Neural Architect: A Multi-objective Neural Architecture Search with Performance Prediction

Yanqi Zhou
Baidu Silicon Valley AI Lab
Sunnyvale, California
zhouyanqi@baidu.com

Gregory Diamos
Baidu Silicon Valley AI Lab
Sunnyvale, California
gregdiamos@baidu.com

ABSTRACT

Despite of prior advances in automatic Neural Architecture Search (NAS), automatic NAS still largely relies on vast computational resources (e.g. hundreds of GPUs, thousands hours of training time). A Reinforcement Learning (RL) based NAS takes prohibitively long time as a RL agent requires measuring the validation error for each generated target architecture and the algorithm has little intrinsic parallelism. Moreover, prior works target only accuracy but lack consideration for computational resources or efficiency. For instance, increasing the model size can improve model accuracy, but the network might be suboptimal under resource constraints (e.g. total number of parameters). We propose Neural Architect, a resource-aware multi-objective reinforcement learning based NAS with network embedding and performance prediction. Instead of searching a target network from scratch, we use network embedding to encode an existing network to a trainable embedding vector. Based on the embedding, a controller neural network generates next "moves" that transforms the target network. We introduce multi-objective reward function that takes network accuracy, computational resource, and training time into consideration. The reward is predicted by multiple performance simulation networks that are pre-trained or trained jointly with the controller network. With network embedding and performance prediction, Neural Architect can find a good network architecture fast and efficiently. With multi-objective optimization, we can find efficient network architecture with resource constraints.

1 INTRODUCTION

Many techniques for automatic neural architecture search have been proposed [1, 3–6] and yield promising results of designing competitive models compared to human designed models. However, many results are based on vast computational resources such as hundreds of GPUs and thousands of GPU hours, which makes NAS research less realistic for individual researchers or university students. For example, to expedite the process, Google [5] uses distributed and asynchronous training with 800 GPUs.

Moreover, existing techniques do not take consideration of resource constraints (e.g. total number of parameters, memory requirement) for different hardware platforms or neural architecture efficiency (e.g. accuracy per million parameters). As a result, those techniques are likely to find network architectures that are gigantic in size but less efficient. For example, several existing NAS works achieve similar accuracy as DenseNet [2], but the total number of parameters for these networks are significantly higher than DenseNet.

In this work, we propose Neural Architect, as shown in Figure 1, a resource-aware multi-objective reinforcement learning based NAS with network embedding and performance prediction. The goal is to find resource-efficient neural architectures with a multi-objective reinforcement learning algorithm. The main contributions include:

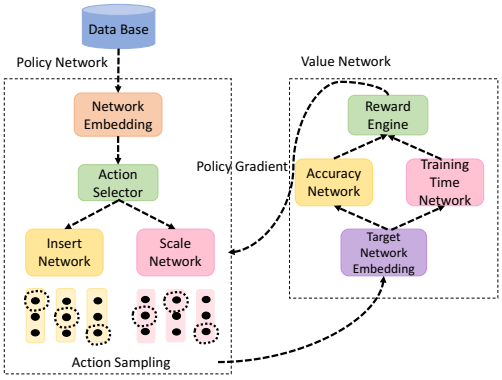

Figure 1: High-level Ideas of Neural Architect. A policy network generates next "moves" based on network embedding. A performance prediction network predicts target network accuracy and training time without actually running the target network till convergence. Policy gradient is applied to train the controller network.

(1) A reinforcement learning based controller neural network with trainable network embedding that takes a configuration of an existing network and makes adaptations to get better neural networks.
(2) A multi-objective reward function that takes into consideration of network accuracy, total number of parameters, and training time. This enables users to customize the reward function according to different needs (e.g. model accuracy, hardware resources and training time).
(3) A performance prediction model that predicts network accuracy and training time without running the target model till convergence. It can be a regression based model or a trainable neural network.

2 METHODOLOGY

Figure 1 shows high-level ideas of Neural Architect. Neural Architect consists of two networks: a policy network and a performance simulation network. The policy network takes in network embedding of current network and generates network transformation actions such as "insert" (insert a layer) or "scale" (scale a layer). The performance simulation network takes in network embedding of

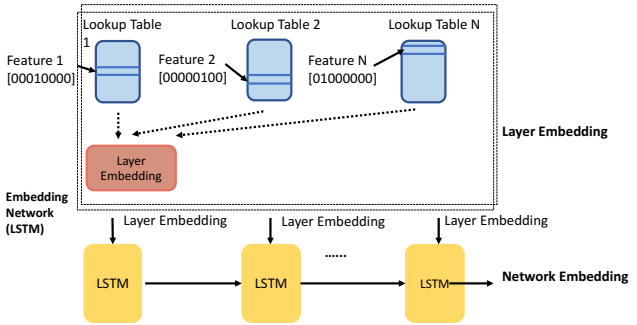

Figure 2: Network Embedding: an LSTM based network that transforms an existing neural network configuration into a trainable representation.

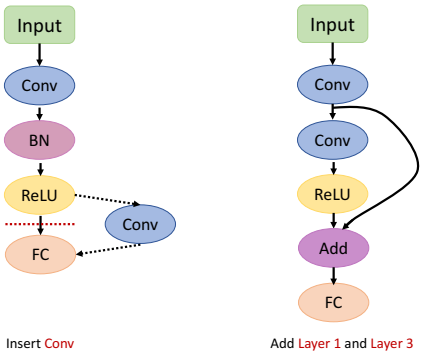

Figure 3: Two representative "Insert" operations: "Insert Conv" and "Insert Add".

the generated target network and data distributions to approximate the reward by predicting both network accuracy and training time. Both the accuracy network and training time network are trainable neural networks that are pre-trained or trained jointly with the policy network. The final reward engine sets weights to network accuracy, model size, and training time respectively according to user specification. The configurable reward engine enables finding neural architectures with various resource constraints such as memory size and GPU time.

2.1 Network Embedding

An critical design feature of Neural Architect is its ability to adapt an existing neural architecture rather than building from scratch. To enable network adaptation, we design a LSTM based embedding network to transform an existing neural architecture configuration into a trainable representation. Figure 2 shows the embedding network, where a layer embedding network takes a layer description and maps layer features into multiple lookup tables. Lookup tables transform the discrete feature space into trainable feature vectors. An LSTM network takes layer feature vectors and generates a layer embedding. After multiple layer embedding have been produced, a network embedding LSTM network processes the sequential information in these layer embeddings and generates a network embedding. This network embedding will be used as an input to the policy network and value network.

2.2 Policy Network

The policy network consists of several operation networks that transforms the existing network using network embedding. Operations including "Insert" and "Scale" can be performed by the policy network, but the policy network can be extended to support other operations as necessary. A trainable action selector chooses the operation to execute the next training iteration. An insert network generates layer type, layer size, and related hyperparameters for that layer. A scale network changes the size and hyperparameters of an existing layer, including the channel size, filter size, and dropout rate.

Figure 3 shows two of the representative "insert" operations generated by the policy network. "Insert Conv" inserts a convolution layer into an existing network. "Insert Add" operation concatenates intermediate results from two previous layers, which is similar to residual layer.

2.3 Performance Simulation Network and Multi-objective Reward

Instead of running the target network till convergence, we use a regression model or a neural network based performance prediction to reduce the training time of the policy network. The performance simulation network takes a target network embedding and an training dataset in terms of size, distribution, and regularity to generate approximated accuracy and training time. Leveraging the embedding network, we can unify layer representation and integrate the information from individual layers. Given a set of sample networks, we can obtain performance curves for each network. For each network x_i, we can obtain a validation accuracy a_i and training timet_i. The objective is to reduce the L1 loss of the predicted accuracy and target evaluated accuracy, and the L1 loss of the predicted training time and target training time. Once the performance prediction network is trained properly, it can be fixed and reused for neural architecture search under various resource constraints. The training time network could be used to model a real system (e.g. Tensorflow running on a V100), or it could use a more idealized hardware model (e.g. a roofline model). For the latter case, the trained policy network can be used to guide future hardware and software optimizations.

If trained jointly, the performance simulation network becomes a value network V. The parameters θ of the policy network are optimized via gradient descent following:

$$\nabla_\theta log\pi(a_t|s_t;\theta)A(s_t,a_t;\theta_v) \quad (1)$$

$$A(s_t,a_t) = r_t + \gamma V(s_{t+1};\theta_v) - V(s_t;\theta_v) \quad (2)$$

The parameters θ_v if the value network is updated via gradient descent using: $\nabla_{\theta_v}[(r_t + \gamma V(s_{t+1};\theta_v) - V(s_t;\theta_v))^2]$

In the multi-objective reward function, we penalize large models by applying a piece-wise linear negative reward function over model size and training time. For instance, we can start applying negative rewards once the model size exceeds 16 MB.

REFERENCES

[1] Chrisantha Fernando, Dylan Banarse, Malcolm Reynolds, Frederic Besse, Max Jaderberg David Pfau, Marc Lanctot, and Daan Wierstra. 2016. Convolution by Evolution: Differentiable Pattern Producing Networks. https://arxiv.org/abs/1606.02580

[2] Gao Huang, Zhuang Liu, Kilian Q. Weinberger, and Laurens van der Maaten. 2017. Densely Connected Convolutional Networks. https://arxiv.org/abs/1611.01578
[3] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. 2017. Evolving Deep Neural Networks. https://arxiv.org/abs/1703.00548
[4] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. 2017. Large-Scale Evolution of Image Classifiers. https://arxiv.org/abs/1703.01041
[5] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. https://arxiv.org/abs/1611.01578
[6] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2017. Learning Transferable Architectures for Scalable Image Recognition. https://arxiv.org/abs/1707.07012