




Report of *Prefrontal cortex as a meta-reinforcement learning system*

 Xinyu HUANG ,  Qingyuan YAO , and  Ruohui HU

Master in Artificial intelligence, Sorbonne University

1 INTRODUCTION

This project aims to reproduce and compare the simulations done in the paper *Prefrontal cortex as a meta-reinforcement learning system*, Below is a short summary of work done, including:

- The reading and learning process of extracurricular, but required theories and tools
- The implementation of the model and the simulations
- The analysis of the results obtained and the understanding of the limitations of the meta-model

2 READING

Having finished reading the initial paper, it was soon realized that the methods and simulations are all based on other papers previously written, the paper serves as a comparison between the results of experiments on animals in real life and those in the simulations some modeled by the researchers and some by previous papers. The additional readings include previous implementation of the LSTM with A3C by Mnih et al., as well as precision on the tasks in papers that originally introduced them (Wang et al., Tsutsui et al., Lau et al., Behrens et al., Bromberg-Martin et al., Miller et al., Stopper et al.)

3 LEARNING

As students in the ANDROIDE course, one of us has not attended ML class and none of us have attended AMAL which presents Long Short-Term Memory and TensorFlow. Therefore, self-learning has been an important part of the project. YouTube videos on LSTM and A2C and official documentation of TensorFlow and Keras have been extremely helpful.

4 IMPLEMENTATION

Our work is based on the code of a previous student who build a model to test the Two Step Task. However, as TensorFlow has been updated from version 1 to version 2 and the two have significant differences, the code written in tf1 differs from the tutorials found online and therefore is hard to understand and modify. We had decided to split into two group: one adapted the model to another simulations, the other built their own model in tf2 to reproduce the Two Step Task.

Nevertheless, the two versions of TensorFlow do share the same principles of implementations as below:

- Long Short-Term Memory: Used directly from the TensorFlow library with the *LSTMCell* class, which progresses timestep by timestep. This is essential for Reinforcement Learning, since rewards and observations are obtained and fed into the networks between timesteps during the training.
- Advantage Actor-Critic: The Actor selects behaviors based on probability like a policy gradient, and then uses a value-based algorithm based on the scores of the Actor's behaviors to judge the behaviors as Critic, and the Actor modifies the probability of selecting behaviors based on the Critic scores. And we used an estimate of the advantage of action a_t in state s_t , $A(a_t, s_t) = Q(s_t) - V(s_t)$, to scale the policy gradient, because R_t is an estimate of $Q(a_t, s_t)$. This algorithm estimates both a value function $V(s)$ and a policy $\pi(s)$.
- Gradient Descent: Along with minimizing the loss of actor value, critic value, the model aims also to minimize entropy, which helps the model to make decision and prediction with

more certainty. As the loss function takes in more than just actions and rewards, which are usually given to a fit function in TensorFlow. A customized gradient descent is required, here we take inspiration from the formulas given in the paper as well as the code from the previous student. In a GradientTape, the loss value is calculated and then derived to the gradient for all the trainable variables, which is then applied to the model by the *RMSprop* optimizer with the learning rate of $7e-4$.

4.1 TensorFlow v1

4.1.1 Simulation 1

In this experiment, we simulate the PFC's function of keeping recent actions and rewards. The simulation is based on real-life experiments on monkeys, it shows that PFC neurons reconstruct after each trial, and the monkey's choices updates according to recent experience.

This is a two-armed bandit problem, which has two actions, a fixed Baseline Reward probability sum equals 0.5: $P_0(a_L) = 0.5 - P_0(a_R)$ for each episode. For every trial, this reward probability will change according to the number of actions accumulated not chosen last time, so action to choose is $p(a) = 1 - [1 - p_0(a)^{n(a)+1}]$, n_a is the number of action a not chosen from it has been chosen last time.

We use 48 cells for LSTM, the input included the observation, a scalar indicating the reward received on the preceding time-step, and a one-hot representation of the action taken on the preceding time-step. The outputs consisted of a scalar baseline (value function) and a vector with length equal to the number of available action. At first, Adam (Adaptive Moment Estimation) Optimizer was used, but convergence was very, very slow, so we switch to Rmsprop (Root Mean Square Propagation) to solve the problem by adding a decay factor.

We test for 10K episodes, for each episode, we have 50-80 trials. As a result, the model successfully adapts its behavior accordingly and shows similar behavioral patterns and probability observed in the real-life experiment.

4.1.2 Simulation 6

In this experiment, the researchers want to implement the two distinct roles of DA, it can modulate synaptic plasticity, and it can support activity-based RL computations in the prefrontal network by injecting information about recent rewards. We want to simulate this experiment by a neural network using a variant of our original architecture in which two separate LSTMs (48 units each) model value estimate and policy (rather than a single LSTM with two linear outputs), in alignment with standard actor/critic architectures associated with basal ganglia and PFC function.

Based on the code implemented by simulation 1, we added a second LSTM which takes as input the state observation, the last action taken and the reward prediction error that is computed on the basis of the output from the first LSTM. We have implemented the probabilistic reward/reward task, and built the neural network containing these two LSTM. Unfortunately, we didn't get the result expected: with a probability of getting the high reward set to 0.5, and 40000 episodes, we didn't get the expected value of the ratio between the number of high reward action trials and the number of all trials. (Value expected from the figure 7 of the article is around 0.75, but we got 0.56 instead)

4.2 TensorFlow v2

A great length of time has been used on implementing the model, especially the loss function. The structure was taken from an example Actor-Critic RL model that plays CartPole on the website of Keras. The LSTM network was first built with the LSTM class which takes in inputs of every previous timestep, it is then realized that, the LSTMCell is much more optimized and faster for RL. The loss function has been modified many times, despite having access to the formulas in the paper, the tutor has warned us of a possible error which deviates from how Mnih et al. originally implemented. A lot of trials and errors has gone into the calculation of temporal difference and the "filters" for the actor and critic loss. It is finally decided to take the previous student's calculation as the reference, and promising results have been obtained.

For easier training and adapting to other environments, the code has been made modular. The training process is done by an Agent class which takes all the hyperparameters and one of the Env_... class that inherits Env_Model. The notebooks import these classes and define the number of episodes and trials, as well as hyperparameters, before training and testing. (The testing process is more complex for the Simulation 2 and 3, the code has been copied from Agent and modified in the corresponding notebooks)

4.3 Simulation 4

The environment has been taken directly from the previous student and passed into the model. Upon choosing an action (left/right), the subject has certain probability to pass to a state or another, which also have their own probabilities to get a reward. moreover, at the start of each trial, there is a probability of the two states switch their probabilities of rewards.

Just as the previous student has observed, the reward starts to improve very late into the training. For this, we have also augmented the number of episodes to 20k. The actor and critic loss managed to converge to 0 around 5k episodes, but the entropy descends much slower and never reaching 0 even after 20k episodes, this could be one of the explanations of why the reward starts improving around 5k and 10k episodes.

The result showed the same phenomenon where the stay probabilities of common and uncommon transition differs from each other whether in rewarded or unrewarded situations. This proves the model-based behavior of the meta-RL network despite being model-free. However, the fact that whether LSTM itself is a general model-based network is still up to the debate.

Variations: After obtained the results with hyperparameters from the paper ($\gamma=0.9$, learning rate= $7e-4$), some tests with different combination of hyperparameters ($\gamma \in [0.9, 0.75]$, learning rate $\in [7e-4, 0.1]$) have been made, and it is observed that the learning rate affect more the performance than the discount value.

Different "filters" of the actor and critic loss have been experimented while and after the tuning of the loss function. It is confirmed that the performance of Two Step Task only improves while applying the Huber Loss on actor loss ($\text{actor_loss} = 0.5 * \text{tf.square}(\text{actor_loss})$) and inversion on critic loss ($\text{critic_loss} = -\text{critic_loss}$).

4.3.1 Other Simulations

After fine-tuning the loss function and obtained the expected result in Simulation 4. Applications on other simulations have been made, though all unfinished in various degrees:

Simulation 1: The environment was taken from the other group and trained by the model. The sentence "n(a) is the number of trials since action a was chosen." was interpreted differently. We have modified the model in order to test 3 interpretations:

1. number of trials since action a was **last** seen (counting the current trial as 0 or 1)
2. number of trials since action a was first seen
3. the total number of trials where action a was chosen so far

The first interpretation, counting the current trial as 1 gives the closest result to the paper. However, the distribution of points expand between $[-7.5, 7.5]$ instead of $[-4, 4]$.

Simulation 2: The environment has been coded and the model has been trained. The test and graph couldn't be made as the calculation of the actually learning rate requires building another Bayesian model.

Simulation 3: The environment has been coded and the model has been trained. However, it is later realized that each trial need to be separated into exactly 4 timesteps in order to observe the spikes of RPE. The time didn't allow a rebuild of the environment and the training agent, nor the training and testing.

5 CONCLUSION

In the past two months, we did our best to understand the article, the A2C and LSTM algorithms, and restructure the experiments, learned to use TensorFlow Framework and view the changes of model variables through TensorBoard, compared the impact of different Optimizer and framework on the model. Due to the limited time, the environment was set up and models were tested for simulation1, 4 and 6, but simulation 2-3 could not be tested because of the need for additional test models, and further learning and understanding of this will be done later. we would like to thank Professor Mehdi Khamassi for his email, video and offline guidance during the whole project completion process.

6 REFERENCE

- [1] J. X. Wang et al., "Learning to reinforcement learn," ArXiv, vol. abs/1611.05763, 2016.
- [2] J. X. Wang et al., "Prefrontal cortex as a meta-reinforcement learning system," Nature Neuroscience, vol. 21, no. 6, pp. 860–868, Jun. 2018, doi: 10.1038/s41593-018-0147-8.
- [3] K.-I. Tsutsui, F. Grabenhorst, S. Kobayashi, and W. Schultz, "A dynamic code for economic object valuation in prefrontal cortex neurons," Nature Communications, vol. 7, no. 1, p. 12554, Sep. 2016, doi: 10.1038/ncomms12554.
- [4] C. M. Stopper, M. T. L. Tse, D. R. Montes, C. R. Wiedman, and S. B. Floresco, "Overriding Phasic Dopamine Signals Redirects Action Selection during Risk/Reward Decision Making," Neuron, vol. 84, no. 1, pp. 177–189, 2014, doi: <https://doi.org/10.1016/j.neuron.2014.08.033>.
- [5] V. Mnih et al., "Asynchronous Methods for Deep Reinforcement Learning," in Proceedings of The 33rd International Conference on Machine Learning, New York, New York, USA, Jun. 2016, vol. 48, pp. 1928–1937. [Online]. Available: <https://proceedings.mlr.press/v48/mniha16.html>
- [6] B. Lau and P. W. Glimcher, "Dynamic response-by-response models of matching behavior in rhesus monkeys," Journal of the experimental analysis of behavior, vol. 84, no. 3, pp. 555–579, 2005.
- [7] E. Bromberg-Martin, M. Matsumoto, S. Hong, and O. Hikosaka, "A Pallidus-Habenula-Dopamine Pathway Signals Inferred Stimulus Values," Journal of neurophysiology, vol. 104, pp. 1068–76, Aug. 2010, doi: 10.1152/jn.00158.2010.