Reinforcement Learning for Neural Network Architecture
Lei Lei and Ruoxuan Xiong
Department of Management Science & Engineering

Motivation
Deep Neural Network has been successfully in many fields. However, each domain specific tasks requires a different network architecture with human crafted hyperparameters. 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: $x_t = \{(h_i, w_i, n_i) \mid 1 \leq i \leq t - 1\} \in \mathbb{N}^{(t-1)}$.
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 constant updated to approximate the optimal policy $\pi^*$. Note that the state dimension increases linearly with number of actions taken so far, to combat this, we assume there exists an efficient embedding of $x_t \in \mathbb{R}^m$ with $h(x_t) \in \mathbb{R}$ for some fixed $m \in \mathbb{N}$. Thus the policy $\pi(a_t|x_t; \theta)$ is approximated with a recurrent network $f(a_t, s_t; \theta)$. $A \times \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}$, 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

$$
\tilde{c}_t = \sigma(W^{(f)}c_{t-1} + U^{(f)}h_{t-1} + b^{(f)})
$$

$$
\tilde{f}_t = \sigma(W^{(f)}a_{t-1} + U^{(f)}h_{t-1} + b^{(f)})
$$

$$
\tilde{h}_t = \tanh(W^{(f)}c_{t-1} + U^{(f)}h_{t-1})
$$

$$
\tilde{y}_t = \text{softmax}(W_{y_{1:t}}h_{t} + b_{y})
$$

We also use LSTM controller to sample skip connection:

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

Training LSTM controller with REINFORCE
Given the probability distribution of filter height, width, number of channels and skip connection in epoch $s$ is $P_s = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, \cdots)$ from the LSTM controller, we sample $K$ models from $P_s$. The reward is validation accuracy on a test dataset.

$$J(\theta_t) = E_{P_s,R}[R] = E_{P_{s(a_t之外)}[R]}$$

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

$$\nabla_{\theta} J(\theta_t) = \sum_{i=1}^{K} \sum_{a_t \in \mathbb{N}^3} \nabla_{\theta} \log P(a_t|a_{t-1}; \theta_t) R_i$$

Simulation Results with Linear Bandit Problems

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

References