# Reinforcement learning report: an implementation to ICML 2017 paper Neural Optimizer Search with Reinforcement Learning

**Mo Yang** [1] **Taycir Yahmed** [2]

## Abstract

Optimizer for neural network is hard to design. A paper accepted in ICML 2017, named *Neural Optimizer Search with Reinforcement Learning(1)*, has proposed an approach to automate the process of discovering optimization methods. The author has proposed to train a Recurrent Neural Network controller to generate a string that describe the update rule. The controller is updated by reinforcement learning algorithms, aim to maximize the performance of generated rules on a target network. The purpose of this document is to implement this method and try to reproduce the result the author found.

## 1. Introduction

The choice of optimizer is important to train deep learning neural network. There exist many different optimizers such as SGD, Momentum, ADADelta, RMSProp, ADAM(2)... However, designing am optimizer is very challenging due to non-convex nature of the optimization problem. Designing these optimization algorithms not only demand human prior mathematical knowledge, but also rely on lots of intuitions. So that makes it difficult to get the 'optimum' form of optimizer.

In the paper, the author propose an approach using reinforcement learning to automate the process of designing update rules for optimization methods. The key insight is to use a recurrent neural network controller to generate an update equation for the optimizer. Then the update equation will be used to train a certain target network for fixed number of epochs and get accuracy on a held-out validation set. The controller network use reinforcement learning to update its parameters, with the aim of maximizing the accuracy. The process is shown in figure 1.

[1]ParisSaclay University, France [2]Telecom ParisTech, France.. Correspondence to: Mo Yang <mo.yang@u-psud.fr>, Taycir Yahmed <tyahmed@enst.fr>.

*Proceedings of the $34^{th}$ International Conference on Machine Learning*, Sydney, Australia, PMLR 70, 2017. 

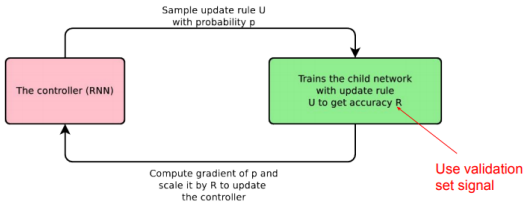


*Figure 1.* An overview of neural optimizer search

## 2. Method

### 2.1. Controller network

In the framework the controller network is a recurrent neural network that generate strings corresponding to update rule, which are applied to target network to estimate update rule's performance. This performance is then used to update the controller so that the controller can generate improved update rules overtime. The key insight is how to describe the update rule? If we draw the computation graph of some common optimization algorithms(see, figure2), we can find that these optimizer can be described as binary expression tree. So the idea is to represent each update rule with a string of five: 1) the first operand to select, 2) the second operand to select, 3) Unary function apply on first operand, 4) Unary function to apply on second operand, 5) the binary function that combine the outputs of unary functions. In such way a string of five describe the update rule.

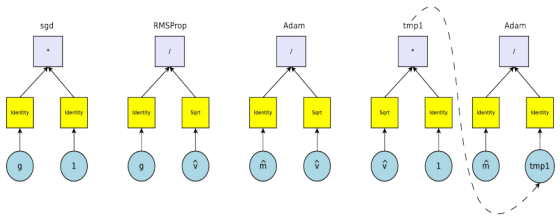


*Figure 2.* Commonly used optimizers

The controller network will sample strings of length 5n (n

is the depth of binary tree, if n is bigger, update rule can be more complex. n is also the number of groups in rnn network). In our experiment for the reason of complexity, we set n equal to 1. The RNN network (see figure 3) will output predictions iteratively. Given the different search space, every prediction is carried out by different softmax.

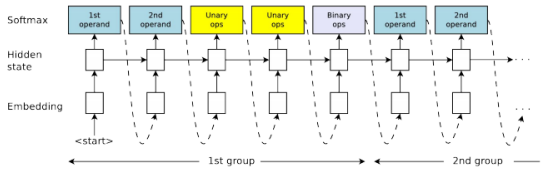


*Figure 3.* Architecture of controller RNN

### 2.2. Policy gradient

The controller network is trained to maximize the performance of sampled update rule on the target network on Cirfar-10.(3) The target network is a simple 2 layers CNN. The training of controller is applied with policy gradients, the objective is defined as follows:

$$J(\theta) = E_{\Delta \sim p^{\theta}(.)}[R(\Delta)]$$

Where $R(\Delta)$ represent the performance of update rule $\Delta$ on target network.

## 3. Experiment

In the experiment, we aim to implement this method and try to reporduce the result the author got. The implementation contains three parts: Controller network, Target network, and Reinforcement learning. In the training process of controller network, each episode contains 1) get probabilities with softmax of each prediction (forward), 2) sample an update rule, use the update rule to train target network, get accuracy, calculate reward, 3) apply policy gradients to update parameters of controller network (back propagation). We will further explain the details

### 3.1. Controller network implementation

We have implemented a RNN in tensorflow(4) with a fixed time step of 5 in each episode. We use nascell(5) as rnn cell. In the architecture of RNN we implemented, the output of each layer will be the next layer's input vector. The initial input (only first episode) is given by user with no matter what values. Hidden state size is set to 32 by default. The RNN is trained with Adma optimizer with a learning rate of $10^{-5}$. We also use R2 regularization loss to avoid overfitting, the R2 regularization coefficient is set to $0.001$.

### 3.2. Target network implementation

The target network is a small 2 layer 3x3 ConvNet. We use update rule generated by controller network on the target network. Because we could not know which learning rate is the best for the generated update rule, we separately train the target network in one epoche with the learning rate differs from $10^{-5}$ to $10^{1}$, and test their accuracies on held-out validation set. We then use the learning rate with best performance to train the target network for 5 epochs, and test on validation set. Finally the this returns the accuracy on valid set as a signal of reward.

### 3.3. Reinforcement learning

The experiment is built on reinforcement learning framework as described in the beginning. We set the maximum episodes 2500. There are a few details about implementations.

First, the idea of policy gradient is to encourage the actions with positive reward and punish the actions with negative reward. If we use accuracy as reward, this will actually cause problem because each time target network will feed back a positive reward. So the idea is to estimate an average accuracy, and compare each accuracy with the estimated accuracy. The way to estimate an average accuracy is incremental with a learning rate 0.1. It is basically the same form of equation that we estimate a value of state in TD(0).

Second, as we know that policy gradient is easy to stuck in local optimum, we actually use two mechanisms to help. We have used $\epsilon$-greedy, with the initial exploration rate at 0.8. After every 20 episode, exploration rate decay by 0.03 after every 20 episodes. And the action is also generated by softmax prediction, this also add some schocastics to encourage exploration.

## 4. Expriement result

We train the controller network as well as target network on Amazon EC2, with 1 GPU- Tesla V100, 16GB GPU memory, 8 CPU, 61GB memory. But this is not enough to train over 2500 episodes. Although we have realized the entire training system mentioned in the paper, within the time budget we could only train 70 episodes. Comparing to the experiment the author did in the paper, he has used 100 CPUs to do distributed computation on different target nets simultaneously and finally train around 45000 episodes. So at the end we could not get the result the author have. In the 70 episodes, the best action generated is $\Delta w = 1 * [sign(ADAM) * (1 * v)]$, which have the accuracy of 0.5219 on the held-out validation set. This result already outperforms SGD with accuracy 0.4897. The visulized report can be found in \log directory of source code

in tensorboard. (see figure 4) Given the the training process is still in exploration, the graph does not make much sense.

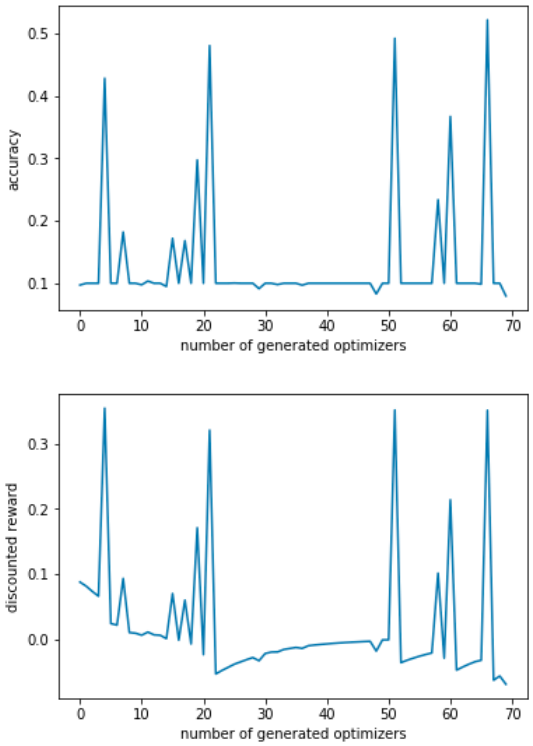


*Figure 4.* Controller reward and target network accuracy evolving over time

## 5. Conclusion

In this project although we couldn't reproduce the author's result due to lack of computation resources, we have realized the entire training process using Policy Gradients described in the paper. We have learned a lot in this project. In this project we have combined Reinforcement learning along with Recurrent neural network and Convolutional neural network. For the first time, we have used tensorflow to implement the recurrent neural network, which helps not only to improve our programming skills, but also to enhance our understanding to computation process of RNN.

What's more, this paper has proposed an interesting aspect to use policy gradients, use controller network and generate actions and updated by reward. We have seen lots of similar ideas from other papers, like using reinforcement learning to design neural network architechture(5), imporving dialogue system(6), trading(7). The future of reinforcement learning in decision making problems will be promising. In the future we will continue to follow the activities.

## Reference

[1] Irwan Bello; Barret Zoph; Vijay Vasudevan; Quoc V. Le, Neural Optimizer Search with Reinforcement Learning. In International Conference on Machine Learning, 2017.

[2] Kingma, Diederik P. and Ba, Jimmy. Adam: Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. In International Conference on Learning Representations, 2015.

[3] Baker, Bowen, Gupta, Otkrist, Naik, Nikhil, and Raskar, Ramesh. Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167, 2016.

[4] Abadi, Martn, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, Kudlur, Manjunath, Levenberg, Josh, Monga, Rajat, Moore, Sherry, Murray, Derek G., Steiner, Benoit, Tucker, Paul, Vasudevan, Vijay, Warden, Pete, Wicke, Martin, Yu, Yuan, , and Zheng, Xiaoqiang. Tensorflow: A system for large-scale machine learning. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2016.

[5] Barret Zoph and Quoc V. Le. "Neural Architecture Search with Reinforcement Learning" in International COnference on Learning Representation, 2017.

[6] Williams, J. D., Asadi, K., and Zweig, G. (2017). Hybrid code networks : practical and efficient end-to-end dialog control with supervised and reinforcement learning. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers), pages 665677, Vancouver, Canada. Association for Computational Linguistics.

[7] John Moody and Matthew Saffell. Reinforcement learning for trading.