CS 394R Reinforcement Learning: Theory and Practice
Final Project Literature Survey

Wei Shi (EID: ws8675)
Ge Li (EID: gl9476)

1 Introduction to the problem

In this project, we aim to explore playing *Boxing*, which is one of the Atari 2600 video games, with reinforcement learning algorithms. The game shows a top-down view of two boxers, one white and one black. When two players are close enough, one boxer can hit his opponent with a *punch*, which causes his opponent to real back slightly. Punches with a long distance will receive one point, while closer punches receive two points. The game ends either when one player lands 100 punches (*knockout*) or two minutes has passed (*decision*). In the case of *decision*, the player with the most punches is the winner. Draw game could be a possible case.

2 Solving the problem with RL

2.1 Sequential process of the problem

The *Boxing* game is made up of sequential frames in time domain. Meanwhile, it is perfectly an episodic task. The episode is essentially a match. Termination occurs when a match is completed.

2.2 State space of the problem

The state space is a complete set of all frames in terms of RGB images captured from Atari screen. The dimension of images is [210, 160, 3], and each pixel has 256 possible values. Each image includes the following information: the location coordinates of the agent, relative location with respect to the opponent, whether or not hitting the opponent and whether or not being hit.

2.3 Action space of the problem

In the Gym environment, the action space of all the Atari 2600 video games is generalized as 18 possible actions: *noop, fire, up, right, left, down, up_right,* *up_left, down_right, down_left, up_fire, right_fire, left_fire, down_fire, up_right_fire, up_left_fire, down_right_fire, down_left_fire*. For the *Boxing* game specifically, fire corresponds to punch, and two players can move in different directions. Since the players will always punch in the same direction (white boxer punches to the right and black boxer punches to the left), the *up_fire, right_fire, left_fire, down_fire, up_right_fire, up_left_fire, down_right_fire, down_left_fire* actions are not actually utilized in the *Boxing* game. Therefore, the actual action space of the problem is 10.

2.4 Reward function plan to use

The default reward of *Boxing* game is based on score rules. If our agent hit the opponent with a long punch, a +1 reward will be given; if hitting the opponent with a power/close punch, a +2 reward will be given. Similarly, if our agent is hit by a long punch, we get -1 reward; if being hit by a power/close punch, we get -2 reward. We may also plan to exploit more sophisticated reward schemes to improve the performance of the algorithms.

3 Goal of the project

3.1 Starting point of the project

Playing Atari 2600 video games with reinforcement learning algorithms have been well exploited. Our main target is to implement several different RL algorithms to *Boxing* game, either covered in the textbook or state-of-the-art, and exploit the trade-offs between these algorithms to maximize our learning experience. Since the environment of Atari games is well defined in Gym library, our starting point will be getting familiar with the Gym library, prototyping algorithms from the textbook and performing literature review for state-of-the-art algorithms.

3.2 RL algorithms plan to use

We plan to exploit the following algorithms:

- **DQN**: Deep-Q-learning. A CNN is utilized to do function approximation. The input to the CNN is an image, which corresponds to a specific frame in the game. **Implement a minimalistic version**

- ~~**Monte Carlo Methods**: Use hand-selected feature construction methods to perform function approximation.~~

- ~~**Q-learning**: Use hand-selected feature construction methods to perform function approximation.~~

- **A3C**: Asynchronous advantage actor-critic algorithm.**Implement a minimalistic version**

- ~~**NoisyNet A3C**: NoisyNet for exploration, in combination with A3C algorithm.~~

3.3 Baseline

- ~~**Q-learning**: Use hand-selected feature construction methods (eg: take into account of relative location of two boxing players, and explore polynomial basis, radial basis, tile coding or neural networks).~~
- ~~**Random policy**: At least we expect the algorithms to perform better than a random policy.~~

We plan to implement DQN and A3C from scratch and compare with each other.

3.4 Initial goal of the project

~~Our initial goal is to implement Q-learning with hand selected feature construction methods, and DQN, which is the classic algorithm to play Atari games from literature. After implementing these target algorithms, we will compare their performances and discuss trade-offs behind them.~~

Our initial goal would be implementation of DQN and A3C from scratch.

3.5 Stretch goal of the project

Our stretch goal is to finish the implementation of the rest of the algorithms: Monte-Carlo Methods, A3C and NoisyNet A3C, and include these algorithms into our trade-off discussion as well. In addition, we are also interested in incorporating current researches and challenges in reinforcement learning with our specific problem.

- ~~**Algorithm acceleration**: We wish to fit computation-hungry algorithms into limited computation resources such as PC without GPUs or mobile phones. Such fitting possibly includes modifications on neural network, customized reward function and so on.~~
- ~~**Human coach assistance**: We wish to accelerate the training process with the assistance of human expert. The key challenge is how to incorporate human expertise into the iteration flow.~~
- ~~**Multi-agent RL**: We wish to make both boxing players to be RL agents. Such multi-agent setting may bring better performance.~~
- **Transfer-Learning**: Many games have something in common. We wish to develop a way to put a pre-trained agent from one game into other games, which may shorten the time of training in the new game.
- **NoisyNet A3C: NoisyNet for exploration, in combination with A3C algorithm. Implement a minimalistic version**

4 Related Work

4.1 Deep Q-Learning

In [1], a new deep learning model for reinforcement learning has been introduced, which is able to master control policies for Atari 2600 games. The key point of this paper is to use experience replay to ease the training of deep neural network. In the meantime, it introduced several techniques in both problem formulation and solution. First, atari games are not standard since we cannot fully understand the current situation from only the current screen image. So complete state and action sequence has been chosen to represent state at time t. Second, experience buffer is used to execute random sampling, reducing update variance and avoiding divergence. Third, even the number of pixels in atari screen is not large, input dimensionality reduction is still needed. Raw screen images will be downconvert to the gray scale image. Only four frames from the sequence will be fed to neural network each time. And differentiating from q-learning, states are the input of neural network instead of state-action pair.

In [2], Wang *et al* proposed a new neural network architecture compared with original deep reinforcement learning work [1]. The intuition is that for many states, it is unnecessary to estimate the value for each action, but it is important to know the state value estimation since it is required in bootstrapping based algorithms. Compared with the neural network architecture in [1], which takes images as input and output the Q value for each action, the proposed dueling network architecture not only calculates Q value for each action, but also outputs the state value estimation for given state. Specifically, the dueling network shares similar convolution layers as [1], but diverges to two streams in fully connected layers. One stream calculates state value estimation and the other one calculates advantages for each action, which could be combined with state value to calculate Q values for each action. The experiments show that dueling architecture can more quickly identify the correct action to take during policy evaluation.

Q-learning is a bootstrapping method and is known to show bias in estimation [3]. The estimate for the action value is based on the current reward received and the maximum Q value for the next state. Since both the action selection (greedy policy) and action evaluation are based on max operation of current value estimation, this may lead to an overestimation of action values. The overestimated value may lead to sub-optimal policies. A way to solve this problem is to utilize double Q learning, where max operation is decoupled into action selection and action evaluation [4]. Therefore, two value functions are learned by assigning each experience randomly to update one of the value functions, such that two sets of weights will be obtained. For each update, one value function is used to select action based on greedy policy, while the other is used to determine the action value. To apply double Q learning to DQN, a natural way is to decouple the max operation to target network and the online network that is being learned. The online network is used to select action following greedy policy, while the target network is used to estimate action value.

In [5], the author stated that it is more efficient to do update when focusing on some useful state-action pairs. The measure of usefulness or urgency could be reward or other function. Therefore, state-action pairs with their priority can be stored with a priority queue. Each time when agent does planning, it will first pick the top element in priority queue to start simulation. Such prioritized sweeping technique in planning algorithms is believed to improve efficiency especially in real-world problems.

In [6], The author found that prioritized experienced replay can speed up learning by factor of 2, leading to a new state-of-art of performance on the Atari benchmark. Similar to what people have done in the planning algorithms, TD-error can be a good way to measure priorities. But either TD-error prioritized replay or uniform-sampling replay has many issues. This paper defined a new probability of sampling a certain transition. And author called it stochastic prioritization, which interpolates between greedy prioritization and uniform sampling. One issue brought by prioritized replay is extra bias. To fix this, the author introduced importance sampling. Moreover, this paper exploited the flexibility of annealing the amount of importance sampling correction over time.

4.2 Policy Gradients

In [7], the author stated that policy-based model-free methods can update parameters of policy directly. While in value-based model-free methods, for example DQN, it's hard to update many states' value in one update. The author introduced an asynchronous reinforcement learning framework that can work for both value-based and policy-based methods. The key point here is that using parallel agent learners to update a shared model had a stabilizing effect on learning process. The algorithm for each actor learner in each thread is similar to previous work. But different actor learners update on a shard parameter set with their own accumulated gradients asynchronously.

To encourage exploration, Fortunato *et al* [8] proposed noisy net, which injects random perturbation to the weights and biases of the neural network. These perturbations are based on a parametric perturbation of noise, and the parameters are adapted with gradient descent. Specifically, the original weights and biases are characterized by μ, Σ and ϵ, where μ, Σ are learnable parameters by gradient descent and ϵ is a random noise vector with zero mean. Therefore, for linear layers in neural network, the weights and biases and replaced by expressions respect to μ, Σ and ϵ. In order to determine the value of μ and Σ, in the training process, gradient of loss function with respect to μ, Σ is calculated to update the value of μ and Σ in a gradient descent manner. Noisy net is adaptable to many deep RL algorithms with little computational overhead, including A3C [7], DQN [1], Dueling DQN [2] and double DQN [4], and is able to achieve striking gains due to the benefit of exploration.

4.3 Transfer and Multitask RL

In [9], the authors achieved two features: First, prevent the catastrophic forgetting by instantiating a new neural network for each task being solved. Second, transfer via lateral connections to features of previously learned columns. Compared with conventional Fine-tuning method, this Progressive Neural Network doesn't have to assume that there is an overlap/similarity between source tasks and target task. In the meantime, the authors used Average Perturbation Sensitivity and Average Fisher Sensitivity to analyze the contribution from the previous learnt neural network to current tasks.

In [10], the authors focuses on some auxiliary signals from the environment that could be beneficial to the agent's primary goal. These auxiliary tasks are formulated as separate control tasks, which are learned by the agent simultaneously while trying to maximize cumulative rewards. Specifically, the authors focus on two types of auxiliary tasks: (1) *Pixel Control.* The changes in the image pixels often indicates important events in an environment. The agent is trained to maximize the changes in pixels. (2) *Feature Control.* The policy or value network extracts task-relevant high level features from the environment, and therefore the input to hidden neurons of the hidden layers could be viewed as auxiliary rewards. In this control task, the agent is trained to maximize the activation to each hidden unit. In the work, these two auxiliary tasks are combined with traditional A3C [7], and the agent learns the three control tasks in the same time.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[2] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

[3] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *Second Edition*, page 131-132, 2018.

[4] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[5] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *Second Edition*, page 168-171, 2018.

[6] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray

Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[8] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. *CoRR*, abs/1706.10295, 2017.

[9] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

[10] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *CoRR*, abs/1611.05397, 2016.