the top-K architectures seen so far. A higher value on an entry indicates that the corresponding operation-edge pair is favoured by high performing architectures.

4.1.2 Quantile-Driven Search Policy

We introduce a few modifications to the original DDPG algorithm to tailor it to our specific search problem, mainly including a quantile loss in critic training.

In particular, the CADAM agent consists of two different neural networks: The actor, $\mu(s_t)$, which generates an action a_t given s_t; and the critic, $Q(a_t)$, which predicts the action value. The agent maintains a replay buffer, R, which stores experience tuples in the form of (s_t, a_t, r_t).

The actor network is defined as,

$$a_t = \mu(s_t) + Z_t, \tag{4.2}$$

where Z_t is a small noise following a uniform distribution $U(-\xi, \xi)$, added to the actor output to encourage exploration. Furthermore, we introduce additional exploration strategies in the form of taking random actions drawn from a uniform distribution, $U(0,1)^{|E|\times|\mathcal{O}|}$, instead of being determined by the actor network in Equation 4.2. We apply two different exploration strategies depending on the search space:

- **ϵ-greedy**: At every step, the actor will take a random action with probability ϵ. We initialize ϵ to a high value and anneal it to a minimum value over time.
- **Random warm-up**: The agent takes random actions in the first W steps. Actions taken during all remaining steps $t > W$ are determined by Equation 4.2.

The critic network of CADAM differs greatly from that of the originally proposed DDPG as it does not take the state as an input,

$$r_t \approx Q(a_t). \tag{4.3}$$

At every time step t, the agent randomly samples a batch B_R from the experiential replay buffer and uses it to update the critic, and then the actor. Furthermore, CADAM updates the critic network using the *check loss* (Koenker, 2005) in quantile regression:

$$\begin{aligned} u_i &= r_i - Q(a_i), \\ \mathcal{L}_{Critic} &= \frac{1}{|B_R|}\sum_{i\in B_R} u_i(\tau - \mathbf{1}(u_i < 0)), \end{aligned} \tag{4.4}$$

where $\tau \in [0,1]$ corresponds to the desired quantile level. The actor network learns directly from the critic with the loss:

$$\mathcal{L}_{Actor} = \frac{1}{|B_R|}\sum_{i\in B_R} Q(\mu(s_i)). \tag{4.5}$$

As Preiss et al. (2020) point out, "NAS and RL problems are not an exact match. RL targets sequential tasks where major challenges are unknown transition dynamics, temporal credit assignment and exploration (Sutton and Barto, 2018)." In DDPG the critic is responsible for interfacing with the environment, which defines the problem. The DDPG critic, originally proposed as $Q(s_t, a_t)$, accommodates issues of RL by calculating a discounted estimate of future rewards (Lillicrap et al., 2016) based on state transitions and learns using the L_2 loss function, formally given by Equation 3.4.

In NAS the only relevant issue mentioned is exploration. Our definition of the state is not related to the accuracy of single architectures. Intuitively speaking, a critic trained by the L_2 loss predicts the mean reward of a_t. In contrast, a critic trained by the check loss is predicting the τth-quantile of the reward. The key advantage is that our critic is capable of picking up and dealing with the best architectures while the DDPG critic is only able to handle average architectures. Put succinctly, our loss function ensures that the critic predicts the tail of the reward, which corresponds to the accuracy of the best architectures selected. This knowledge is then passed onto the actor following Equation 4.5.

Finally, we accelerate policy training by sampling more than one batch from the replay buffer and training the networks multiple times per step. This is useful in situations where the number of steps is tightly budgeted. We determine the number of training cycles C by

$$C = min(\left\lfloor \frac{|R|}{|B_R|} \right\rfloor, C_{max}), \tag{4.6}$$

where $|R|$ is the number of samples in the replay buffer and C_{max} determines the maximum number of cycles. The actor and critic networks start training once the experience buffer has $|B_R|$ samples.

4.1.3 Supernet Pre-training

The CADAM agent interacts with an environment by sending continuous actions and receiving rewards, where the environment can take multiple forms, e.g., oracle performance, accuracy predictor or a weight-sharing supernet. The former is available in public benchmarks, e.g., NAS-Bench-201 (Dong and Yang, 2020), in the form of a look-up table of fully evaluated architectures, while the supernet is helpful when integrating with DARTS and PC-DARTS.

Unlike DARTS that updates all the model weights in the entire supernet during the search, we pre-train the supernet by uniformly sampling individual architectures and only train their corresponding edges in the supernet. This strategy is inspired by random search (Li and Talwalkar, 2020) and could better reflect the conditions present when the formal evaluation is performed on individual architectures. Moreover, backpropagation using this method requires less memory, allowing for larger batch sizes.

One criticism of weight-sharing NAS approaches is that search algorithms are biased towards selecting architectures that are wide and shallow (Shu et al., 2020). This phenomenon occurs because the path the gradient must travel in shallow architectures are shorter than those in topologies that are deep and narrow. Thus, given an

Algorithm 3 Map Continuous α to Discrete Architecture (NAS-Bench-201)

```
1: Input: $\alpha \in \mathbb{R}^{|E|\times|\mathcal{O}|}$
2: Output: $\alpha^d \in \{0,1\}^{|E|\times|\mathcal{O}|}$
3: $\alpha^d = 0^{|E|\times|\mathcal{O}|}$                # Initialize the $\alpha^d$ matrix to zero
4: for $k = 0, 1, .., |E| - 1$ do
5:     $A = \alpha[k, :]$                   # $|E|$ is the number of edges
6:     $i_k = \arg\max_i A_i$
7:     $\alpha^d[k, i_k] = 1$
8: end for
9: Return $\alpha^d$
```

equal number of weights, a wide and shallow architecture will be able to train faster than one that is deeper, giving an illusion of superiority. Our approach avoids this problem by pre-training the supernet separately using a random policy that is blind to performance of different topologies. During the search, the weights of the supernet are fixed.

In particular, we sample discrete architectures from the supernet during for training. For each batch of training data, we first sample a random matrix $\alpha \in \mathbb{R}^{|E|\times|\mathcal{O}|}$, which is then discretized by Algorithm 2 into an architecture α^d, whose weights are updated with the batch of data. Since α^d performs both operation and edge selection, the need for β can be omitted when applied to PC-DARTS. By performing a discrete operation and edge selection, we ensure that the architectures seen by our supernet during training exist in the same space as the evaluation models.

4.2 Experiments and Discussion

In this section, we evaluate CADAM on several benchmarks. First, we demonstrate the merit of our novel RL-NAS agent on NAS-Bench-201 (Dong and Yang, 2020). Next, we train supernets on CIFAR-10 (Krizhevsky, 2009) and a downsampled version of ImageNet (Deng et al., 2009), perform model search and evaluate the performance of the best architectures found.

4.2.1 Oracle Performance

NAS-Bench-201 consists of 15,625 cells architectures. The search space is a downsized variant of DARTS featuring a different operation set, $|\mathcal{O}| = 5$, where each cell only receives input from its predecessor. The topology consists of $|N| = 2$ intermediate nodes and $|E| = 6$ edges. Unlike DARTS, all edges perform operations.

Table 4.1: Accuracies obtained on NAS-Bench-201 datasets compared to other methods. The horizontal line demarcates weight-sharing algorithms from those that directly query oracle information. We run CADAM for 500 and 1,000 steps per experiment across ten different random seeds, and report the mean and standard deviation.

	CIFAR-10		**CIFAR-100**		**ImageNet16-120**	
Method	**Valid** [%]	**Test** [%]	**Valid** [%]	**Test** [%]	**Valid** [%]	**Test** [%]
DARTS	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00
ENAS	37.51 ± 3.19	53.89 ± 0.58	13.37 ± 2.35	13.96 ± 2.33	15.06 ± 1.95	14.84 ± 2.10
GDAS	89.89 ± 0.08	93.61 ± 0.09	71.34 ± 0.04	70.70 ± 0.30	41.59 ± 1.33	41.71 ± 0.98
GAEA	–	94.10 ± 0.29	–	$\mathbf{73.43 \pm 0.13}$	–	46.36 ± 0.00
RS	90.93 ± 0.36	93.70 ± 0.36	70.93 ± 1.09	71.04 ± 1.07	44.45 ± 1.10	44.57 ± 1.25
REA	91.19 ± 0.31	93.92 ± 0.30	71.81 ± 1.12	71.84 ± 0.99	45.15 ± 0.89	45.54 ± 1.03
REINFORCE	91.09 ± 0.37	93.85 ± 0.37	71.61 ± 1.12	71.71 ± 1.09	45.05 ± 1.02	45.24 ± 1.18
BOHB	90.82 ± 0.53	93.61 ± 0.52	70.74 ± 1.29	70.85 ± 1.28	44.26 ± 1.36	44.42 ± 1.49
CADAM-500	$\mathbf{91.36 \pm 0.19}$	$\mathbf{94.11 \pm 0.16}$	$\mathbf{72.47 \pm 0.74}$	72.69 ± 0.58	$\mathbf{46.23 \pm 0.28}$	$\mathbf{46.74 \pm 0.39}$
CADAM-1k	$\mathbf{91.47 \pm 0.15}$	$\mathbf{94.28 \pm 0.08}$	$\mathbf{73.02 \pm 0.52}$	73.09 ± 0.35	$\mathbf{46.58 \pm 0.08}$	$\mathbf{47.03 \pm 0.27}$
Optimal	*91.61*	*94.37*	*73.49*	*73.51*	*46.77*	*47.31*

All cells were evaluated on CIFAR-10, CIFAR-100 and ImageNet16-120 (Chrabaszcz et al., 2017). The inclusion of accuracy metrics across the entire search space allows us to test the CADAM RL agent without a weight-sharing supernet. In this context, our goal is to use the given oracle information to find the highest performing architecture in the least number of steps.

We set $K = 64$, $\tau = 0.9$, $|B_R| = 8$, $\xi = 1e^{-4}$ and $C_{max} = 10$. On NAS-Bench-201, CADAM performs exploration using ϵ-greedy. The initial value of ϵ is 1.0 however it is annealed via cosine schedule to a minimum of 0.05 by step 175. Discretization for NAS-Bench-201 is simpler than it is for DARTS. Formalized by Algorithm 3, the process consists of performing an argmax on each row of $\alpha \in \mathbb{R}^{6\times5}$ to select the operations.

Results and comparison against related works are given by Table 4.1. Also, Figure 4.2 contrasts the final state of a 500 step CADAM experiment against the best architectures queried using exhaustive search.

Table 4.1 demonstrates the state-of-the-art performance of CADAM. In terms of validation set accuracy, CADAM is second to none. For test accuracy, CADAM is only second to GAEA (Li et al., 2020) on one dataset, CIFAR-100. In addition, GAEA relied upon training the supernet from scratch and is not bound by oracle performance. The standard deviation on CADAM is highest on CIFAR-100, but lower than the majority of other algorithms on CIFAR-10 and ImageNet16-120. For

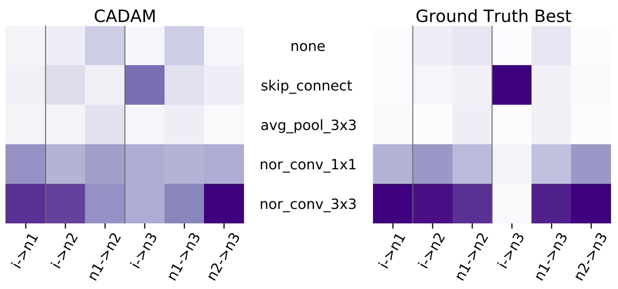


Figure 4.2: Comparison of the state at the end of a CADAM experiment on NAS-Bench-201 CIFAR-100 test accuracy (left) with the average of the absolute top-K best architectures as determined by exhaustive search (right). Rows indicate operations, columns indicate edges. 'i' denotes input, 'n' denotes intermediate nodes and $K = 64$. Darker elements indicate higher values. Vertical bars demarcate intermediate nodes.

500 step tests, CADAM found the absolute best validation and test architectures, by rank, at least once for CIFAR-10, CIFAR-100 and the ImageNet16-120 test set. The third best architecture was found twice for the ImageNet16-120 validation set. In terms of the 1,000 step tests, CADAM found the best architecture at least twice on all datasets, regardless of whether validation or test data was used.

Figure 4.2 shows how close CADAM is to finding the top architectures in the dataset. The average α^d of the top-64 architectures found by CADAM after querying only a fraction of NAS-Bench-201 is very close to that found using exhaustive search. The 'none' and 'avg_pool_3x3' operations are rarely selected in high-performance architectures, while both 'nor_conv' operations are, '3x3' more so than '1x1'. Skip connections should only ever be selected for the fourth edge which connects the cell input to the third and final node. This indicates an extreme preference for a ResNet (He et al., 2016) topology in NAS-Bench-201. Taken together, Table 4.1 and Figure 4.2 establish the advantages of the CADAM RL agent both in terms of results and information regarding a search space.

4.2.2 Supernet Architecture Search

We modify the frameworks of DARTS and PC-DARTS to train our supernets. The topology of both supernets are identical, consisting of $|N| = 4$ intermediate nodes and $|E| = 14$ edges. Algorithm 2 ensures that only 8 edges are activated at a time, matching the constraints of the evaluation space. Our operation space is a subset of DARTS that consists of $|\mathcal{O}| = 7$ potential candidates. Originally both methods feature an operation space with $|\mathcal{O}| = 8$ potential candidates. We omit the 'none' operation during the search, reducing $|\mathcal{O}|$ to 7.

Supernets are trained on CIFAR-10 and ImageNet32-120[1] (Chrabaszcz et al., 2017). CIFAR-10 consists of 50k training samples and 10k test samples. ImageNet32-120 consists of 155k training and 6k test samples, respectively. In both cases, we split the original training set in half into equally sized training and validation partitions. The new training set is used to train the supernet. The validation set is used to query the supernet during the model search. The test set is untouched until the end of the model search when we perform a preliminary gauge the test performance on all architectures in the top-K.

The best architectures and accuracy metrics are not known when using a supernet. Additionally, DARTS dwarfs NAS-Bench-201 by many magnitudes (Siems et al., 2020). Therefore, in this scenario, the goal of CADAM is to explore the search space sufficiently such that a range of high-performing architectures can be stored in the top-K. We set $K = 500$, $\tau = 0.95$, $|B_R| = 64$, $\xi = 5e^{-5}$ and $C_{max} = 1$. We run the agent for 20k steps on each supernet and limit the replay buffer to contain the last 5k experiences. Exploration is achieved using random warm-up with $W = 3000$.

Figure 4.3 provides an analog to Figure 4.2 for PC-DARTS. It should be noted that elements corresponding to operation-edge pairs leading into nodes 2 and 3 will have smaller values than those leading into nodes 0 and 1. This is because there are more candidate operation-edge pairs competing to connect to the deeper nodes, yet the number of operation-edge pairs that can lead into each node remains constant at 2. Put more simply, in every case one element from column 'k-2 -> 0' is guaranteed to be chosen. The same holds for column 'k-1 -> 0', but not for any other column. This shows that high-performing architectures are not limited to specific topologies.

Analyzing Figure 4.3, the normal cell shows a strong preference for routing convolution operations into the first two nodes and feeding the output from node 0 to node 1. Average pooling operations are more common along the edges that feed into nodes 2 and 3, while maximum pooling and skip connections are rare. The reduction cell is less discriminating. Most operations can occupy the start of the cell. A small preference for separable convolutions and pooling operations is shown for nodes 2 and 3. Skip connections are strongest when linking the reduction cell input and the first two nodes but rarely appear elsewhere.

Figure 4.4 illustrates the best cell architectures found during the same experiment that produced Figure 4.3. The structure of these cells are in agreement with the average depiction given by Figure 4.3. With exception to a lone average pooling operation, the normal cell is dominated by numerous convolution operations and the

[1]First 120 classes of ImageNet (Deng et al., 2009) downsampled to 32x32 images using the 'box' method.

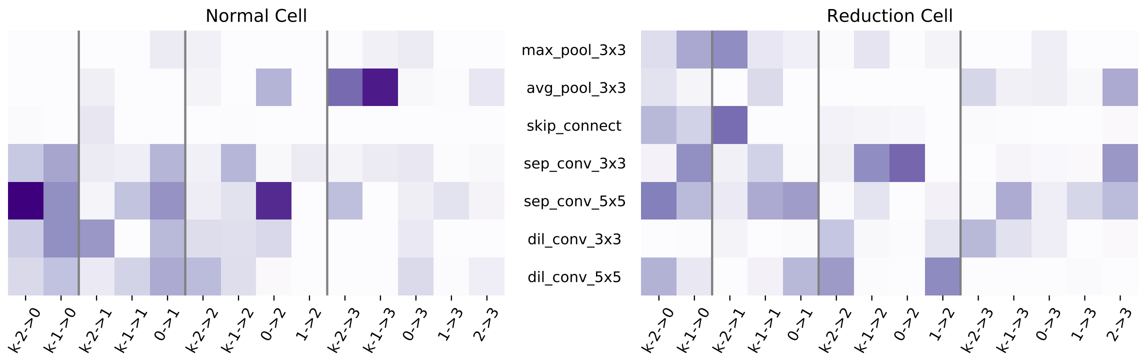
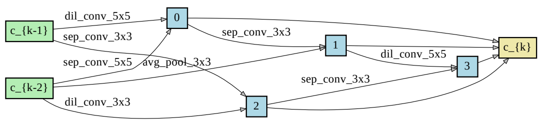
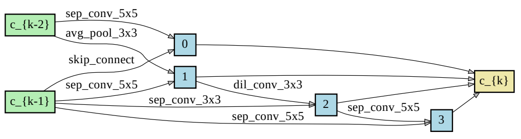


Figure 4.3: Final state of CADAM on CIFAR-10 using a PC-DARTS supernet. Rows indicate operations, columns indicate edges. ‘k’ stands for cell input. Lone numbers denote nodes and $K = 500$. Darker elements indicate higher values. Vertical bars demarcate boundaries between nodes where operation-edge pairs compete.



(a) Normal Cell, 2.5c, 4



(b) Reduction Cell, 3c, 4

Figure 4.4: Cell architectures found by running PC-CADAM on CIFAR-10, annotated with width and depth metrics defined by Shu et al. (2020).

‘sep_conv_5x5’ operation connecting input $k-2$ to node 0 is clearly represented. The reduction cell is similar, with exception to the skip connection between $k-1$ and node 1. Figure 4.3 shows that there is some preference to connect separable convolutions into node 3 of the reduction cell and this preference is reflected in the choice of operators for that reduction cell node.

4.2.3 Supernet Evaluation and Comparison

We now evaluate the best architectures found by CADAM on two well-known benchmark datasets, CIFAR-10 (Krizhevsky, 2009) and ImageNet (Deng et al., 2009). Given that the “goal of NAS is to find the optimal architecture which produces the best performance on the test set” Xie et al. (2020), we do not determine the best architecture using validation set performance. However, as K is large when using a DARTS-like supernet, formally evaluating each architecture in the top-K would be computationally expensive. Instead, for every discrete architecture in the environment history at the end of search, we assign the α^d of said architecture to the supernet. The preliminary *test accuracy* of each architecture is then measured using

Table 4.2: Evaluation of CADAM architectures with other NAS algorithms on CIFAR-10. The horizontal bar separates algorithms that evaluate on the DARTS search space from those with different search spaces. Prefixes denote the variant of DARTS search space.

Architecture	**Test Acc. [%]**	**Params**
ENAS	97.11	4.6M
GDAS	97.18	2.5M
AlphaX	97.46 ± 0.06	2.8M
ProxylessNAS	97.92	5.7M
DARTS 1st	97.00 ± 0.14	3.3M
DARTS 2nd	97.24 ± 0.09	3.3M
SNAS	97.15 ± 0.02	2.8M
P-DARTS	97.50	3.4M
PC-DARTS	97.43 ± 0.07	3.6M
PC-SDARTS	97.51 ± 0.04	3.5M
P-SDARTS	97.52 ± 0.02	3.4M
PC-GAEA	97.50 ± 0.06	3.7M
CADAM	97.46 ± 0.09	3.7M
PC-CADAM	97.42 ± 0.08	3.8M

the supernet. To combat the optimization gap present in weight-sharing NAS, the architecture in the final top-K that produces the highest preliminary test accuracy is considered the best. Validation accuracy is still meaningful in this context as the metric which determines which architectures are stored within the top-K.

Table 4.2 lists results of CADAM architectures on CIFAR-10 against other relevant algorithms. In terms of both accuracy and model parameters, the found architectures are on-par with the current state-of-the-art for DARTS. It is interesting how the CADAM architectures not only outperform second-order DARTS, but are on-par with PC-DARTS and P-DARTS. While GAEA and SDARTS modify the gradient-descent mechanisms that perform the model search, the novelty of PC-DARTS and P-DARTS pertain to improvements made to the structure of the original search model. By achieving performance that is on-par with these schemes, we demonstrate the validity of our supernet training procedure as a means of improving the original search model without inserting constraints on how many times a candidate operation may be selected.

Finally, as shown by Table 4.3, we formally evaluate all relevant architectures on ImageNet. The architecture found using DARTS on CIFAR-10 performed the worst and barely passed 75% top-1 accuracy. The architecture found by CADAM using the proxy subset, ImageNet32-120, outperformed the DARTS architecture in terms

Table 4.3: Evaluation of CADAM architectures with other NAS algorithms on ImageNet. Prefixes denote the variant of DARTS search space; suffix abbreviations indicate dataset used to train the supernet; e.g. 'C10' for CIFAR, 'IN' for ImageNet.

Architecture	**Top-1 Acc.** [%]	**Top-5 Acc.** [%]	**Params**
DARTS-C10	73.3	91.3	4.7M
SNAS-C10	72.7	90.8	4.3M
P-DARTS-C10	75.6	92.6	4.9M
P-DARTS-C100	75.3	92.5	5.1M
PC-DARTS-C10	74.9	92.2	5.3M
PC-DARTS-IN	75.8	92.7	5.3M
PC-SDARTS-C10	75.7	92.6	-
P-SDARTS-C10	75.8	92.8	-
PC-GAEA-C10	75.7	92.7	5.3M
PC-GAEA-IN	76.0	92.7	5.6M
CADAM-C10	75.0	92.3	5.3M
PC-CADAM-C10	75.5	92.5	5.4M
PC-CADAM-IN	75.1	92.3	5.0M

of top-1 accuracy tied with it in terms of top-5 accuracy, while using 300k fewer parameters.

Our best ImageNet architecture was found using the PC-DARTS search model. With top-1 accuracy and top-5 accuracy metrics of 75.5% and 92.5%, respectively, this result is in the same neighborhood as recent DARTS-based algorithms.

4.3 Conclusions

In this chapter, we propose CADAM, a continuous actor-critic reinforcement learning algorithm for differentiable neural architecture search. Based on the DDPG algorithm, CADAM produces continuous actions that are translated into discrete architectures using a many-to-one mapping. A reward is then computed from the accuracy of an architecture. Through a number of innovations on state representation, actor and critic network design, quantile optimization losses and exploration strategies, CADAM learns to explore a large search space and achieves fast convergence to high-performing architectures in fewer number of validations.

Experiments show that CADAM outperforms the best published results on NAS-Bench-201 after querying only 500 or 1000 architectures. When applied to DARTS and PC-DARTS, CADAM produces architectures that achieve competitive test accuracies on CIFAR-10 and ImageNet relative to other state-of-the-art algorithms using the same search space.

Table 4.4: Comparison of validation and preliminary test architectures for the best architectures found during model search. 'PC' prefix indicates a supernet based on PC-DARTS. Suffixes denote whether the architecture produced the best validation or test accuracy. 'Rank' indicates the place in the top-K. The lower the rank, the higher the validation accuracy.

Dataset and Supernet	**Validation Acc. [%]**	**Test Acc. [%]**	**Rank**
CIFAR-10-Val	88.90	91.81	1
CIFAR-10-Test	87.87	92.25	416
PC-CIFAR-10-Val	87.08	87.78	1
PC-CIFAR-10-Test	86.52	88.04	129
PC-ImageNet32-120-Val	49.71	45.38	1
PC-ImageNet32-120-Test	49.43	46.45	8

4.4 Additional Experimental Details

We provide additional information regarding accuracies recorded during model search. We also list hyperparameters for the supernet training, RL search and formal evaluation stages of our experiments. Finally, we describe the computational resources used to perform experiments.

4.4.1 Validation and Test Ranking

Table 4.4 compares validation and preliminary test accuracies for supernets. Validation accuracy serves as an intermediary when judging the performance of all sampled architectures, but the final decision does not hinge on it. Test accuracy should only be invoked sparingly outside of formal evaluation. We believe that our usage of it, as a way of determining the best of the best architectures from a vetted pool of candidates, is one of these scenarios.

4.4.2 Search Hyperparameters

CIFAR-10 supernets trained for 10,000 epochs using a batch size of 250, while ImageNet supernets trained for 5,000 epochs using a batch size of 750. All supernets consisted of 8 cells and 16 initial channels The initial learning rate was set to 0.025 and was annealed down to $1e^{-3}$ by cosine schedule (Loshchilov and Hutter, 2017). Stochastic gradient descent with a momentum factor of 0.9 is used to optimize the weights. DARTS supernets trained using cutout, however PC-DARTS supernets were not. We did not apply any preprocessing techniques to ImageNet32-120.

4.4.3 Agent Hyperparameters

The CADAM actor and critic networks are both multi-layer perceptrons with 3 hidden layers. The number of neurons in each layer is $(256, 256, 256)$ when operating on a DARTS-like supernet and $(128, 256, 128)$ when quering NAS-Bench-201. ReLU (Nair and Hinton, 2010) serves as the activation function in the hidden layers while the final layer of the actor network features a sigmoid activation and the critic uses an identity function as it produces a scalar. Both networks are optimized using Adam (Kingma and Ba, 2014) with $\vec{\beta} = (0.9, 0.99)$. Learning rates for the actor and critic networks are $1e^{-8}$ and $1e^{-4}$, respectively. Supernet training and search are done separately, but initialized with a random seed of 2 each. The seeds for NAS-Bench-201 trials were in $[1, 10]$.

4.4.4 Evaluation Hyperparameters

We evaluated architectures on CIFAR-10 using models with 20 cells with an initial channel size of 36. All models were evaluated using Cutout (DeVries and Taylor, 2017) using the recommended length for CIFAR-10, as well as an auxiliary head with a weight of 0.4. The initial learning rate is 0.025 and this value is annealed down to zero following a cosine schedule over 1,000 epochs.

The DARTS supernet (listed as 'CADAM') architecture was trained with a batch size of 80 and a drop path probability of 0.1. The PC-DARTS architecture ('PC-CADAM') trained with a batch size of 96 and a drop path probability of 0.2. The listed batch sizes are the maximum we could fit onto a single 11GB NVIDIA RTX 2080 Ti. Finally accuracies consist of the mean and standard deviation across 5 different random seeds: 0, 1, 2, 3 and 4.

ImageNet evaluations were performed using the same hyperparameters as PC-DARTS (Xu et al., 2019). The network consists of 14 cells and is trained for 250 epochs. An initial learning rate of 0.5 is used and is annealed down to zero after an initial 5 epochs of warmup.

4.4.5 Computing Platforms

NAS-Bench-201, supernet training, model search, and some CIFAR-10 evaluations were performed using GPU workstations equipped with dual RTX 2080 Ti GPUs and AMD Ryzen Threadripper 2990WX CPUs. The remaining CIFAR-10 and all ImageNet evaluations were performed on a GPU server with 8 Tesla V100 32GB GPUs and an Intel Xeon Gold 6140 GPU.

Chapter 5

Conclusion

We finalize the contents of this thesis by summarizing the findings of our parallel and sequential optimization methods, DDAS and CADAM. Then, we provide a brief discussion on vectors for future research based on the presented work in relation to recent literature in the fields of Neural Architecture Search and Reinforcement Learning, respectively.

5.1 Contributions

In this thesis we propose the use of continuous RL search agents to the problem space of NAS in two distinct ways. We primarily apply our RL search algorithms to the domain of weight-sharing supernets. Leveraging a continuous agent, we are able to generate and evaluate one architecture per step. By decoupling the search space and search algorithm sufficiently, and using a continuous-to-discrete mapping, we are able to train both components in an optimal fashion without inducing additional performance compromises.

Our first scheme, DDAS, optimizes the supernet and RL agent in parallel. The agent determines which weights the supernet will update, while the reward returned by the supernet influences future decisions the agent will make. In this situation both components are sufficiently separated such that they only see the output of one another, not the internal mechanisms used to generate the output. Additionally, we make use of Pareto frontiers to investigate the performance of architectures generated by our scheme against computational constraints such as FLOPS or number of parameters. Experimental results show that our first scheme explores different areas of the search space while focusing on the most critical locations. This exploration results in DDAS selecting architectures from a range of topologies with a varying number of FLOPS or parameters. When evaluated, the highest performing architectures on the

Pareto frontiers achieve competitive performance on CIFAR-10 and CIFAR-100.

Next, we introduce CADAM, whose search procedure consists of training the supernet and agent sequentially. First, the supernet is rigorously optimized by random sampling that does not permit bias towards specific topologies. Second, the agent produces continuous actions which are translated into discrete representations. These representations are used to query the performance of a specific architecture using the supernet. This methodology allows us to swap out the supernet entirely for an oracle accuracy predictor, such as those provided by public NAS benchmarks. Therefore, we can readily test the performance of our agent on different search spaces. We make use of a quantile regression loss function to ensure that our agent focuses on learning from the architectures at the tail of the accuracy distribution. Additionally, we propose a novel RL state definition that tracks the distribution of high-performing architectures in the search space. This state can be extracted to provide information on which parts of an architecture generally contribute to a high performance overall. Experiments show that CADAM achieves state-of-the-art performance on NAS-Bench-201. Relative to other schemes using the same search space, we achieve competitive performance on CIFAR-10 and ImageNet.

In conclusion, we investigate the problem of weight-sharing NAS using continuous RL. We first apply the agent to the search space in a parallel way synonymous with gradient-based solutions. Next, we separate the training of these components into sequential steps. Both schemes yield respectable results on several benchmarks and we demonstrate both the applicability and usefulness of continuous deterministic reinforcement learning in neural architecture search.

5.2 Future Work

Based on our findings and experiences, we propose the following research avenues,

- DARTS is considered a large search space as it contains approximately 10^{18} different architectures (Siems et al., 2020). Despite this size, it has been shown that random search policies are capable of competitive performance relative to guided policies (Li and Talwalkar, 2020), such as differentiable or RL methods. TuNAS (Bender et al., 2020) refuted this claim, by showing that the performance of random search drops substantially when the search space is *very large* - on the order of 10^{28} to 10^{43}, specifically. Therefore, one avenue for future research would be to deploy our proposed algorithms on similar search spaces to truly test the performance of our RL agent.

- Preiss et al. (2020) categorized RL-NAS methods into three distinct types, *Bandit*, *Editor* and *Tree*. Both schemes presented in this thesis most closely align with the bandit framework, while other RL-NAS methods (Zoph and Le, 2017; Pham et al., 2018; Wang et al., 2020) are tree-based. The scheme presented in Chapter 4 could be modified to fit the editor framework by replacing the state with an architecture representation sampled using information from the top-K. Just as the state is the average of the top-K, the standard deviation could be calculated and sampling could be performed using techniques borrowed from Variational Auto-Encoders (Kingma and Welling, 2019). The agent would then be responsible for proposing an *edit* to this architecture. The reward would be defined by the accuracy difference between the original architecture and the edited version.

- Apply the proposed frameworks to the macro-search setting. In this situation the search space consists of cells with fixed architectures, and the search process consists of finding the optimal ordering of these cells.

- CIFAR-10 and ImageNet are the most popular benchmark datasets for NAS, however algorithms that train on them do not generalize to other datasets. Therefore, we would apply the proposed frameworks to non-standard benchmark datasets, like the ones used by Yang et al. (2019).

- It would be interesting to see how DDAS and CADAM would change if the underlying DDPG agent was replaced with Twin Delayed Deep Deterministic policy gradient (TD3) (Fujimoto et al., 2018), which was originally proposed to address function approximation issues with the critic network.

- Using a multi-GPU setup, it would be possible to generate a multi-actor algorithm. In this scheme, the agent would still consist of an actor-critic framework. However, there would be multiple actor networks, each one corresponding to a different supernet on separate GPUs, that may potentially be different sizes or optimizing on different datasets. There would be one or more critic networks, potentially serving as an ensemble, or instantiated in accordance with the number of datasets used, that would be used to teach the actor networks. Using this scheme it may be possible to generalize or discriminate the results of the DDAS Pareto frontiers or CADAM states across different sizes of supernets or types of datasets.

Bibliography

Bender, Gabriel, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le (2018). "Understanding and simplifying one-shot architecture search". In: *International Conference on Machine Learning*, pp. 550–559.

Bender, Gabriel, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le (2020). "Can Weight Sharing Outperform Random Architecture Search? An Investigation With TuNAS". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14323–14332.

Cai, Han, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han (2020). "Once for All: Train One Network and Specialize it for Efficient Deployment". In: *International Conference on Learning Representations*.

Cai, Han, Ligeng Zhu, and Song Han (2019). "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware". In: *International Conference on Learning Representations*.

Chen, Xiangning and Cho-Jui Hsieh (2020). "Stabilizing Differentiable Architecture Search via Perturbation-based Regularization". In: *International Conference on Machine Learning*.

Chen, Xin, Lingxi Xie, Jun Wu, and Qi Tian (2019). "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1294–1303.

Chrabaszcz, Patryk, Ilya Loshchilov, and Frank Hutter (2017). "A downsampled variant of imagenet as an alternative to the cifar datasets". In: *arXiv preprint arXiv:1707.08819*.

Chu, Xiangxiang, Bo Zhang, Jixiang Li, Qingyuan Li, and Ruijun Xu (2019). "Scarletnas: Bridging the gap between scalability and fairness in neural architecture search". In: *arXiv preprint arXiv:1908.06022*.

Dai, Xiaoliang, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. (2020). "FBNetV3: Joint Architecture-Recipe Search using Neural Acquisition Function". In: *arXiv preprint arXiv:2006.02049*.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

DeVries, Terrance and Graham W Taylor (2017). "Improved Regularization of Convolutional Neural Networks with Cutout". In: *arXiv preprint arXiv:1708.04552*.

Dong, Xuanyi and Yi Yang (2019). "Searching for a robust neural architecture in four gpu hours". In: *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pp. 1761–1770.

— (2020). "Nas-bench-201: Extending the scope of reproducible neural architecture search". In: *International Conference on Learning Representations*.

Duan, Yan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel (2016). "Benchmarking deep reinforcement learning for continuous control". In: *International Conference on Machine Learning*, pp. 1329–1338.

Elsken, Thomas, Jan Hendrik Metzen, and Frank Hutter (2018a). "Efficient multi-objective neural architecture search via lamarckian evolution". In: *arXiv preprint arXiv:1804.09081*.

— (2018b). "Neural architecture search: A survey". In: *arXiv preprint arXiv:1808.05377*.

Fujimoto, Scott, Herke Van Hoof, and David Meger (2018). "Addressing function approximation error in actor-critic methods". In: *International Conference on Machine Learning*.

Guo, Zichao, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun (2020). "Single path one-shot neural architecture search with uniform sampling". In: *European Conference on Computer Vision*. Springer, pp. 544–560.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *arXiv preprint arXiv:1801.01290*.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Jin, Xiaojie, Jiang Wang, Joshua Slocum, Ming-Hsuan Yang, Shengyang Dai, Shuicheng Yan, and Jiashi Feng (2019). *RC-DARTS: Resource Constrained Differentiable Architecture Search*. arXiv: 1912.12814 `[cs.CV]`.

Kingma, Diederik and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*.

Kingma, Diederik P and Max Welling (2019). "An introduction to variational autoencoders". In: *arXiv preprint arXiv:1906.02691*.

Koenker, Roger (2005). *Quantile Regression*. Econometric Society Monographs no. 38. Cambridge University Press. ISBN: 9780521845731.

Krizhevsky, Alex (2009). *Learning multiple layers of features from tiny images.* Tech. rep.

Li, Liam, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar (2020). “Geometry-Aware Gradient Algorithms for Neural Architecture Search”. In: *arXiv preprint arXiv:2004.07802.*

Li, Liam and Ameet Talwalkar (2020). “Random search and reproducibility for neural architecture search”. In: *Uncertainty in Artificial Intelligence.* PMLR, pp. 367–377.

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2016). “Continuous control with deep reinforcement learning”. In: *International Conference on Learning Representations.*

Liu, Hanxiao, Karen Simonyan, and Yiming Yang (2019). “DARTS: Differentiable Architecture Search”. In: *International Conference on Learning Representations.*

Loshchilov, Ilya and Frank Hutter (2017). “SGDR: Stochastic Gradient Descent with Warm Restarts”. In: *ICLR.*

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). “Playing Atari With Deep Reinforcement Learning”. In: *NIPS Deep Learning Workshop.*

Nair, Vinod and Geoffrey E. Hinton (2010). “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning.*

Pham, Hieu, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean (2018). “Efficient neural architecture search via parameter sharing”. In: *International Conference on Machine Learning.*

Preiss, James, Eugen Hotaj, and Hanna Mazzawi (2020). “A Closer Look at Reinforcement Learning for Neural Network Architecture Search”. In: *International Conference on Learning Representations.*

Real, Esteban, Alok Aggarwal, Yanping Huang, and Quoc V Le (2019). “Regularized evolution for image classifier architecture search”. In: *Proceedings of the aaai conference on artificial intelligence.* Vol. 33, pp. 4780–4789.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347.*

Shangtong, Zhang (2018). *Modularized Implementation of Deep RL Algorithms in PyTorch.* https://github.com/ShangtongZhang/DeepRL.

Shu, Yao, Wei Wang, and Shaofeng Cai (2020). “Understanding Architectures Learnt by Cell-based Neural Architecture Search”. In: *International Conference on Learning Representations.*

Siems, Julien, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter (2020). “NAS-Bench-301 and the Case for Surrogate Benchmarks for Neural Architecture Search”. In: *arXiv preprint arXiv:2008.09777*.

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller (2014). “Deterministic policy gradient algorithms”. In: *International Conference on Machine Learning*.

So, David R, Chen Liang, and Quoc V Le (2019). “The evolved transformer”. In: *arXiv preprint arXiv:1901.11117*.

Stamoulis, Dimitrios, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu (2019). “Single-path nas: Designing hardware-efficient convnets in less than 4 hours”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 481–497.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Tan, Mingxing, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le (2019). “Mnasnet: Platform-aware neural architecture search for mobile”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828.

Uhlenbeck, George E and Leonard S Ornstein (1930). “On the theory of the Brownian motion”. In: *Physical review* 36.5, p. 823.

Wan, Alvin, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. (2020). “Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12965–12974.

Wang, Linnan, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca (2020). “Neural architecture search using deep neural networks and monte carlo tree search”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 06, pp. 9983–9991.

Williams, Ronald J (1992). “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4, pp. 229–256.

Wilson, Ashia C, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht (2017). “The marginal value of adaptive gradient methods in machine learning”. In: *Advances in neural information processing systems*, pp. 4148–4158.

Wu, Bichen, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer (2019). “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742.

Xie, Lingxi, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Zhengsu Chen, Lanfei Wang, An Xiao, Jianlong Chang, Xiaopeng Zhang, et al. (2020). "Weight-Sharing Neural Architecture Search: A Battle to Shrink the Optimization Gap". In: *arXiv preprint arXiv:2008.01475*.

Xie, Sirui, Hehui Zheng, Chunxiao Liu, and Liang Lin (2018). "SNAS: stochastic neural architecture search". In: *arXiv preprint arXiv:1812.09926*.

Xu, Yuhui, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong (2019). "Pc-darts: Partial channel connections for memory-efficient architecture search". In: *International Conference on Learning Representations*.

Xu, Yuhui, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Bowen Shi, Qi Tian, and Hongkai Xiong (2020). "Latency-Aware Differentiable Neural Architecture Search". In: *arXiv preprint arXiv:2001.06392*.

Yang, Antoine, Pedro M Esperança, and Fabio M Carlucci (2019). "NAS evaluation is frustratingly hard". In: *arXiv preprint arXiv:1912.12522*.

Ying, Chris, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter (2019). "Nas-bench-101: Towards reproducible neural architecture search". In: *International Conference on Machine Learning*, pp. 7105–7114.

Yu, Jiahui, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le (2020a). "Bignas: Scaling up neural architecture search with big single-stage models". In: *arXiv preprint arXiv:2003.11142*.

Yu, Kaicheng, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann (2020b). "Evaluating The Search Phase of Neural Architecture Search." In: *International Conference on Learning Representations*.

Zela, Arber, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter (2020). "Understanding and robustifying differentiable architecture search". In: *International Conference on Learning Representations*.

Zoph, Barret and Quoc V Le (2017). "Neural architecture search with reinforcement learning". In: *International Conference on Learning Representations*.

Zoph, Barret, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le (2018). "Learning transferable architectures for scalable image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710.