Reinforcement Learning for Neural Network Architecture

Lei Lei and Ruoxuan Xiong

Institute for Computational and Mathematical Engineering

Department of Management Science & Engineering

Contact Information:
Lei Lei
Stanford University
475 Via Ortega, Stanford, CA 94305

Phone: +1 (919) 597 9041
Email: leilei@stanford.edu

Stanford University

Motivation

Deep Neural Network has been successfully in many fields. However, each domain specific tasks requires a different network architecture with human crafted hyperparamers. In this project, we explored ways other than random or lattice search to automatically tune these hyperparameters. Specifically, we used recurrent networks (see Zoph and Le [2016] Zoph et al. [2017]) to generate hyperparameters for convolutional networks with skip connections and experimented with MNIST and cifar10 datasets.

Generating Convolutional Architecture with RNN

Consider a feedforward network with only convolutional (conv) layers. For each layer indexed by i, we have three hyperparameters, filter height h_i, filter width w_i and number of filters n_i. The choice of each parameters are largely decoupled and we can model the decision process as a discrete time, finite horizon Markov Decision Process. Assume at step T, the maximum number of layers is T and

1. **State:** $x_t := \{(h_i, w_i, n_i) \text{ for } 1 \le i \le t-1\} \in \mathbb{N}^{3\times(t-1)}$.
2. **Action:** $a_t = (h_t, w_t, n_t) \in \mathbb{N}^3$.
3. **Reward:** Only a single terminal reward R is considered, which is the accuracy of the network with conv layers, specified by x_T, on a test dataset with the appropriate softmax layers.

We maintained stochastic policy $\pi(a_t|x_t;\theta)$, parametrized by $\theta \in \mathbb{R}^n$, which is constantly updated to approximate the optimal policy π^*.

Note that the state dimension increases linearly with number of actions taken so far; to combat this, we assume there exists an efficient embedded of x_t in $\mathbb{R}^m$ with $h(x_t) \in \mathbb{R}^m$ for some fixed $m \in \mathbb{N}$. Thus the policy $\pi(a_t|x_t;\theta)$ is approximated with a recurrent network $f(a_t, s_t;\theta) : A \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$, where the hidden state s_t provides a natural encoding for the state x_t.

Long Short-Term Memory (LSTM) Controller

Weights are different for different types of hyperparameters, but are the same to generate same type of hyperparameters in different conv layers

$$
\begin{aligned}
i_{t,\alpha}^{(j)} &= \sigma(W^{(i)j}x_{t,\alpha}^{(j)} + U^{(i)j}h_{t,\alpha}^{(j)}) & f_{t,\alpha}^{(j)} &= \sigma(W^{(f)j}x_{t,\alpha}^{(j)} + U^{(f)j}h_{t,\alpha}^{j})\\
o_{t,\alpha}^{(j)} &= \sigma(W^{(o)j}x_{t,\alpha}^{(j)} + U^{(o)j}h_{t,\alpha}^{(j)}) & \tilde{c}_{t,\alpha}^{(j)} &= tanh(W^{(c)j}x_{t,\alpha}^{(j)} + U^{(c)j}h_{t,\alpha}^{(j)})\\
c_{t,\alpha}^{(j)} &= f_{t,\alpha}^{(j)} \circ \tilde{c}_{t-1,\alpha}^{(j)} + i_{t,\alpha}^{(j)} \circ \tilde{c}_{t,\alpha}^{(j)} & h_{t,\alpha}^{(j)} &= o_{t,\alpha}^{(j)} \circ tanh(c_{t,\alpha}^{(j)})\\
y_{t,\alpha} &= softmax(W_\alpha h_{t,\alpha}^{(1)} + b_\alpha) & \alpha &= f, w, c \quad j = 1, 2
\end{aligned}
$$

We also use LSTM controller to sample skip connection:

$$P(\text{Layer } j \text{ is an input to layer } i) = sigmoid(v^T tanh(W_{prev}h_j + W_{curr}h_i))$$

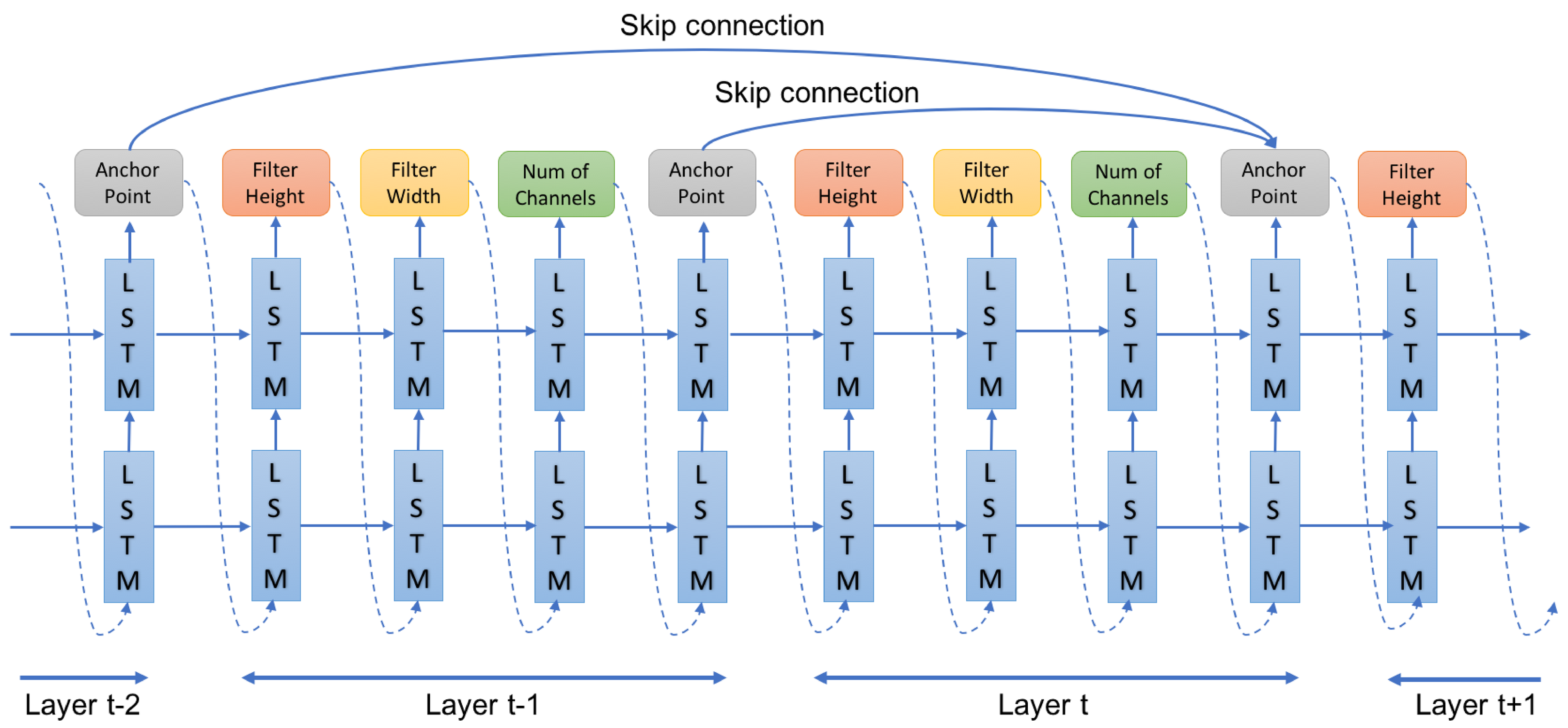


Figure 1: Long Short-Term Memory Controller to generate Convolutional Neural Networks with skip connections. This allows us to search among all possible feedforward network as well as residual network.

Training LSTM controller with REINFORCE

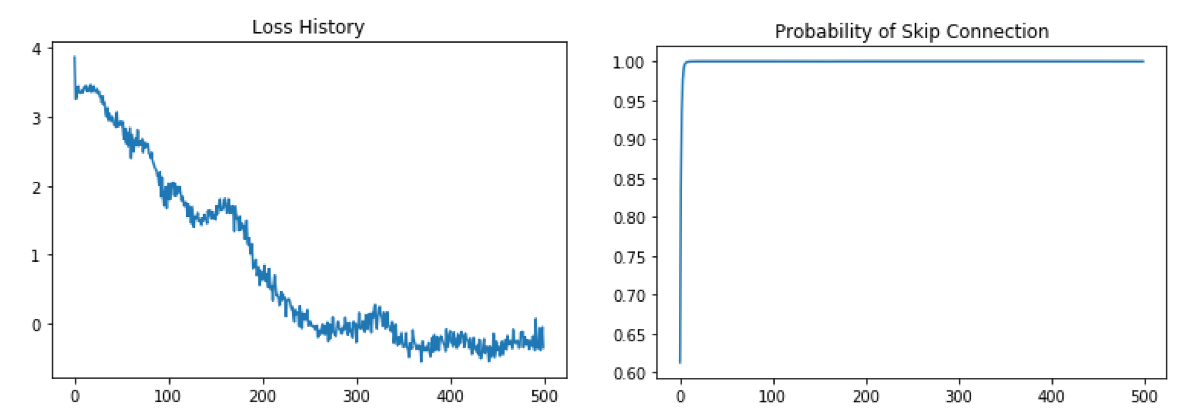
Given the probability distribution of filter height, width, number of channels and skip connection in epoch s is $P_s = (y_{1,h}, y_{1,w}, y_{1,c}, y_{2,h}, y_{2,w}, y_{2,c}, y_{1\to 2}, \cdots)$ from the LSTM controller, we sample K models from P_s. The reward is validation accuracy on a test dataset.

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

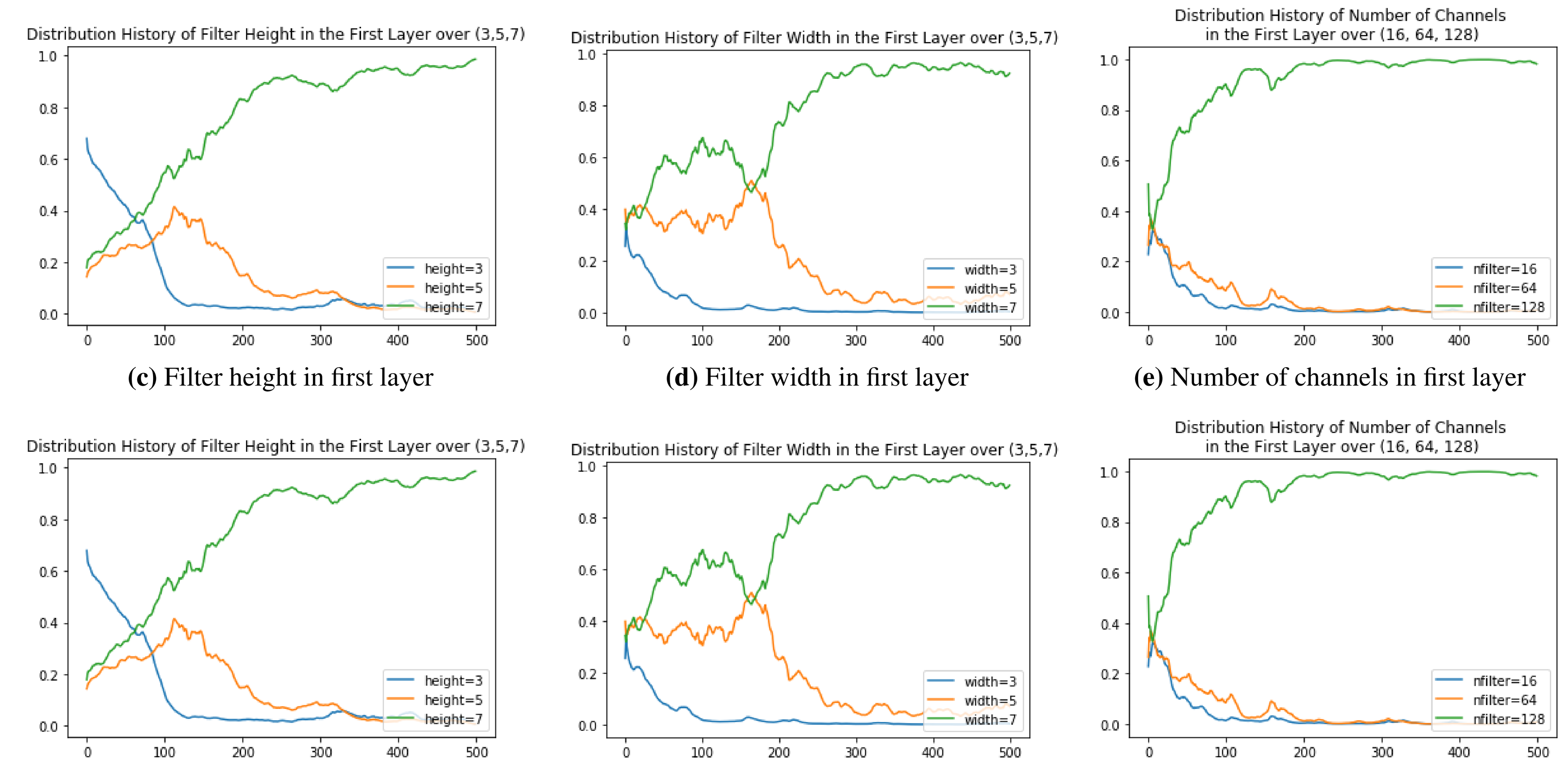
We use REINFORCE algorithm to iteratively update θ_c as to maximize the expected rewards $J(\theta_c)$ under this stochastic policy.

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

Simulation Results with Linear Bandit Problems



(a) Loss History (b) Anchor



(c) Filter height in first layer (d) Filter width in first layer (e) Number of channels in first layer

(f) Filter height in second layer (g) Filter width in second layer (h) Number of channels in second layer

Figure 2: Results from training LSTM controller to generate a two-layer Convolutional Neural Network with rewards being linear bandits. (update rule: Adam, learning rate: 0.01, weight decay: 0.0001, $R_{t,\alpha} = u(0.9\mathbb{1}(a_{t,\alpha}=3) + 0.925\mathbb{1}(a_{t,\alpha}=5) + 0.95\mathbb{1}(a_{t,\alpha}=7))$, $t = 1, 2$, $\alpha = h, w$, $u \sim U(0.9, 1.1)$, and $R_{t,\alpha} = u(0.9\mathbb{1}(a_{t,\alpha}=16) + 0.925\mathbb{1}(a_{t,\alpha}=64) + 0.95\mathbb{1}(a_{t,\alpha}=128))$, $t = 1, 2$, $\alpha = c$, $u \sim U(0.9, 1.1)$)

Simulation Results on MNIST dataset

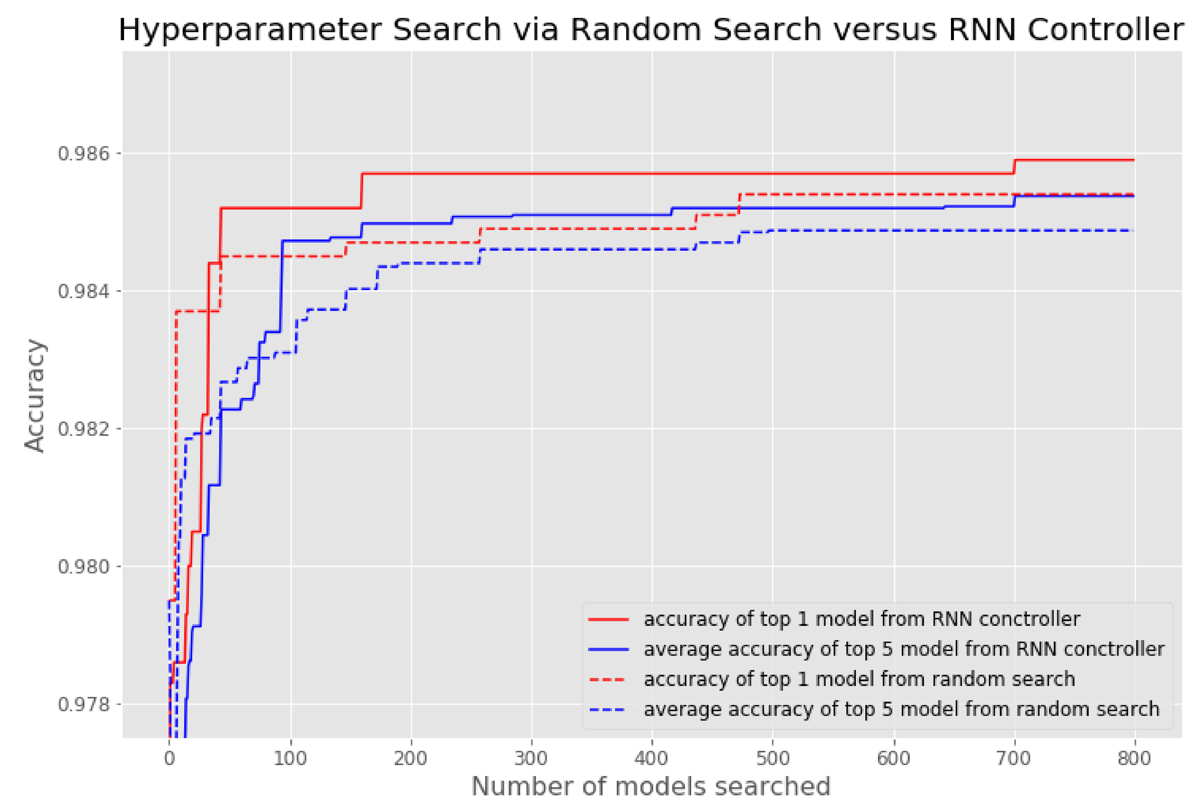


Figure 3: on MNIST dataset, we compare the performance of generated CNN architecture with skip connections via random search with that via LSTM controller.

In the future, we would like to extender the policy network to be able to sample computational graph topology as well.

References

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.