**NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING**

**Barret Zoph∗, Quoc V. Le**

**Google Brain**

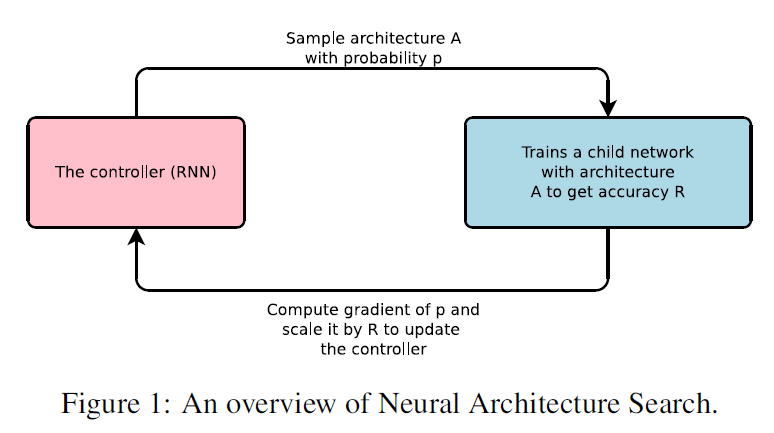
**barretzoph,qvl@google.com**

**Abstract**

In this paper, we use a recurrent network to generate the model descriptions of neural networks and train

this RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set. On the CIFAR-10 dataset, our method, starting from scratch, can design a novel network architecture that rivals the best human-invented architecture in terms of test set accuracy. Our CIFAR-10 model

achieves a test error rate of 3.65, which is 0.09 percent better and 1.05x faster than the previous state-of-the-art model that used a similar architectural scheme.

**What is NAS?**

Neural Architecture Search, a gradient-based method for finding good architectures

(See Figure 1) . Our work is based on the observation that the structure and connectivity of a

neural network can be typically specified by a variable-length string. It is therefore possible to use a recurrent network – the controller – to generate such string. Training the network specified by the string – the “child network” – on the real data will result in an accuracy on a validation set. Using this accuracy as the reward signal, we can compute the policy gradient to update the controller. As a result, in the next iteration, the controller will give higher probabilities to architectures that receive high accuracies. In other words, the controller will learn to improve its search over time.

**Why NAS?**

Hyperparameter optimization is the important method in deep learning world Despite their success, these methods are still limited in that they only search models from a fixed-length space. In other words, it is difficult to ask them to generate a variable-length configuration that specifies the structure and connectivity of a network. In practice, these methods often work better if they are supplied with a good initial model. There are Bayesian optimization methods that allow to search non fixed length architectures but they are less general and less flexible than the method proposed in this paper. Modern neuro-evolution algorithms on the other hand, are much more flexible for composing novel models, yet they are usually less practical at a large scale. Their limitations lie in the fact that they are search-based methods, thus they are slow or require many heuristics to work well.

**What is Controller?**

The controller in Neural Architecture Search is auto-regressive, which means it predicts hyperparameters one a time, conditioned on previous predictions. This idea is borrowed from the decoder in end-to-end sequence to sequence learning Unlike sequence-to-sequence learning, our method optimizes a non-differentiable metric, which is the accuracy of the child network. It is therefore similar to the work on BLEU optimization in Neural Machine Translation Unlike these approaches, our method learns directly from the reward signal without any supervised bootstrapping.

**What is Meta Learning?**

A general framework of using information learned in one task to improve a future task. More closely related is the idea of using a neural network to learn the gradient descent updates for another network and the idea of using reinforcement learning to find update policies for another network.

**How NAS Works?**

A controller to generate architectural hyperparameters of neural networks. To be flexible, the controller is implemented as a recurrent neural network. Let’s suppose we would like to predict feedforward neural networks with only convolutional layers, we can use the controller to generate their hyperparameters as a sequence of tokens.

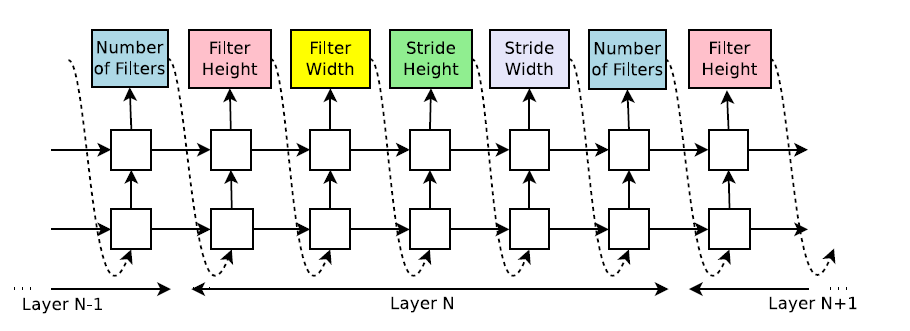


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a soft max classifier and then fed into the next time step as input. Once the controller RNN finishes generating an architecture, a neural network with this architecture is built and trained. The parameters of the controller RNN**, θc (model parameters)**  are then optimized in order to maximize the expected validation accuracy of the proposed architectures.

**Neural Architecture Search Components**

Neural architecture search is a growing area in deep learning research that aims to deliver better performing models and applications. However, it can still be challenging to implement. To further understand how NAS works, let’s dive into its components.

Neural Architecture Search has three main building blocks that can be categorized in terms of search space, search strategy/algorithm, and evaluation strategy ([Elsken et al., 2019](https://arxiv.org/abs/1808.05377)). Each of these components can utilize different methods.

**01 Search space**

Defining the operations used to design DNNs.

**02 Search strategy**

Optimizing metrics according to the approach used to explore the search space is essential for search strategy and performance estimation.

**03 Evaluation strategy**

Evaluating the performance of the DNN, prior to construction and training.

**Model Training with Reinforcement Learning**

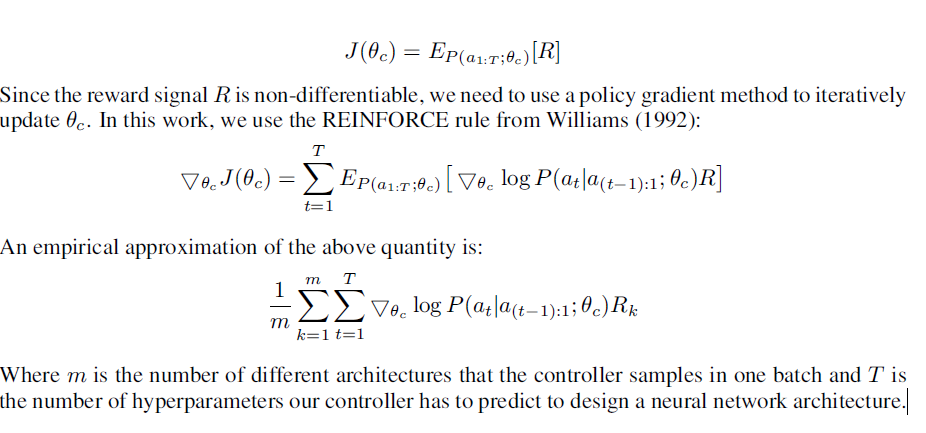
The list of tokens that the controller predicts can be viewed as a list of actions a1:T to design an

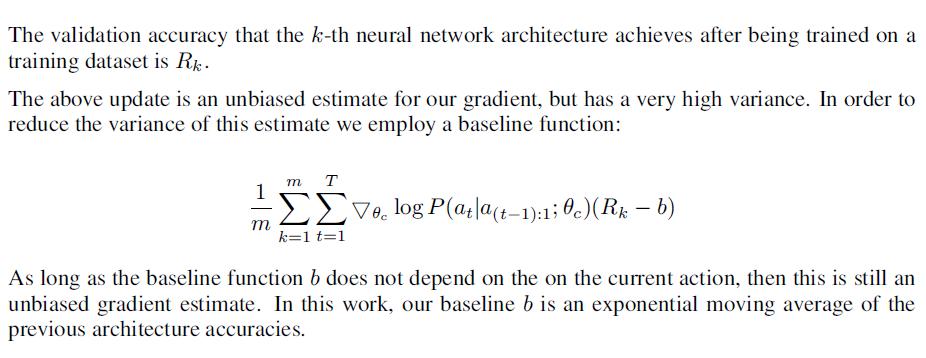
architecture for a child network. At convergence, this child network will achieve an accuracy R on

a held-out dataset. We can use this accuracy R as the reward signal and use reinforcement learning

to train the controller. More concretely, to find the optimal architecture, we ask our controller to

maximize its expected reward.





**Accelerate Training with Parallelism and Asynchronous Updates**

In Neural Architecture Search, each gradient update to the controller parameters θc corresponds to training one child network to convergence. As training a child network can take hours, we use distributed training and asynchronous parameter updates in order to speed up the learning process of the controller. We use a parameter-server scheme where we have a parameter server of S shards, that store the shared parameters for K controller replicas. Each controller replica samples m different child architectures that are trained in parallel. The controller then collects gradients according to the results of that minibatch of m architectures at convergence and sends them to the parameter server in order to update the weights across all controller replicas. In our implementation, convergence of each child network is reached when its training exceeds a certain number of epochs.

