CS 229 Project Final Report: Reinforcement Learning for Neural Network Architecture
Category : Theory & Reinforcement Learning

Lei Lei
Ruoxuan Xiong *

December 16, 2017

1 Introduction

Deep Neural Network has been successfully applied to computer vision problems with convolutional neural network (CNN), speech recognition problems with Long Short Term Memory (LSTM) networks. The choices of hyperparameters for neural network architecture still requires expert opinions; in this project, we would like to explore ways to alleviate the pain of hyperparameters search by hand.

Recently, there have been effort to use machine learning techniques to perform hyperparameter search. In Zoph and Le (2016), the authors use a recurrent neural network to output parameters for convolutions filter across consecutive layers in the CNN. In Li and Malik (2016), the authors uses reinforcement learning to choose gradient search algorithms across different epoches during training. In this project, we compared these approaches with traditional discrete/combinatorial optimization techniques.

1.1 Related Work

The first use of recurrent network as a general policy network to sample neural network architecture appeared in Zoph and Le (2016); they authors followed up Zoph, Vasudevan, et al. (2017) to search Inception-like network architectures. In Baker et al. (2016), the authors used a similiar approach while adding model complexity to the objective function as a constraint. In Cortes et al. (2016), the authors proposed a structure learning algorithm with theoretical guarantee on convergence. Since then, there have been active research in this area (Andrychowicz et al. (2016), He et al. (2016))

2 Methods

2.1 Generating Convolutional Architecture with Recurrent Neural Network

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:** $x_t := \{(h_i, w_i, n_i) \text{ for } 1 \leq i \leq t-1\} \in \mathbb{N}^{3\times(t-1)}$.
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.

*Not enrolled

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 $\mathbb{R}^m$ $h(x_t) \in \mathbb{R}^m$ for some fixed $m \in \mathbb{N}$, and the function $f(a_t|x_t;\theta) \approx \pi(a_t|h(x_t);\theta) : \mathbb{R}^m \times A \times \mathbb{R}^n \to \mathbb{R}$ can thus be encoded with a recurrent network. Following Zoph and Le (2016), we discretize the action into four bins and let the output of recurrent network be activated by a softmax layer, thus $f(a_t|h(x_t);\theta)$ gives the probability of choosing one of the four discrete values. This can be understood as sampling the network parameters at different frequency, as opposed to, say uniformly at random by random search.

2.2 Long-Short-Term-Memory Network

2.2.1 Model Description

$f_t(\theta)$ is modeled as a two-layer Long-Short-Term-Memory (LSTM) network. Its output $y_t \in \mathbb{R}^{N_h} \times \mathbb{R}^{N_w} \times \mathbb{R}^{N_c}$ is the predict probability of filter height, h_t, filter width w_t, and number of channels n_t at t-th convolution layer of the feedforward convolutional network. That is, our controller predicts in this order: filter height in 1-st layer, filter width in 1-st layer, number of channels in 1-st layer, filter height in 2-nd layer, filter width in 2-nd layer, number of channels in 2-nd layer and so on, see Fig. 1. Assume there are N_h, N_w, N_c choices for filter height, filter width, and number of channels respectively, then this becomes a classification task. As an example, we can choose filter height or filter width from $\{1, 3, 5, 7\}$, and number of channels from $\{16, 32, 64, 128\}$ according to the corresponding predicted probability y_t of the LSTM controller. The model is

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

$$s.t. \quad h_{t,\alpha}^{(0)} = x_{t,\alpha}^{(1)}, y_{t,h} = x_{t,w}, y_{t,w} = x_{t,c}, y_{t,c} = x_{t+1,h}$$

where $j = 1, 2$ is either first or second layer LSTM, t is the t-th convolution layer, parameter $\alpha = h, w, c$ is either filter height, filter width or number of channels, $i_{t,\alpha}^{(j)}$, $f_{t,\alpha}^{(j)}$, $o_{t,\alpha}^{(j)}$ are input, forget and output gates, $c_{t,\alpha}^{(j)}$ is the state cell, $h_{t,\alpha}^{(j)}$ is the hidden state, $y_{t,\alpha}$ is the predicted probability distribution of N_α choices of parameter α i the t-th layer.

$h_{t,\alpha}^{(0)} = x_{t,\alpha}^{(1)}$ implies the output of the first LSTM layer is the input of the second LSTM layer. $y_{t,h} = x_{t,w}, y_{t,w} = x_{t,c}, y_{t,c} = x_{t+1,h}$ implies previous step's output is the input of next step's input. We set the initial input of filter height in the 1-st layer $x_{1,h}$ as $(1/N_c, 1/N_c, \cdots, 1/N_c)$.

Deep neural networks are usually hard to train, because of the vanishing gradient problem. ResNet (He et al. (2016)), with a shortcut at each layer, provides an easy way to solve the vanishing gradient problem and won the 1st place on the ILSVRC 2015 classification task. Since shortcuts in network architecture have already achieved successes, we allow skip connection in the LSTM controller and sample if a previous layer j has connection with current layer i.

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

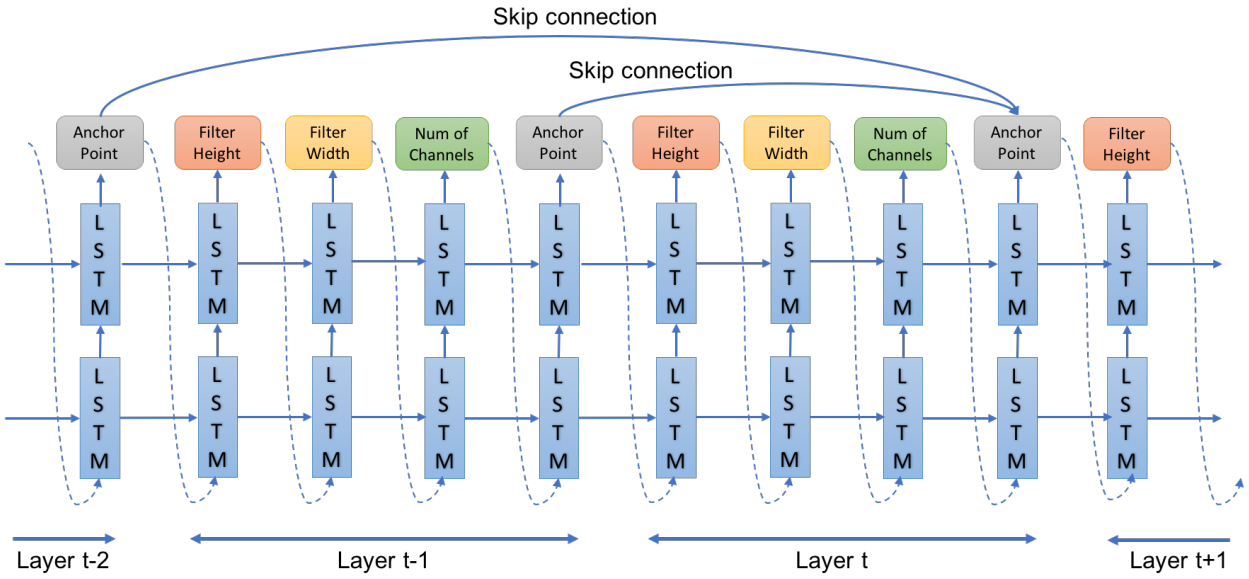


Figure 1: Long Short-Term Memory Controller to generate Convolutional Neural Networks with skip connections. This allows us to search among all possible feedforward network as well as residual network.

2.2.2 Training LSTM controller with REINFORCE

When the maximum number of layers is T, in each epoch s, we have the probability distribution of filter height, width, number of channels and skip connection in epoch s is

$$P_s = (y_{1,h}, y_{1,w}, y_{1,c}, y_{2,h}, y_{2,w}, y_{2,c}, y_{1\to 2}, \cdots, y_{T-1\to T})^T \in [0,1]^{3T+\binom{T}{2}}$$

from the LSTM controller. We sample K models from the the probability distribution P_s, denoted as $(a_{k,1,h}, a_{k,1,w}, a_{k,1,c}, a_{k,2,h}, a_{k,2,w}, a_{k,2,c}, \cdots) \triangleq (\tilde{a}_1, \tilde{a}_2, \tilde{a}_3, \tilde{a}_4, \tilde{a}_5, \tilde{a}_6 \cdots, \tilde{a}_{3T+T(T-1)/2})$ (there are $T(T-1)/2$ skip connection probabilities when max number of layers is T), where k is the k-th model, and $a_{k,t,\alpha}$ is the value of parameter α in t-th layer in k-th CNN model. For each CNN model, the reward R is defined as the accuracy after we train the models for n epochs. The reward for the LSTM controller is

$$J(\theta_c) = E_{P_s,\theta_c}[R] = E_{P(\tilde{a}_{1:3T+T(T-1)/2};\theta_c)}[R]$$

where θ_c represents all parameters in the LSTM controller. We can replace the population reward by the empirical rewards for the LSTM controller,

$$\hat{J}(\theta) = \sum_{k=1}^{K} R_k$$

We use REINFORCE rule in policy gradient method from Williams (1992) to approximate the gradient of objective function,

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{3T+\binom{T}{2}} E_{P(\tilde{a}_{1:3T+\binom{T}{2}};\theta_c)} \left[\nabla_{\theta_c} log P(\tilde{a}_t | \tilde{a}_{t-1:1}; \theta_c) R\right]$$

The empirical approximation via Monte Carlo methods is

$$\nabla_{\theta_c} J(\theta_c) = \frac{1}{K} \sum_{k=1}^{K} \sum_{t=1}^{3T+\binom{T}{2}} \nabla_{\theta_c} log P(\tilde{a}_t | \tilde{a}_{t-1:1}; \theta_c) R_k \tag{1}$$

Finally, we use vanilla gradient descent to update θ_c given $\nabla_{\theta_c} J(\theta_c)$:

$$\theta_c := \theta_c + \gamma \nabla_{\theta_c} J(\theta_c)$$

3 Experimental Results

The gradient descent algorithm for our LSTM controller is typically applied when ther are many observations of state action transition pair $(s_t, a_t, s_{t+1}, r_{t+1})$, often in the magnitude of millions. Due to constraints of training on a single GPU, we could only sample convolutional neural network in the low thousands.We device the following scheme in our experiments:

1. We implemented REINFORCE algorithm for LSTM controller and tested it with linear bandit problem first, similiar in Wang et al. 2016.

2. We coupled the LSTM controller with convolutional network on MNIST and CIFAR-10 datasets. We train the controller for 20 epochs; in every epoch, 40 different neural network architectures are sampled. In total, 800 different network architectures are sampled and we compared its performance with random search strategy in the same parameter space.

3. On MNIST, we used regularized loss with weight decay 0.001 with Adam optimizer with learning rate 0.01 and trained for two epochs. On CIFAR-10, we used regularized loss with weight decay 0.001 with Adam optimizer with learning rate 0.01 and trained for five epochs. We don not recommend our setting for CIFAR-10 as typical training epoch should be between 20 to 50.

4. At last, we used response surface methodology (see Section 6.2), to model the validation accuracy as a function of hyperparameter and it can capture the general trend of larger filter size at earlier layers and more skip connections are preferred.

3.1 LSTM Controller on Linear Bandit Problem

We experiment with hyperparameters of a two-layer CNN. For each convolution layer, filter height, filter width and number of channels can choose from $[1, 3, 7]$, $[1, 3, 7]$ and $[16, 64, 128]$. We study the question:

If there exists an optimal two-layer CNN architecture, can the LSTM controller generate this model with high probability after training the LSTM controller with some epochs?

In the simulation, we generate models from LSTM controller and simulate the reward (model validation accuracy) according to model architectures by

$$R = (1 + u)R_{1,h}R_{1,w}R_{1,c}R_{2,h}R_{2,w}R_{2,c}R_{1\to 2} \tag{2}$$

where $R_{t,\alpha} = r_1\mathbb{1}(a_{t,\alpha} = 3) + r_3\mathbb{1}(a_{t,\alpha} = 5) + r_3\mathbb{1}(a_{t,\alpha} = 7)$, $t = 1, 2$, $\alpha = h, w$ is the reward of filter height or filter width in t-th layer, and $R_{t,\alpha} = r_1\mathbb{1}(a_{t,\alpha} = 16) + r_2\mathbb{1}(a_{t,\alpha} = 64) + r_3\mathbb{1}(a_{t,\alpha} = 128)$, $t = 1, 2$, $\alpha = c$ is the reward of number of channels in t-th layer, and $u = U([-\epsilon, \epsilon])$ is to generate some noise in the reward. If there is a skip connection between layer 1 and 2, the reward is multiplied by a factor $R_{1\to 2}$. The way to generate reward R has a constraint that $(1 + \epsilon)max(r_1, r_2, r_3)^6 max(R_{1\to 2}) \leq 1$.

From our experiments to generate some CNN models and run a few epochs on image classification tasks, we found that in general when filter height, filter width and number of channels in the CNN model are larger, and if there is skip connections, the model accuracy after training 5 epochs is generally higher. In simulation, we set $r_1 < r_2 < r_3$ and $R_{1\to 2} > 1$, and therefore, the larger the filter height, filter width, or number of channels, the higher the reward.

In each epoch to train LSTM controller, we generate $K = 100$ models from $P_s = (y_{1,f}, y_{1,w}, y_{1,c}, y_{2,f}, y_{2,w}, y_{2,c}, y_{1\to 2})$ (defined in section 2.2.2). We then generate rewards of these model from reward function 3.1 with $r_1 = 0.90, r_2 = 0.925, r_3 = 0.95$ instead of using the model accuracy from training the CNN model on image classification tasks(it is too computational expensive). We use the REINFORCE rule 2.2.2 to update parameters in the LSTM controller. After iterating for 500 epochs, the loss history is Fig. 2a. The history of anchor (skip connection) probability sees 2b. The history of $P_s = (y_{1,f}, y_{1,w}, y_{1,c}, y_{2,f}, y_{2,w}, y_{2,c})$ sees Fig. 2c, 2d, 2e, 2f, 2g and 2h respectively. The reward function 3.1 has largest expected value when filter height is 7, filter width is 7, number of channels is 128 in each convolution layer and there is a skip connection from layer 1 to layer 2 in the two-layer CNN model.

The LSTM controller ***can generate this optimal model with probability almost 1*** *after training the LSTM controller for 300 epochs with 100 models in each epoch.*

3.2 Coupling LSTM Controller with CNN

On CIFAR-10 datasets, We experimented with max pooling and different optimizer to see if it accelerates training. It turned out that batch normalization makes a big impact (see Figure 4) and max pooling is necessary for deeper network to reduce the number of parameter in the softmax layer to avoid hardware resource exhaustion (see Figure 5). Thus, all convolutional layer have batch normalization and max pooling by default.

In our experiments, the convolution neural network is constrained to have at most four layers with $6 = \binom{4}{2}$ possible skip connections. To resolve tensor dimension mismatch, we use the following strategies to build the network:

1. If a layer has not incoming inputs, its input is the training image.
2. If a layer has not outgoing outputs, its outputs is the softmax layer.
3. If a layer has multiple inputs, they are first padded to the same height and width, then concatenate into one input.

With validation set accuracy as the metric, we monitor the best models sampled by LSTM controller and compared it with random search strategy. On the MNIST dataset we could observe a small advantage of LSTM controller (see Figure 3a) ; on the CIFAR-10 datasets, the best model from random search strategy could match that from LSTM controller (see Figure 3b). This could be due to insufficient training for each convolutional network we sampled, as typical CIFAR-10 classifier would require 20 to 50 epoches to achieve an accracy over 70%.

4 Conclusion

In this project, we implemented the neural network hyperparameter search algorithm in Zoph and Le 2016 and tested it on MNIST and CIFAR-10 dataset. Due to resource constraints, we could not observe its advantage over random search strategy on the CIFAR-10 datasets, but we did see a small advantage on the simpler classification task, MNIST. In the future, we would like to explore ways to search other types of hyperparameters.

5 Contribution

The code is available from bitbucket repository upon request. Ruoxuan Xiong implemented the LSTM controller while Lei Lei implemented the convolutional network generator.

References

Andrychowicz, M. et al. (2016). "Learning to learn by gradient descent by gradient descent". In: *Advances in Neural Information Processing Systems*, pp. 3981–3989.

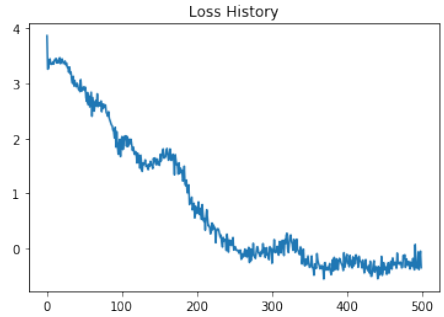
Baker, B. et al. (2016). "Designing neural network architectures using reinforcement learning". In: *arXiv preprint arXiv:1611.02167*.

Cortes, C. et al. (2016). "AdaNet: Adaptive Structural Learning of Artificial Neural Networks". In: *arXiv preprint arXiv:1607.01097*.

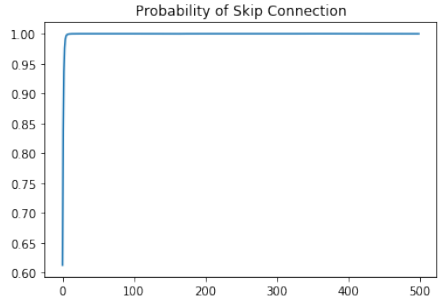
He, K. et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Li, K. and J. Malik (2016). "Learning to optimize". In: *arXiv preprint arXiv:1606.01885*.

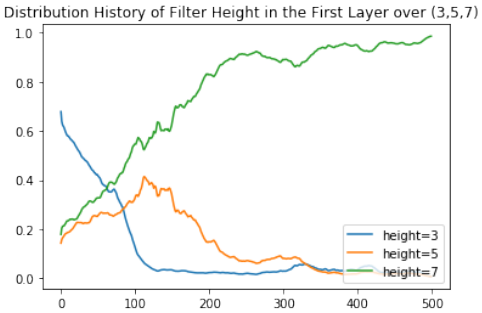
Wang, J. X. et al. (2016). "Learning to reinforcement learn". In: *arXiv preprint arXiv:1611.05763*.



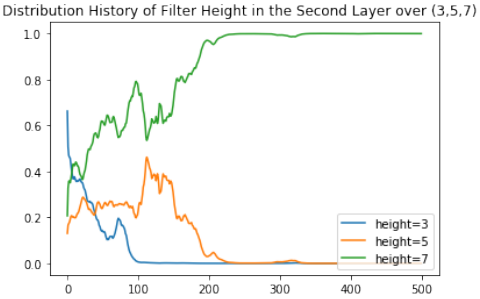
(a) Loss history



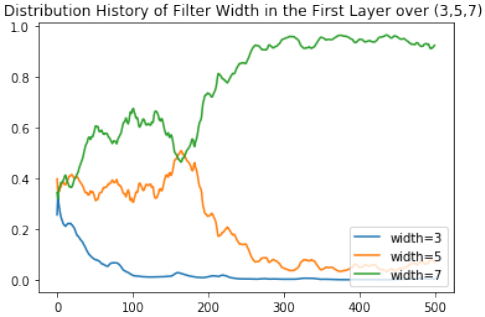
(b) Loss history



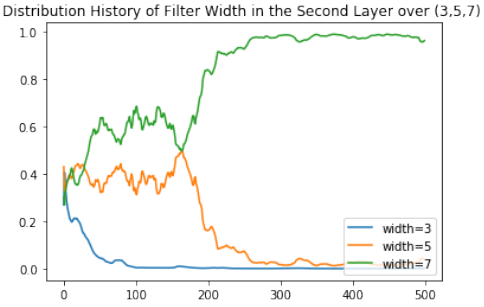
(c) Filter height in first layer $y_{1,h}$



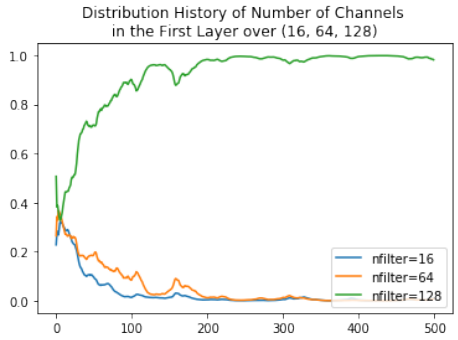
(d) Filter height in second layer $y_{2,h}$



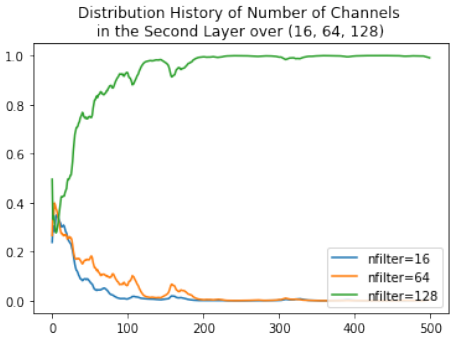
(e) Filter width in first layer $y_{1,w}$



(f) Filter width in second layer $y_{2,w}$

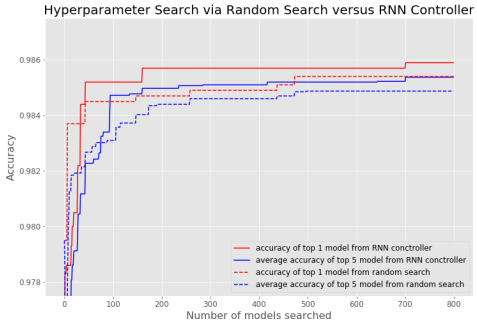


(g) Number of channels in first layer $y_{1,c}$

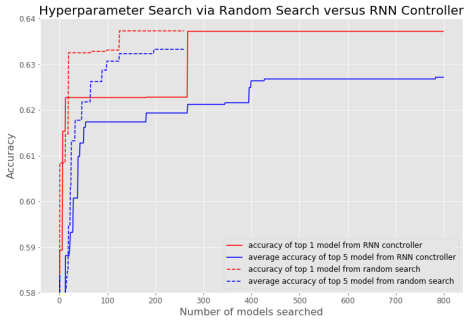


(h) Number of channels in second layer $y_{2,c}$

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, $R_{t,\alpha} = (1+u)(0.90\mathbb{1}(a_{t,\alpha}=3) + 0.925\mathbb{1}(a_{t,\alpha}=5) + 0.95\mathbb{1}(a_{t,\alpha}=7))$, $t=1,2$, $\alpha = h,w$, $u \sim U(0.9,1.1)$, and $R_{t,\alpha} = u(0.90\mathbb{1}(a_{t,\alpha}=16) + 0.925\mathbb{1}(a_{t,\alpha}=64) + 0.95\mathbb{1}(a_{t,\alpha}=128))$, $t=1,2$, $\alpha = c$, $u \sim U(-0.1,0.1)$



(a) Result on MNIST dataset



(b) Result on CIFAR-10 dataset

Figure 3: Random search strategy (dashed line) is used as a baseline model for comparison. This is equivalent to sample all hyperparameters uniformly at random.

Williams, R. J. (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4, pp. 229–256.

Zoph, B. and Q. V. Le (2016). "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578*.

Zoph, B., V. Vasudevan, et al. (2017). "Learning transferable architectures for scalable image recognition". In: *arXiv preprint arXiv:1707.07012*.

6 Appendix

6.1 Effect of Batch Normalization and Max Pooling on Image Classfication Task

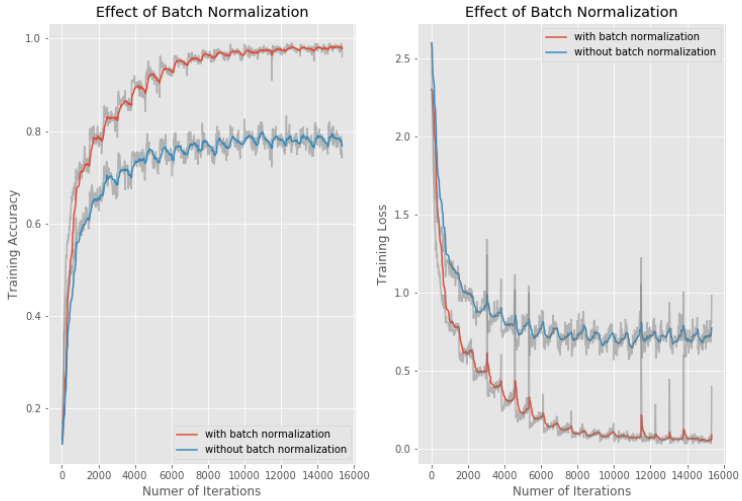


Figure 4: Here we show the inclusion of batch normalization layer after relu activation on a 4 layer CNN with parameters sampled by LSTM controller. The network are run for 20 epochs. The inclusion of batch normalization accelerate the training considerably, to the extent of overfitting, and justifies our choice of training only 10 epochs.

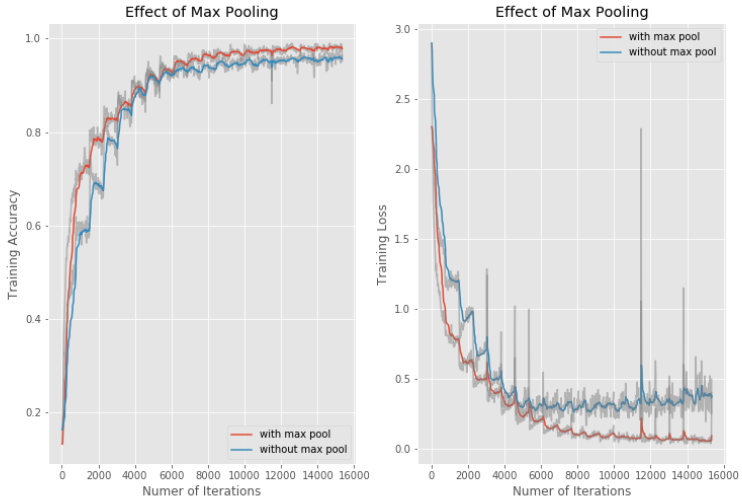
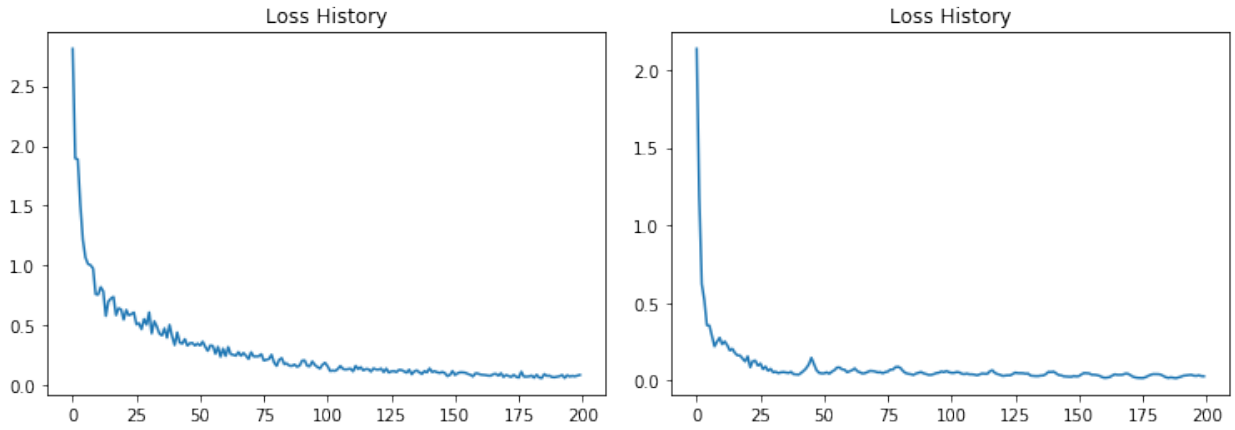


Figure 5: Here we show the inclusion of max pooling layer after every two convolution layers on a 4 layer CNN with parameters sampled by LSTM controller. The network is run for 20 epochs. The inclusion of max pooling payer shortens the training time considerably (from 41m41s to 13m7s) without affecting training loss or accuracy too much. For CNN with more than 4 layers, max pooling is necessary to avoid hitting hardware limit.

6.2 Responce Surface Methodology to predict reward as a function of hyperparametric with linear bandit problem

Instead of using LSTM controller, we would like to fit a simple response surface model that predict rewards from all hyperparameters. We use two models, a quadratic model and a two-layer neural network, to estimate the reward as a function of all hyperparameters. We use the same setting as the simulation for LSTM controller 3.1. Both models are trained with 200 epochs and each epoch has 100 samples. The loss histories for the quadratic model and the two-layer neural network are in Fig. 6a and Fig. 6b. Both models can quickly learn the reward function.

Since filter height, width and number of channels are positively correlated with the expected rewards, we use the quadratic model and the two-layer neural network to predict the rewards for all possible models, and regress with the hyperparameters, see Fig. 8a and 8b. We compare the regression results with the results of regressing the ground truth expected rewards on hyperparameters, see Fig. 7. The regression results using fitted rewards from quadratic model and two-layer neural network are very close to the regression results using ground truth rewards. Therefore, both quadratic model and two-layer neural network fit the reward function very well.



(a) Loss history for quadratic model

(b) Loss history for two-layer neural network

Figure 6: Loss histories of learning the reward function with respect to hyperparameters of the CNN models with the ground truth reward function being linear bandits. (learning rate: 0.01, weight decay: 0.0001, $R_{t,\alpha} = (1+u)(0.90\mathbb{1}(a_{t,\alpha}=3)+0.925\mathbb{1}(a_{t,\alpha}=5)+0.95\mathbb{1}(a_{t,\alpha}=7))$, $t=1,2$, $\alpha=h,w$, $u\sim U(0.9,1.1)$, and $R_{t,\alpha} = u(0.90\mathbb{1}(a_{t,\alpha}=16)+0.925\mathbb{1}(a_{t,\alpha}=64)+0.95\mathbb{1}(a_{t,\alpha}=128))$, $t=1,2$, $\alpha=c$, $u\sim U(-0.1,0.1)$

OLS Regression Results

Dep. Variable:	y	R-squared:	0.999
Model:	OLS	Adj. R-squared:	0.999
Method:	Least Squares	F-statistic:	9.878e+04
Date:	Fri, 15 Dec 2017	Prob (F-statistic):	0.00
Time:	21:21:07	Log-Likelihood:	3879.6
No. Observations:	729	AIC:	-7745.
Df Residuals:	722	BIC:	-7713.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.5248	0.000	3773.433	0.000	0.525	0.525
x1	0.0169	5.39e-05	314.289	0.000	0.017	0.017
x2	0.0169	5.39e-05	314.289	0.000	0.017	0.017
x3	0.0169	5.39e-05	314.289	0.000	0.017	0.017
x4	0.0169	5.39e-05	314.289	0.000	0.017	0.017
x5	0.0169	5.39e-05	314.289	0.000	0.017	0.017
x6	0.0169	5.39e-05	314.289	0.000	0.017	0.017

Omnibus:	326.461	Durbin-Watson:	0.927
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1472.128
Skew:	2.070	Prob(JB):	0.00
Kurtosis:	8.596	Cond. No.	9.28

Figure 7: Regressing ground truth expected rewards on hyperparameters (x1: filter height of 1st CNN layer; x2: filter width of 1st CNN layer; x3: number of channels of 1st CNN layer; x4: filter height of 2nd CNN layer; x5: filter width of 2nd CNN layer; x6: number of channels of 2nd CNN layer; $R_{t,\alpha} = (1+u)(0.90\mathbb{1}(a_{t,\alpha}=3)+0.925\mathbb{1}(a_{t,\alpha}=5)+0.95\mathbb{1}(a_{t,\alpha}=7))$, $t=1,2$, $\alpha=h,w$, $u\sim U(0.9,1.1)$, and $R_{t,\alpha} = u(0.90\mathbb{1}(a_{t,\alpha}=16)+0.925\mathbb{1}(a_{t,\alpha}=64)+0.95\mathbb{1}(a_{t,\alpha}=128))$, $t=1,2$, $\alpha=c$, $u\sim U(-0.1,0.1)$)

OLS Regression Results

Dep. Variable:	y	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	1.323e+16
Date:	Fri, 15 Dec 2017	Prob (F-statistic):	0.00
Time:	21:28:14	Log-Likelihood:	11736.
No. Observations:	729	AIC:	-2.346e+04
Df Residuals:	722	BIC:	-2.343e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.1923	2.9e-09	-6.62e+07	0.000	-0.192	-0.192
x1	0.1036	1.12e-09	9.21e+07	0.000	0.104	0.104
x2	0.1045	1.12e-09	9.3e+07	0.000	0.105	0.105
x3	0.1478	1.12e-09	1.31e+08	0.000	0.148	0.148
x4	0.1799	1.12e-09	1.6e+08	0.000	0.180	0.180
x5	0.0841	1.12e-09	7.48e+07	0.000	0.084	0.084
x6	0.1319	1.12e-09	1.17e+08	0.000	0.132	0.132

Omnibus:	7.921	Durbin-Watson:	1.511
Prob(Omnibus):	0.019	Jarque-Bera (JB):	8.006
Skew:	-0.217	Prob(JB):	0.0183
Kurtosis:	3.275	Cond. No.	9.28

(a) Regressing fitted rewards of quadratic model on hyperparameters

OLS Regression Results

Dep. Variable:	y	R-squared:	0.337
Model:	OLS	Adj. R-squared:	0.332
Method:	Least Squares	F-statistic:	61.26
Date:	Fri, 15 Dec 2017	Prob (F-statistic):	2.32e-61
Time:	21:26:11	Log-Likelihood:	1142.1
No. Observations:	729	AIC:	-2270.
Df Residuals:	722	BIC:	-2238.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.6466	0.006	108.775	0.000	0.635	0.658
x1	0.0175	0.002	7.592	0.000	0.013	0.022
x2	0.0142	0.002	6.152	0.000	0.010	0.019
x3	0.0101	0.002	4.399	0.000	0.006	0.015
x4	0.0106	0.002	4.583	0.000	0.006	0.015
x5	0.0326	0.002	14.153	0.000	0.028	0.037
x6	0.0129	0.002	5.603	0.000	0.008	0.017

Omnibus:	66.745	Durbin-Watson:	1.187
Prob(Omnibus):	0.000	Jarque-Bera (JB):	101.978
Skew:	-0.656	Prob(JB):	7.17e-23
Kurtosis:	4.278	Cond. No.	9.28

(b) Regressing fitted rewards of two-layer neural network on hyperparameters

Figure 8: Regressing fitted rewards of quadratic model and two-layer neural network on hyperparameters (x1: filter height of 1st CNN layer; x2: filter width of 1st CNN layer; x3: number of channels of 1st CNN layer; x4: filter height of 2nd CNN layer; x5: filter width of 2nd CNN layer; x6: number of channels of 2nd CNN layer; $R_{t,\alpha} = (1+u)(0.90\mathbb{1}(a_{t,\alpha}=3) + 0.925\mathbb{1}(a_{t,\alpha}=5) + 0.95\mathbb{1}(a_{t,\alpha}=7))$, $t=1,2$, $\alpha = h, w$, $u \sim U(0.9, 1.1)$, and $R_{t,\alpha} = u(0.90\mathbb{1}(a_{t,\alpha}=16) + 0.925\mathbb{1}(a_{t,\alpha}=64) + 0.95\mathbb{1}(a_{t,\alpha}=128))$, $t=1,2$, $\alpha = c$, $u \sim U(-0.1, 0.1)$)