**Neural Architecture Search**



**Apoorva Verma 2K15/IT/016**

**Karan Goyal 2K15/IT/036**

**Mudit Verma 2K15/IT/044**

**Pradyumna Sinha 2K15/IT/053**

**PROBLEM STATEMENT**

**Neural architecture search** (NAS) is a technique that utilizes [machine learning](https://en.wikipedia.org/wiki/Machine_learning) to automate the design of [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network). It involves tasks like model selection, hyperparameter optimization, etc.

Our problem is, given a neural architecture search space F, the input data X, and the cost metric Cost(·), we aim at finding an optimal neural network f ∗ ∈ F with its trained parameter θf\*, which could achieve the lowest cost metric value on the given dataset X .

Mathematically, this definition is equivalent to find f ∗ satisfing:

f∗ = argmin min Cost(f(X;θf)), f∈F θf where θf ∈ Rw(f) denotes the parameter set of network f , w(f) is the number of parameters in f.

**INTRODUCTION**

Google CEO Sundar Pichai wrote that, *“designing neural nets is extremely time intensive, and requires an expertise that limits its use to a smaller community of scientists and engineers. That’s why we’ve created an approach called AutoML, showing that it’s possible for neural nets to design neural nets.”*

What Pichai refers to as using “neural nets to design neural nets” is known as **neural architecture search;** typically, **reinforcement learning or evolutionary algorithms**are used to design the new neural net architectures. This is useful because it allows us to discover architectures far more complicated than what humans may think to try, and these architectures can be optimized for particular goals. Neural architecture search is often very computationally expensive.

To be precise, neural architecture search usually involves learning something like a layer (often called a “cell”) that can be assembled as a stack of repeated cells to create a neural network.

**NAS EXAMPLE**

Neural architecture search is good for finding new architectures! Google’s AmoebaNet was learned via neural architecture search, and (with the inclusion an aggressive learning schedule and changing the image size as training progresses) is now the **cheapest way to train ImageNet on a single machine.**

AmoebaNet was not designed with a reward function that involved the ability to scale, and so it didn’t scale as well as ResNet to multiple machines, but a neural net that scales well could potentially be learned in the future, optimized for different qualities.

**INSPIRATION**

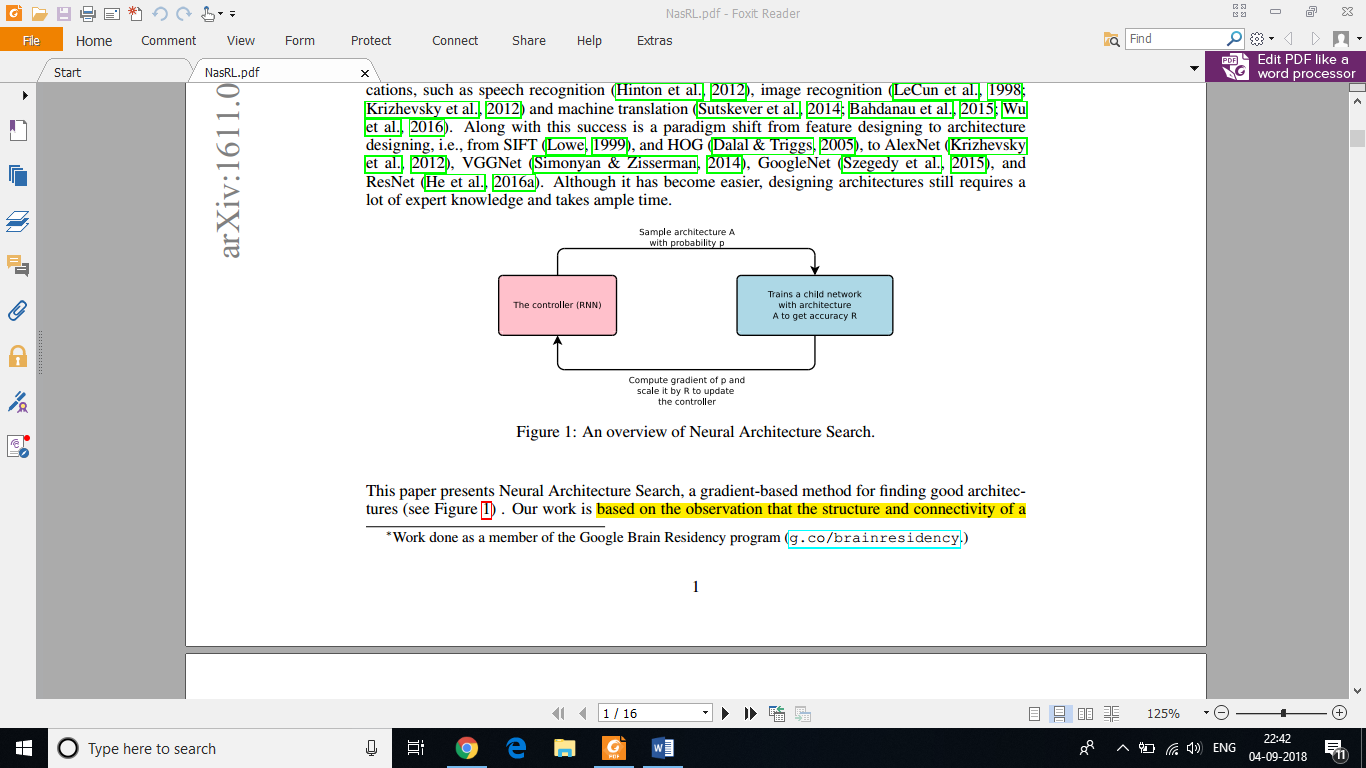
* Designing neural nets is extremely time intensive, and requires an expertise that limits its use to a smaller community of scientists and engineers.
* Allows to discover architectures far more complicated than what humans may think to try that can further be optimized for respective goals.
* Parameter Tuning is not at all a trivial task and may take days to perfect.
* It enables the true meaning of Machine Learning, by making the machine learn the “method” (architecture) of learning itself as well.

**GENRAL APPROACH TO NAS**

**Refer** <https://arxiv.org/pdf/1808.05377.pdf>

**SOLUTION (AutoML)** <https://arxiv.org/abs/1611.01578>

A controller implemented as a Recurrent Neural Network(RNN) will be used to generate architectural hyperparameters of neural networks. Training the network specified by the controller 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.



STEPS INVOLVED

**Step 1 : Generate Model Descriptions**

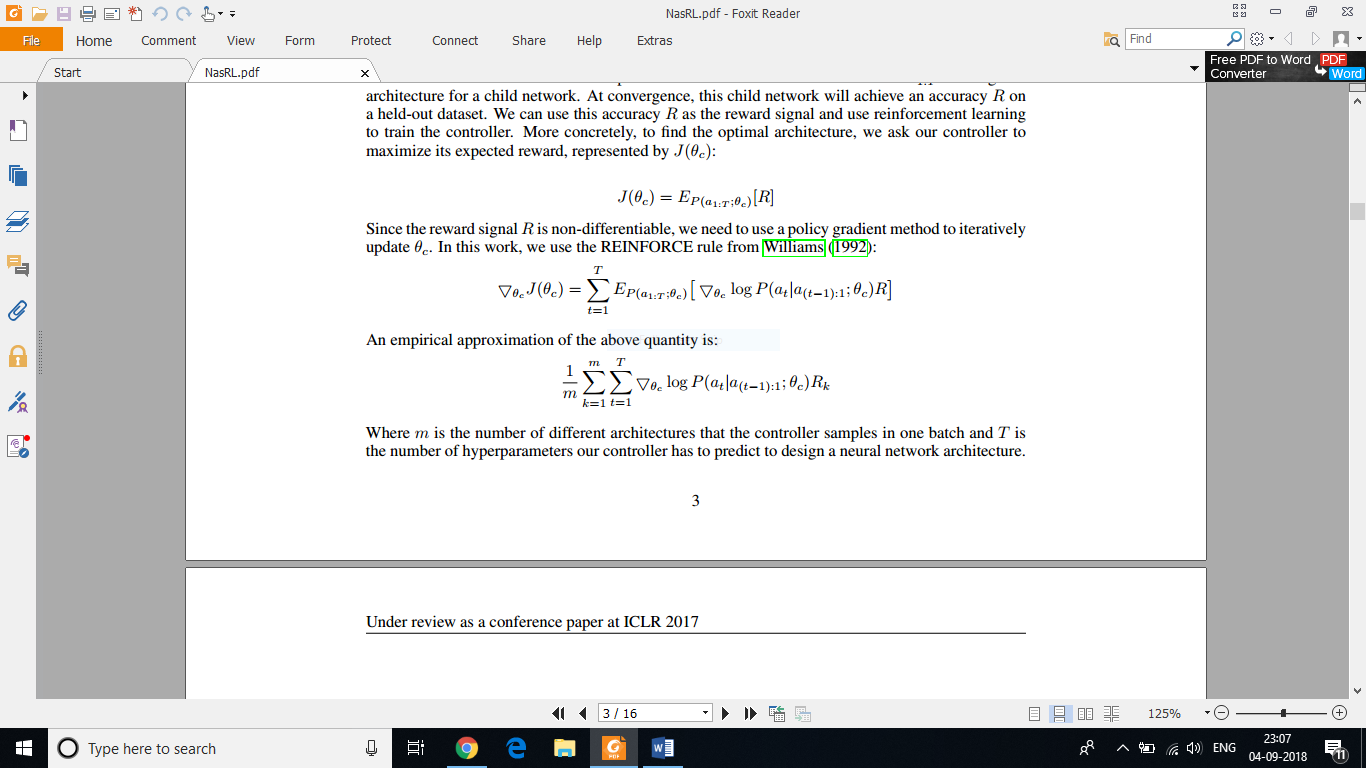
Once the controller RNN finishes generating an architecture, a neural network with this architecture is built and trained. At convergence, the accuracy of the network on a held-out validation set is recorded.  
The parameters of the controller RNN, θc, are then optimized in order to maximize the expected validation accuracy of the proposed architectures.

**Step 2 : Updating parameters & Training with REINFORCE**

At convergence, the proposed 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. Expected reward [J(θc)] can be represented by,

***J*(*θc*) = *EP* (*a*1:*T* ;*θc*)[*R*]**

Since the reward signal R is non-differentiable, we need to use a policy gradient method to iteratively update θc. Using REINFORCE algorithm, we get,



Where m is the number of different architectures that the controller samples in one batch and T is the number of hyperparameters our controller has to predict to design a neural network architecture.

**Step 3 : Using Skip Connections to Optimize**

We use skip connections to widen the search space and make our deep network easy to optimize. At layer *N*, we add an anchor point which has *N -* 1 content-based sigmoids to indicate the previous layers that need to be connected. Each sigmoid is a function of the current hidden state of the controller and the previous hidden states of the previous *N -* 1 anchor points,

**P(Layer j is an input to layer i) = sigmoid(*v*Ttanh(*Wprev ∗ hj* + *Wcurr ∗ hi*))**

**SCOPE OF IMPROVEMENT**

* The proposed method is computationally expensive (800 GPUs)
* Does not implement training cut-off through Learning Curve Prediction, rather uses a naïve thresholding mechanism.
* There is no interpretability of the architectures produced unlike popular networks like ResNets.
* Has many fixed hyper parameters which are not learned like learning rate, choice of optimizer.
* The mechanism is strict upon selecting window sizes from a set.

**Other Notable Solutions:**

**1. Hill Climbing Approach** <https://arxiv.org/abs/1711.04528>

Hill climbing procedure applies network morphisms, followed by short cosine-annealing optimization runs. Surprisingly, the approach yielded competitive results, requiring resources on the same order of magnitude as training a single network. E.g., on CIFAR-10, the method designed and trained a network with an error rate below 6% in 12 hours on a single GPU.

**2. ENAS Based Solution**<https://arxiv.org/abs/1711.04528>

Another solution which uses a controller for neural architecture search. In the so-called Efficient Neural Architecture Search (ENAS), a controller discovers neural network architectures by learning to search for an optimal subgraph within a large graph. The controller is trained with policy gradient to select a subgraph that maximizes the validation set's expected reward. The model corresponding to the subgraph is trained to minimize a canonical cross entropy loss. Multiple child models share parameters, ENAS requires fewer GPU-hours than other approaches and 1000-fold less than "standard" NAS. On CIFAR-10, the ENAS design achieved a test error of 2.89%, comparable to NASNet.On Penn Treebank, the ENAS design reached test perplexity of 55.8.

**3. Differentiable architecture search (DARTS)**

This research was recently released from a team at Carnegie Mellon University and DeepMind. DARTS assumes the space of candidate architectures is continuous, not discrete, and this allows it to use gradient-based approaches, which are vastly more efficient than the inefficient black-box search used by most neural architecture search algorithms.

To learn a network for Cifar-10, **DARTS takes just 4 GPU days**, compared to **1800 GPU days for NASNet** and **3150 GPU days for AmoebaNet** (all learned to the same accuracy). This is a huge gain in efficiency! Although more exploration is needed, this is a promising research direction. Given how Google frequently equates neural architecture search with huge computational expense, efficient ways to do architecture search have most likely been under-explored.

**NASNet**

This work searches for an architectural building block on a small data set (Cifar10) and then builds an architecture for a large data set (ImageNet). This research was **very computationally intensive** with it taking 1800 GPU days (the equivalent of almost 5 years for 1 GPU) to learn the architecture (the team at Google used **500 GPUs for 4 days**!).

**REFERENCES**

* <https://medium.com/syncedreview/sjtu-mit-paper-reinvents-neural-architecture-search-slashes-computational-resource-requirements-f3171f1ea1b4>
* <http://www.arxiv-sanity.com/search?q=neural+architecture+search>
* <http://www.fast.ai/2018/07/16/auto-ml2/>
* <http://www.fast.ai/2018/07/23/auto-ml-3/>
* <https://www.ml4aad.org/automl/literature-on-neural-architecture-search/>
* <https://arxiv.org/abs/1611.01578>
* <https://arxiv.org/pdf/1808.05377.pdf>
* <https://en.wikipedia.org/wiki/Neural_architecture_search>
* <https://arxiv.org/abs/1711.04528>
* <https://arxiv.org/abs/1802.03268>
* <https://arxiv.org/abs/1611.01578>
* <https://www.youtube.com/watch?v=sROrvtXnT7Q>
* <https://www.youtube.com/watch?v=XP3vyVrrt3Q>
* <https://www.youtube.com/watch?v=fbCcJaSQPPA>
* <https://www.youtube.com/watch?v=79tmPL9AL48>