Neural Architecture Search

Ashish Kumar Jayant [*1] **Somyajit Guin** [*1] **Ankith Kumar** [*1]

Abstract

Most of the state of the art deep learning architectures for various datasets are highly complex and it took time and effort of team of experienced machine learning researchers and engineers. Neural Architecture Search is an attempt to solve the problem of coming up with such an complex architecture in an automated fashion. Challenge lies in bounding our search space, improving the efficiency of our search strategy and also evaluation of models coming up in process.

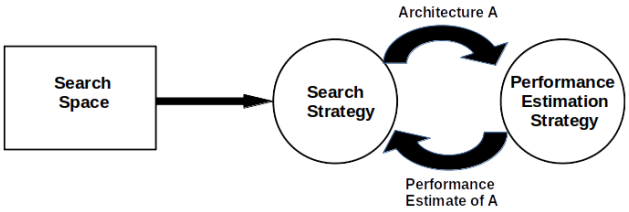

Figure 1. Components of Neural Architecture Search (Zoph & Le, 2016)

1. Problem Description

The problem of Neural Architecture Search is defined in three components (Elsken et al., 2018):

- **Search Space** : The search space consists of all architectures that can be represented in principle. However we use some prior knowledge about elements of architectures suited for a particular task for e.g we will include convolution and maxpool filters for a image classification problem in search space. Incorporating prior knowledge reduces search space but increases human bias.
- **Search Strategy**: The search strategy is about how to explore the search space of architectures in an efficient way as well as we do not miss an optimal architecture. Exploration-exploitation tradeoff needs to balanced here.
- **Performance Estimation Strategy**: When we come up with an architecture, we evaluate its performance but doing that it in conventional fashion may result in computationally expensive NAS model, so we need strategy to do evaluation of our sampled models in efficient way as well.

Let's explore the approaches being taken till now in addressing these problems in literature review.

[*]Equal contribution [1]Department of CSA, IISc Bangalore. Correspondence to: Ashish Kumar Jayant <ashishjayant@iisc.ac.in>.

Machine Learning Spring 2020 Project,CSA, IISc Bangalore

2. Literature Review

2.1. Neural Architecture Search via Reinforcement Learning (Zoph & Le, 2016)

Neural Architecture Search can be formulated as a reinforcement learning problem where search space consists of all architecture that can be represented in principle and action space is same as our search space. Agent's action is to generate an architecture from search space and reward is accuracy on validation set. Let's look at implementation part of it. Neural Network can be represented as a variable length string which makes it ideal for RNN to process. So RNN is chosen as controller and it generates child network. Child network is generated and after training it results in some accuracy on validation set, then this accuracy is used as reward signal to compute policy gradient to update the controller. In next iteration, the controller gives higher probability to architectures with higher accuracy and search is improved over time.(Figure 2)

2.1.1. Generation

We will consider Image Classification problem (on CIFAR dataset) here as an example. RNN controller samples hyperparameters of network which can consist of no. of filters, filter width, filter stride etc. While implementing it generation stops after it exceeds threshold no. of layers. This threshold is increased as training progresses. After RNN stops, the child network formed is trained and tested on validation set and parameters of RNN, θ is updated using Policy Gradient method.

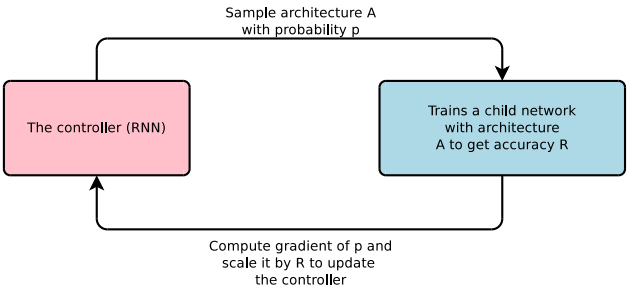

Figure 2. Neural Architecture Search via Reinforcement Learning (Zoph & Le, 2016)

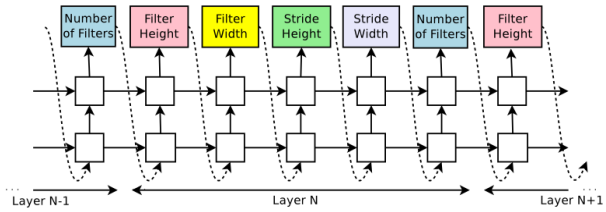

Figure 3. RNN controller sampling elements of CNN layer by layer(Zoph & Le, 2016)

2.1.2. Training

We are using exact notation as used in paper for uniformity. Controller predicts token at every time step and this can viewed as sequence of actions $a_{1:T}$ agent (in this case our RNN controller) takes. At convergence every child network will have accuracy R which will work as agent's reward signal. Objective is to vary θ in such a way that expected reward is maximised. Expected reward is represented by $J(\theta)$,

$$J(\theta) = E_{a_{1:T}}[R]$$

Policy gradient method is being used since reward signal is non-differentiable. REINFORCE rule is used to do policy gradient to iteratively update θ -

$$\bigtriangledown_\theta J(\theta) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta)}\big[\bigtriangledown_\theta \log P(a_t|a_{(t-1):1};\theta)R\big]$$

Empirical estimate of above quantity taking problem of high variance into consideration is :

$$\bigtriangledown_\theta J(\theta) = \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \bigtriangledown_\theta \log P(a_t|a_{(t-1):1};\theta)(R_k - b)$$

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. The validation accuracy that the k-th neural network architecture achieves after being trained on a training dataset is R_k, b is exponential moving average of previous architecture accuracies. In order to speed up the learning process of a controller, distrubuted training and asynchronous parameter update is used.

Pros : Image classification model achieves a 3.65 test set error, while being 1.05x faster than the current best model.Note that model is fast not the whole NAS process. On language modeling with Penn Treebank, it achieved a test set perplexity of 62.4 on the Penn Treebank dataset, which is 3.6 perplexity better than the previous state-of-the-art.

Cons : On CIFAR, it required 800 GPUs to achieve good results in distributed training. On Penn TreeBank language modelling they had to use 450 GPUs and took 3-4 days to train. It also makes this approach difficult to implement.

2.2. Efficient Neural Architecture Search via Parameter Sharing (Pham et al., 2018)

The main idea of this approach is that NAS actually iterates over subgraphs of some large computational graph, hence we can represent search space as one directed acyclic graph. The local computation at each node have their own parameters which are used only when it part of sampled subgraph by controller. This design allows parameters to be shared among all child models and it results in reduced computational expense by 1000x because problem of conventional NAS (Zoph & Le, 2016) was that training of each child model to convergence, only to measure its accuracy and then throw away all trained weights while this approach forces all child model to share weights.

2.2.1. Generation and Training

We will define search space as well for e.g in case of Image Classification on CiFAR we have 6 operations - convolutions with filter sizes 3 and 5,depthwise-separable convolutions with filter sizes 3 and 5,max pooling and average pooling of kernel size 3. Total no of networks possible for L layers = $6^L * 2^{L(L-1)/2}$, for L=12 it gives $1.2 * 10^{29}$ networks. So as to reduce this instead of designing whole CNN we design smaller modules and combine them (Zoph et al., 2017). We define computational graph of B nodes to represent what happens locally in a cell. We use LSTM controller here which makes 2 decisions 1) Two previous nodes to be used as input 2) two operations to apply. After operations are done their results are added. This results in reduced search space and better performance on validation set. There are two sets of learnable parameters here,shared parameters of child models ω and parameters of LSTM controller θ. First θ is fixed ω is trained using Monte Carlo estimate, then ω is fixed and θ is optimized using REINFORCE rule to maximise the expected reward.

Pros : Single NVIDIA GTX 1080i took less than 16 hrs to come up with an optimal model for Penn TreeBank dataset.

References

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey, 2018.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing, 2018.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning, 2016.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition, 2017.