Experimentally Assessing RL based NAS Controllers

Rohan Jain, Kanav Singla, Benjamin Zhuo
University of Toronto
{rohan.jain,kanav.singla,benjamin.zhuo}@mail.utoronto.ca

Abstract

Neural networks are powerful and flexible models that work well for many complicated tasks such as computer vision, text & speech processing, combinatorial optimization problems and etc. "AI that creates AI", was a common headline of what people initially described as Automated Machine Learning (i.e. Auto ML). This novel concept gained attraction in 2016 when Google Brain released their first "Neural Architecture Search with Reinforcement Learning (NAS)". Traditionally, choosing a neural network architecture manually is a tiring, inefficient & computationally expensive task. Even the standard NAS is very computationally expensive as it required over 450 GPU's for 3-4 days to train on CIFAR-10. We are analyzing & reproducing the method of standard Neural Architecture Search (NAS) through an existing improved method known as "Efficient Neural Architecture Search via Parameter Sharing (ENAS)". We are assessing the quality of the ENAS architectures' RNN controller using different experimental techniques such as **search space poisoning and random search to see how the controller performs when decoupled with strong search spaces**.

1 Introduction

In NAS, the RNN controller (Diagram: Appendix A) samples a "child" network's architecture as a network description, trains it from scratch until convergence, & calculates its accuracy as a reward signal. We then compute the policy gradient to update the controller. Process is repeated for many iterations, for the controller to learn to improve its search. To improve the computational complexity of standard NAS, we explore NAS via parameter sharing (i.e ENAS). ENAS constructs a large computational graph, where each subgraph represents a neural network architecture, hence forcing all architectures to share their parameters. Sharing parameters among child models allows ENAS to reduce GPU hours by over 1000x compared to standard NAS.

2 Related Works

Hyperparameter optimization is an existing topic in machine learning that has been researched before (Barret Zoph, Quoc V. Le, 2017;). Despite this, previous solutions suffered from problems such as long compute times (Zoph et al, 2017). Neural architecture search (NAS) has been applied to language modelling & image classification (Zoph et al, 2017). In NAS, an RNN controller is trained using a loop. The controller samples a child architecture, trains to convergence & then measures the performance of the child architecture on the task. This performance is then used to find more child architectures, through which this process is repeated many times. However the concerns about NAS are that it is computationally expensive & takes a lot of time to perform. For Zoph et al, the number of controller replicas was 100, the number of child replicas was 8, meaning they had 800 networks trained on 800 GPUs at any given time.

Another approach to the problem is described in Luo et al (2019) as neural architecture optimization. In this optimization based approach, an encoder is used to map a neural architecture to a continuous representation where a regression model is built on, and then evaluate the performance of. Notably, they use gradient descent to optimize the module to obtain continuous representations of networks. Their experiments took less than 10 hours on 1 GPU.

3 Method

3.1 Generating a CNN child description with controller RNN

A controller is used to generate hyperparameters of neural networks. The controller is implemented as an RNN for flexibility. Consider sampling a convolutional network. The RNN samples the hyperparameters of a child CNN description as a list of tokens. At each layer we predict filter height, stride height, stride width, and number of filters with a softmax classifier and then fed as inputs to the next layer. (Diagram: Appendix C) After an architecture is generated, a neural network is built and trained with that architecture. We record the accuracy of the new network on a validation set and use the accuracy to optimize the parameters of the controller RNN (θ_c) in order to maximize the expected validation accuracy.

3.2 Training with REINFORCE

The list of tokens that the controller predicts are fed as a list of actions $a_{1:T}$ for designing a child architecture. Suppose the child architecture achieves validation accuracy R. The idea is to use the accuracy R as a reward signal and use reinforcement learning to train the controller. The maximized expected reward of the controller is represented by $\mathcal{J}(\theta_c)$:

$$\mathcal{J}(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

Since R is non-differentiable, we use iterative policy gradient (PG) method to optimize and update θ_c. In the PG, the probability of choosing an action (architecture) with higher reward in the next time step is increased, meanwhile the probability of choosing actions with lower reward is decreased, thus the RNN controller generates better architectures over time. The REINFORCE rule from Williams (1992) is given by:

$$\begin{aligned}\nabla_{\theta_c}\mathcal{J}(\theta_c) &= \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}[\nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R] \\ &\approx \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \nabla_{\theta_c} logP(a_t|a_{(t-1):1};\theta_c)R_k\end{aligned}$$

With m being the number of architectures in one batch, T is the number of hyperparameters, R_k is the validation accuracy of the k^{th} neural network. In effort to reduce the variance of this estimate as our previous update had high variance, we add a baseline function, b ; is an exponential moving average of the previous architecture accuracies:

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \nabla_{\theta_c} logP(a_t|a_{(t-1):1};\theta_c)(R_k - b)$$

The execution of this process takes several hours to train the networks. In effort to **accelerate the training with parallelism**, the processing of the search is distributed over S (# of servers). Each server has K controller replicas where each K_i replica samples m different child architectures which are trained in parallel. The accuracy of each child model is recorded to compute the gradients w.r.t θ_c, which is then sent to the according server. In the end, the action with highest reward is considered as the best candidate. (Diagram: Appendix D)

3.3 Improvement with parameter sharing

The computational burden of NAS is that once we train the architectures we "throw" them away after computing an reward. All of the architectures that NAS eventually iterates over can be noticed as sub-graphs of a larger graph, specifically the NAS search space is represented via a single directed acyclic graph (DAG). In, ENAS the child models are superpositioned into a DAG (hence all child models share their parameters. (Diagram: Appendix B) The nodes represent local computations & edges represent flow of information. At each step, sample a different architecture and train them continuously. We use the architecture to update the shared parameters and use the reward computed on a subset of the validation set to update the controller with reinforcement learning. The controller tries to sample a "good" subgraph C from graph G, the training process is split up into two alternating phases. (Diagram: Appendix E & F)

4 Experiments

We are trying to assess the quality of the controller to see if these previous implementations actually solve the problem of Neural Architecture Search. This can be done by decoupling the effect of strong ENAS architecture, from good search space designs using two different experiments.

4.1 Macro Search Space

In the search space, the controller has to make 2 decisions at each block:

1. The previous nodes this layer connects to (skip connections)
2. The computation operation to use

At each layer k, the previous $k-1$ layers can have a connection to it. In the original search space design for ENAS, the controller has 6 operations available:

- 3×3 convolution (separable and non-separable)
- 5×5 convolution (separable and non-separable)
- Max and average pooling with kernel size 3×3

4.2 Search Space Poisoning

We are poisoning the search space by designing it to be less powerful so that the controller has very limited (and relatively worse) subset to pick its actions from. We create multiple different "poisonings" and make the ENAS architecture compatible to perform its search over this new search space. Then we assess the performances of the best architecture the controller comes up with.

Table 1 - Results:

#	Search Space Design	Average Validation Accuracy	Average Test Accuracy
1	3×3 convolution 3×3 separable convolution 5×5 convolution 5×5 separable convolution Average pooling Max Pooling	0.7981	0.7541
2	5×5 convolution 5×5 separable convolution Average Pooling	0.7770	0.7259
3	3×3 convolution 5×5 convolution Max Pooling	0.1250	0.1125
4	3×3 convolution 3×3 separable convolution 5×5 convolution 5×5 separable convolution Average pooling	0.7845	0.7354
5	5×5 convolution Average pooling Maximum pooling	0.7685	0.7104

Note that these accuracies are the average performances of the 10 architectures sampled by the controller after 30 epochs of training on the poisoned search spaces (denoted as **2-5**)

Interpreting the Results:
During training our controller samples architectures consisting of 12 layers. (Diagram: Appendix G)

The first entry in each layer represents the type of operation we perform at each branch. By default, we have 0 - Convolution branch with kernel size 3, 1 - Separable convolution branch with kernel size 3, 2 - Convolution branch with kernel size 5, 3 - Separable convolution branch with kernel size 5, 4 - Average pooling branch, 5 - Max pooling branch. The rest of the entries in each layer correspond to if a skip connection exists between that layer and the n^{th} layer.

Analysis: Here we present the graphs for average accuracy and loss for one of our poisoned search space design #3:

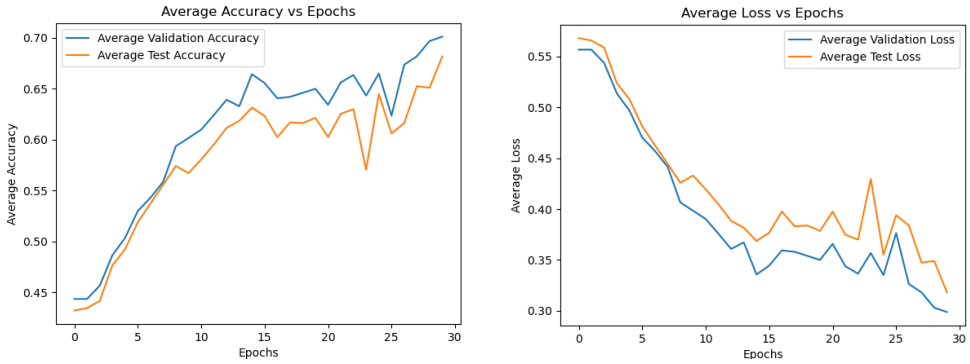

Figure 2: Left: average accuracy, Right: average loss

Looking at the results [1], we can say that the controller performs sub par when given limited options using the poisoned search space designs. For 3 out of the 4 poisoned search spaces, the validation and test accuracies are within 10% of the original ENAS search space (#1). This validates that the controller actually learns to make these decisions and the empirical performance is not just due to a good search space design. However, within the trained models, we can see a general trend of better accuracies with better search space design. Still this difference in accuracies is pretty small in regards to the aggressive poisoning of the search space. Looking at our results from model 3, we can see that the accuracy is incredibly low compared to the other models. This model lacked separable layers, which seems to be the reason why the accuracy achieved was so low.

4.3 Guided Random Search

Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. In ENAS, a recurrent cell, an CNN, and a pair of convolutional & reduction cells are sampled uniformly from their search spaces. They are trained to convergence using the same settings determined by ENAS architectures. The convolutional network reached a 5.86% test error, two random cells reached a 6.77% test error on CIFAR-10, whlist ENAS achieved 4.23% & 3.54% test errors, respectively. There is an average increase of $\approx 62\%$ in test errors. Hence, for this experiment we clearly see the increase in performance when using ENAS as opposed to random search. Thus ENAS is capable of finding good architectures.

5 Conclusion

The development of Neural Architecture Searching has made important advances in automating & designing neural networks. We can see that even after poisoning the search space, the controller was able to come up with a decent architecture that can classify different images of CIFAR-10 with a decent accuracy. Moreover, we see that the model performs rather poorly when the decisions are made through Guided Random Search. Hence by decoupling the effect of controller from a good search space design (Experiment 1) and taking informed decisions (through Experiment 2), we conclude that the the controller for our assessed algorithm, ENAS depicted that it actually learns and solves the problem of Neural Architecture Searching.

[1]NOTE: Our results are not complete as we ran into issues regarding GPU usage with Google Colab. We aimed to run 30 epochs as opposed to 310 epochs which is what Pham et al (2018) used. However we encountered problems such as running out of GPU hours available, thus we were not able to fully run our training.

6 References

Zoph, Barret and Le, Quoc V. Neural architecture search with reinforcement learning. *arXiv: 1611.01578*. In ICLR, 2017.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In Machine Learning, 1992.

Hieu Pham and Melody Y. Guan and Barret Zoph and Quoc V. Le and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. *arXiv: 1802.03268*, 2018.

Renqian Luo and Fei Tian and Tao Qin and Enhong Chen and Tie-Yan Liu. Neural Architecture Optimization. *arXiv: 1808.07233*, 2019.

Chenxi Liu and Barret Zoph and Maxim Neumann and Jonathon Shlens and Wei Hua and Li-Jia Li and Li Fei-Fei and Alan Yuille and Jonathan Huang and Kevin Murphy. Progressive Neural Architecture Search. *arXiv: 1712.00559*

Hochreiter, Sepp and Schmidhuber, J¨urgen. Long shortterm memory. In Neural Computations, 1997.

Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding LSTM - a tutorial into Long Short-Term Memory Recurrent Neural Networks. *arXiv: 1909.09586*, 2019.

7 Contributions

7.1 Rohan Jain

- Wrote the Abstract, Intro, Method & contributed to section 4.3 in Experiments.
- Made some of the diagrams from Appendix
- Training search space poisoned models w/ Kanav Singla & Benjamin Zhuo
- Edited & reviewed entire paper

7.2 Kanav Singla

- Worked on the code for search space poisoning and running ENAS w/ Benjamin Zhuo
- Abstract w/ Rohan Jain
- Experiment section, Conclusion & Analysis
- Trained various poisoned models w/ Benjamin Zhuo
- Edited & reviewed entire paper

7.3 Benjamin Zhuo

- Worked on code w/ Kanav Singla
- Related Works w/ Rohan Jain
- Macro Search Space
- Search Space Poisoning w/ Kanav Singla
- Edited & reviewed entire paper

8 Source Code

GitHub Repository Link: Here

Appendix

A

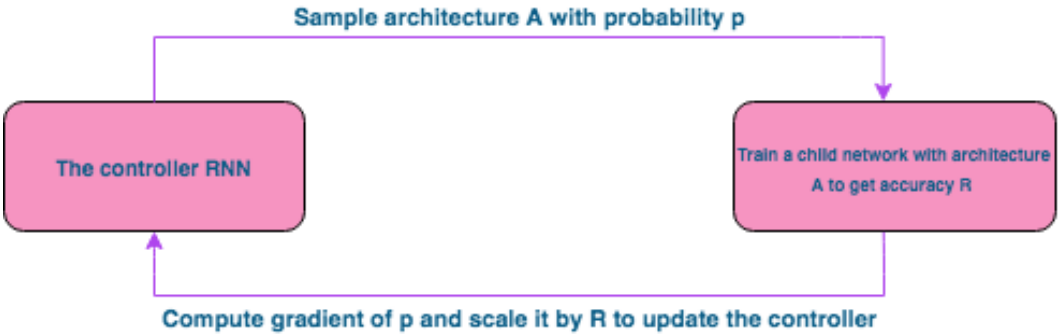

Figure 1: Overview of the NAS controller

B

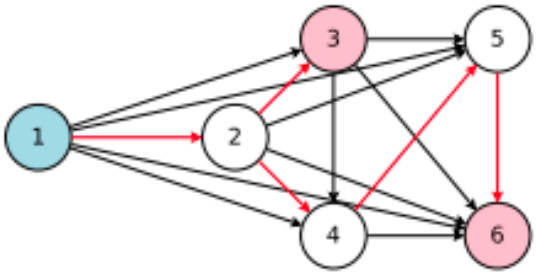

Figure 2: DAG that represents the entire NAS search space.

C

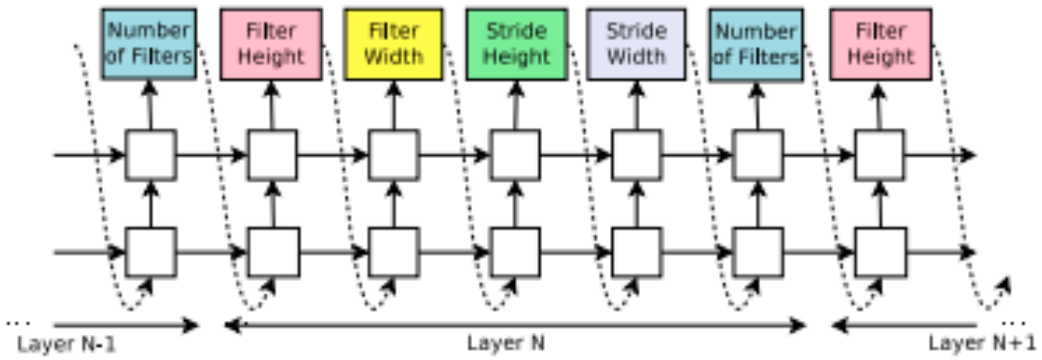

Figure 3: Model description to be predicted by the RNN controller

D

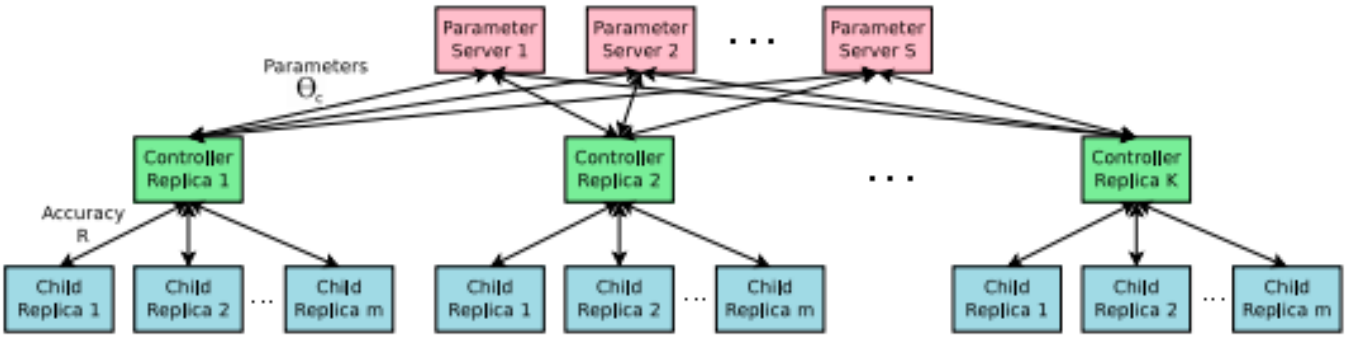

Figure 4: Parallel execution of children training

E

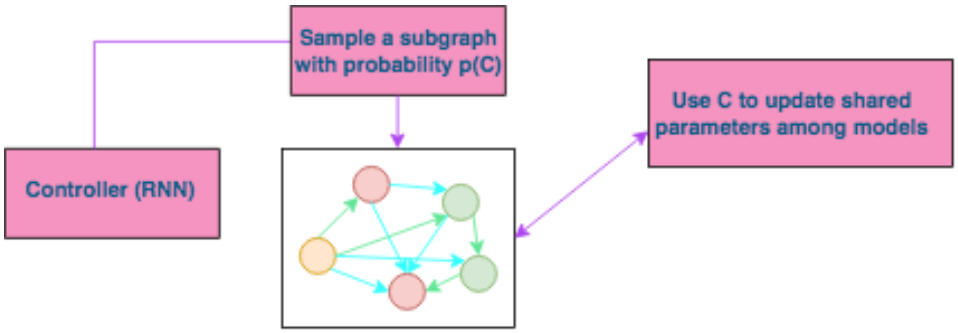

Figure 5: **Phase 1** - Train shared parameters using grad. methods computed on sample architectures.

F

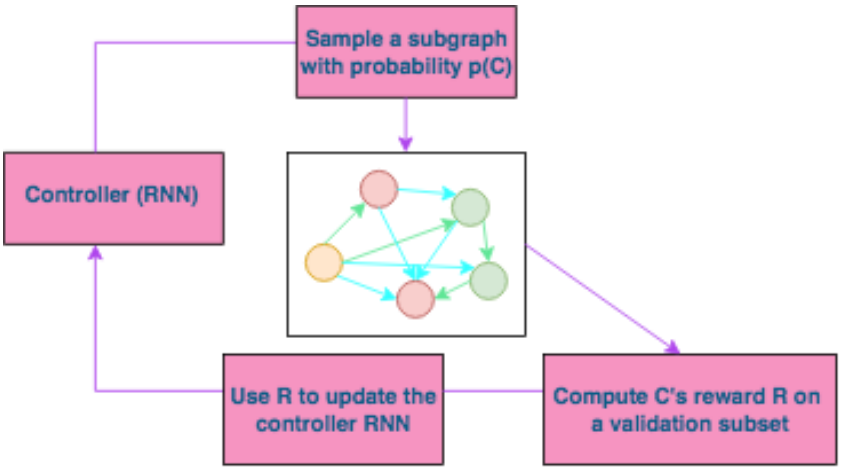

Figure 6: **Phase 2** - Train controller using RL, freeze the shared parameters.

G

[3]
[3 1]
[3 0 0]
[3 0 0 0]
[3 0 0 0 0]
[2 0 1 0 0 0]
[3 0 0 1 0 0 1]
[3 0 1 0 0 0 0 1]
[3 0 1 0 0 1 0 0 1]
[3 0 0 1 0 0 0 0 0 1]
[3 1 0 0 1 0 0 1 0 0 0]
[3 1 0 1 1 1 0 0 1 0 1 0]

Figure 7: Example of sample architecture with 12 layers