Neural Network Agents for Control Tasks in OpenAI Gym

Shengjun (Sophia) Qin
Department of Electrical Engineering
Stanford University
Email: sjqin@stanford.edu

Junkai Jiang
Department of Electrical and Computer Engineering
University of California, Santa Barbara
Email: junkaijiang@ece.ucsb.edu

***Abstract*—In recent years, reinforcement learning has been reported to be successful in many fields that involves highly non-linear systems. It is also promising in developing automated agents that deal with complexed tasks, without lots of human efforts. Based on an open-source simulation framework, multiple reinforcement algorithms, including policy gradient, deep Q-network (DQN), deep deterministic policy gradient (DDPG) and advantage actor-critic (A2C) method, and control tasks, including discrete and continuous action spaces, are explored in this project. Different neural network (NN) architectures in policy network are also explored. The results indicate that a wider NN is preferred for efficient agent training.**

I. Introduction

Reinforcement learning (RL) [1] is a branch of machine learning that is concerned with making sequences of decisions. RL is different from supervised learning, where a prediction model learns from a training set with labels. Instead, RL learns from an interactive process, and trains an agent that gives actions, based on current state, to achieve the most reward. The RL has demonstrated wide applications in different fields, including data-center resource management [2], robotics [3] and solving different games like the AlphaGo [4]. The OpenAI Gym [5] provides a collection of benchmark problems in multiple scenarios for developing RL algorithms. In this project, we design and train neural-network (NN) based agents to solve multiple control tasks provided in the OpenAI Gym, and use reinforcement learning algorithms to train these agents to achieve desired performance.

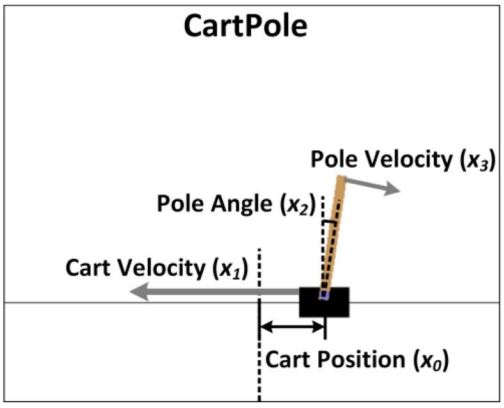

Fig. 1 Cart-Pole simulations from OpenAI Gym. The observation (x_0 – x_3) of the simulation consists of cart velocity, cart position, pole angle and pole velocity, respectively. The control agent gives binary instructions of left and right moves, based on the observation, and avoids (1) falling of the pole, or (2) cart moving out of the boundary.

II. Policy Network and Reinforcement Learning

Unlike supervised learning, where training is based on a dataset with desired output, the idea of reinforcement learning is learning what to do, without being told what the direction action is, to maximize rewards [1]. A policy defines the agent's action, given the observation of the environment, and a reward is the goal of the environment and is immediately given to the agent after its actions at each timestep. In this section, we explore and show the need for neural network, a highly non-linear system, policy/agent, and the reinforcement algorithms, including policy gradient and deep Q-network (DQN), to effectively train the neural network agent, in the environment of CartPole [5]. In CartPole, the task is to train an agent that gives actions (whether to move left or right, encoded by 0 and 1), based on observations (**Fig. 1**), so that the reward, which is the number time steps that the cart-pole stays stable, is maximized.

A. Neural Network based Agent

A simple random selection between 0 and 1 will result in a very unstable system. Additionally, a naïve agent that chooses actions only based on the pole angle (move left/right when the pole angle is negative/positive, respectively, also results in low

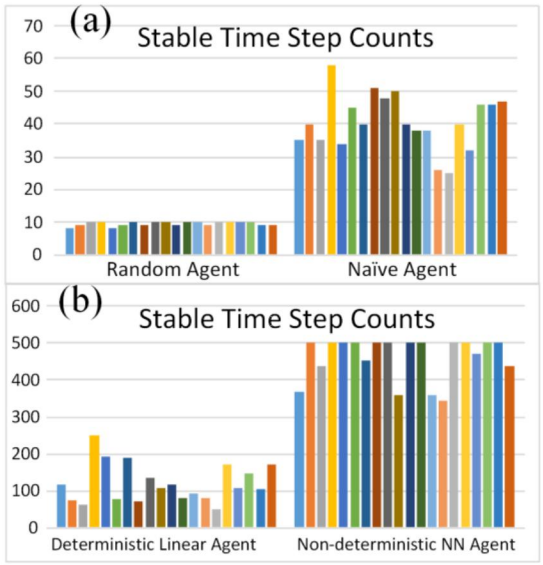

Fig. 2 Stable time step counts in 20 experiments for (a) agents of random actions (left) and naïve actions (right), (b) deterministic linear agent (left) and non-deterministic neural network (NN) agent, both trained by policy gradient. Note that the maximum time step in each episode is 500.

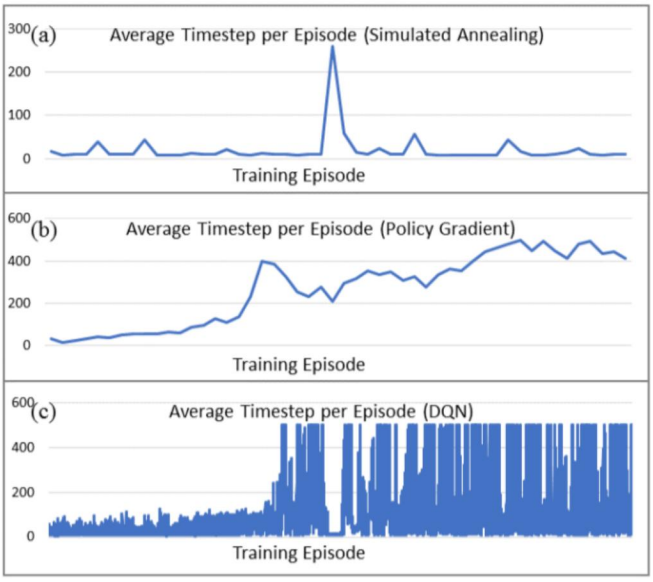

Fig. 3 Average timestep per episode in training for (a) simulated annealing optimization, (b) policy gradient and (c) deep Q-network (DQN). The noise in (c) is from ε-greedy policy to explore policy search space. Note that the maximum time step in each episode is 500.

reward (**Fig. 2a**), indicating the 4 inputs are needed in determining the proper action. We also explored a simple deterministic linear agent, based on a simple linear combination of the observation vector and a non-deterministic neural network agent with one hidden layer (Fig. 2(b)). The two agents are trained by policy gradient, which will be described in the next part. Neural network agent, obviously, performs better than the deterministic linear agent, most likely because of the non-linear nature of CartPole environment. Thus, in the following experiments, we use neural network agents.

B. Optimization in Model-Free Environments

Under the assumption of Markov decision process, the current reward and the expected discounted reward is only dependent on the current observation or state and action. The optimal action is the action that leads to the most expected discounted reward. In a model-free environment, the reinforcement learning is to find the state value (Q) function that predicts the expected discounted reward, based on current observation and action. According to our experiments, random search or traditional optimization algorithm, such as simulated annealing, does not lead converged solutions within reasonable time steps. On the other hand, reinforcement algorithms, including policy gradient and DQN, offers convergence (**Fig. 3**) and good trained agent performance.

III. Simulations and Environments Setup

The OpenAI Gym offers multiple control simulation environments, ranging from discrete-action-space environments like CartPole and Atari Arcade games to continuous-action-space environments like Roboschool simulations [5]. Stable Baselines [6] provides a set of reinforcement learning algorithms that are used in this project.

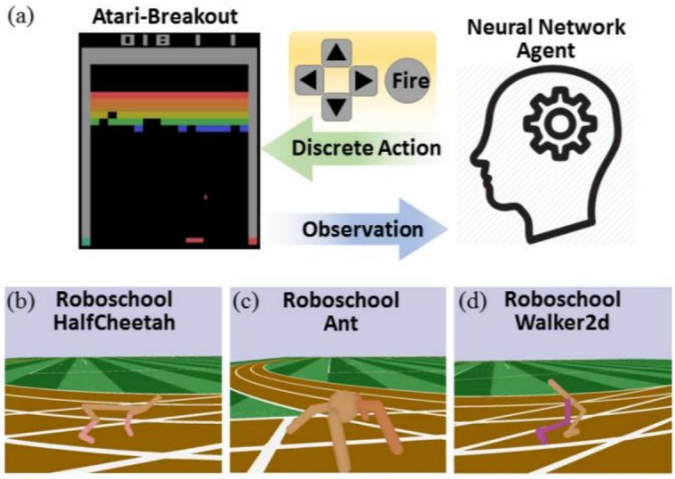

Fig. 4 (a) Schematic showing a neural network agent takes in images from Atari-Breakout game and gives discrete actions. The continuous action space environments studied in this project are (b) HalfCheetah, (c) Ant and (d) Walker2d. The aim in (b)-(d) is to run as fast as possible.

A. Discrete and Continuous Action Spaces

In some control environments from OpenAI Gym, the agent only needs to give discrete actions. For example, in the Breakout game from Atari Arcade (**Fig. 4a**), the actions are only combinations of a single or multiple of the buttons. In this case, the agent or the policy network takes the input of the environment observation and gives the output Q-function of each discrete action in the current time step, as in DQN [4]. In other environments, the action space is continuous, as shown in Roboschool robot control tasks (**Fig. 4b-c**), and the agents are trained by deep deterministic policy gradient (DDPG) [7] in this project.

B. Reinforcement Learning Algorithms

A basic reinforcement learning model is shown in **Fig. 5**, where an agent (e.g. a human) observes the environment and takes actions based on certain algorithms. Various types of reinforcement learning algorithms are studied to train the agent and decide how to interact with the environment within the next action. Different algorithms would be desired for different tasks and limited by specific constraints and the progress of relevant research. In this work, we explore Policy Gradients, Actor-Critic methods (DDPG and Advantage Actor-Critic (A2C)) and Q-learning with DNN (DQN).

The Policy Gradient method is one of the major categories of reinforcement learning algorithms and focuses on the policy to make better rewards happen with higher possibilities. Q-learning is an action-value function learning with an exploration

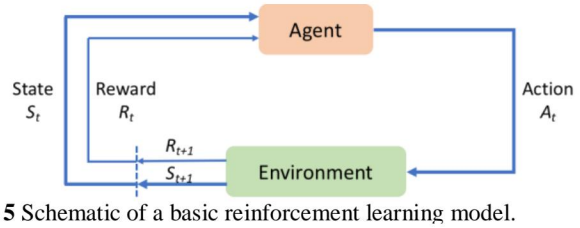

Fig. 5 Schematic of a basic reinforcement learning model.

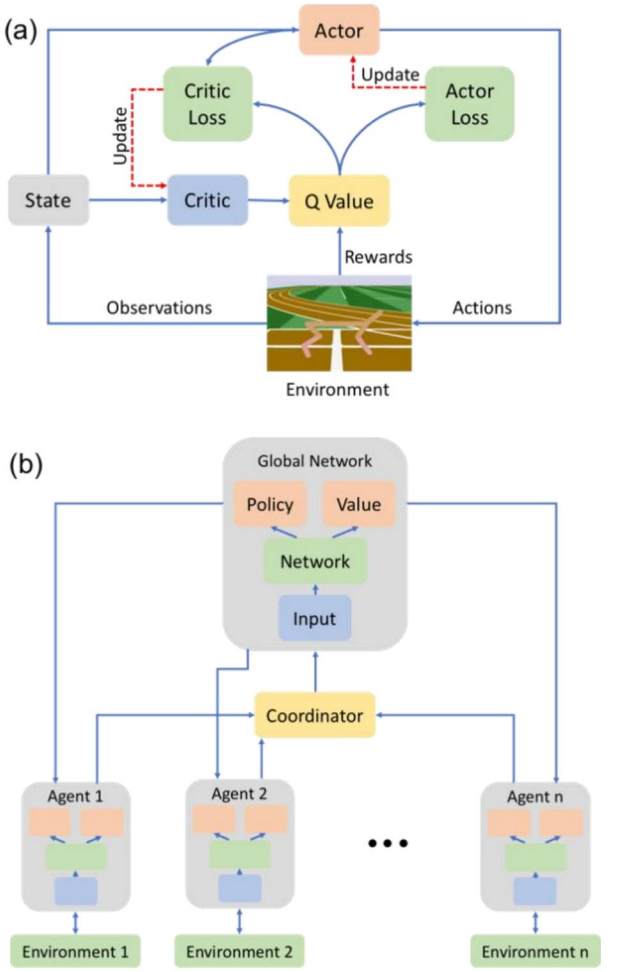

Fig. 6 Schematics of (a) Deep deterministic policy gradient (DDPG) and (b) Advantage Actor-Critic (A2C) methods.

policy. DQN is a Q-learning method adopting deep neural network to estimate Q-values using a replay buffer and a target network. While Policy Gradient methods need large number of samples to achieve an optimal result, the Actor-Critic methods require less samples and use an actor to model the policy and a critic to model the value. DDPG and A2C are two methods in the Actor-Critic method category (schematics in **Fig. 6**). DDPG extends DQN to continuous action space by introducing another actor network to pick the best action and using critic network to update Q-values. A2C is a synchronous, deterministic implementation that updates with the average over all of the actors after all finish the experience.

IV. Neural Network Agnets in Control Tasks

We explore the effects of different neural networks in control tasks. For continuous action space, we build and experiment deep neural networks (**Fig. 7a**) with different layers (from single layer to three layers) and different numbers of neurons (64 and 128). The training results are compared for the Walker2D environment. We also implement a typical convolutional neural network [4] to train agents on Atari Arcade games. In this CNN model we use 3 Conv layers with strides followed by fully-connected layers.

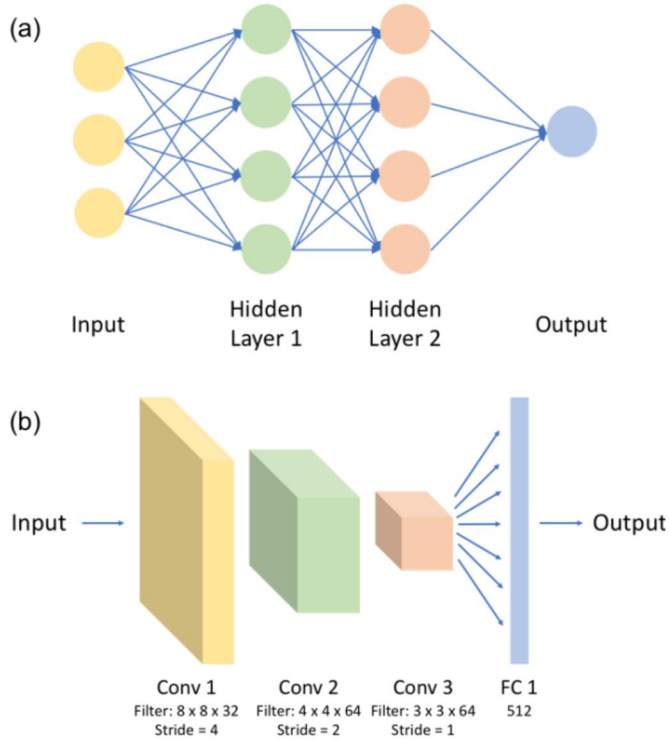

Fig. 7 Schematics of (a) Deep Neural Network (DNN) and (b) Convolutional Neural Network (CNN).

A. *Deep Neural Network Agents*

In an environment, where the observation can be encoded into one-dimensional vectors, such as Roboschool control tasks, deep neural network (DNN) is used in the control agent. Exploration of DNN width and depth will be discussed in Section V.

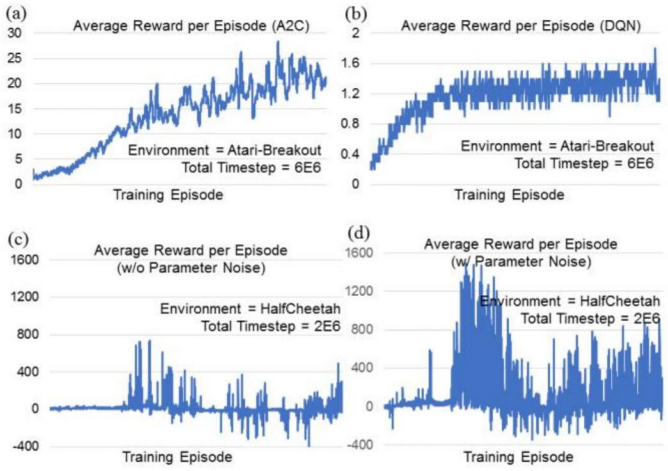

Fig. 8 Average reward per episode during training for (a) Atari-Breakout CNN agent by A2C, (b) Atari-Breakout CNN agent by DQN, (c) Roboschool-HalfCheetah by DDPG without parameter noise, and (d) Roboschool-HalfCheetah by DDPG with parameter noise.

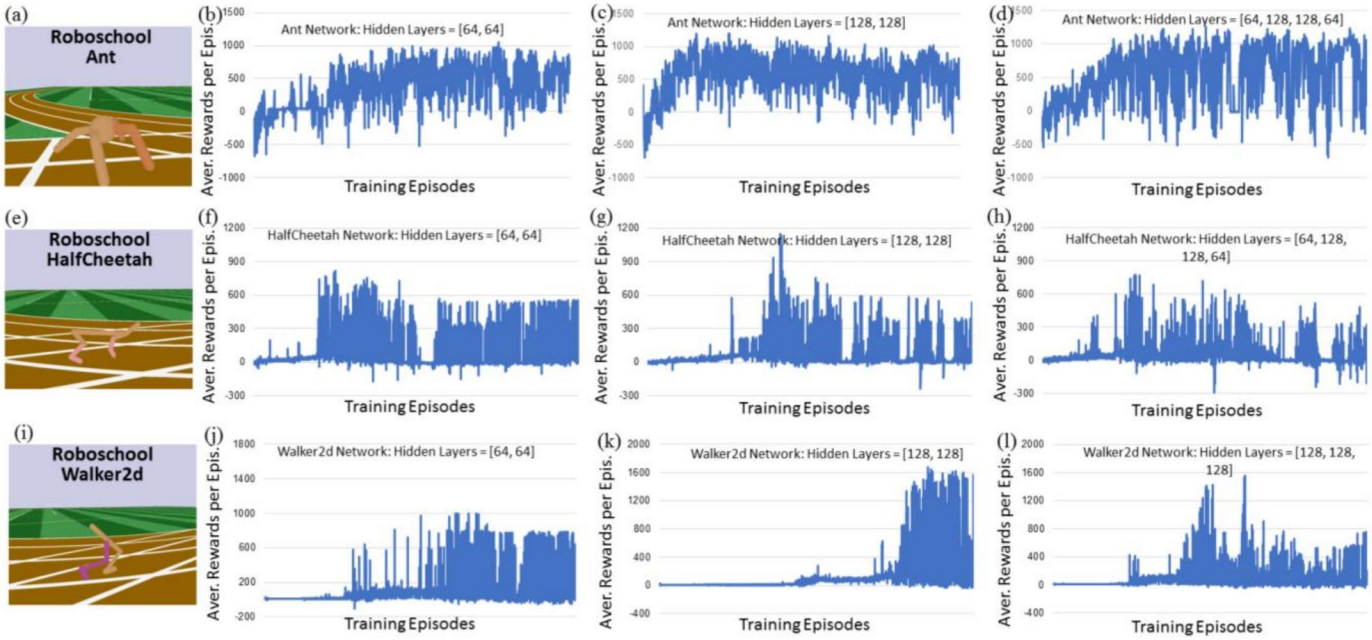

Fig. 9 Average reward per episode during training for (a)-(d) Roboschool-Ant with DNN hidden layers = (b) [64, 64], (c) [128, 128] and (d) [64, 128, 128, 64], (e)-(h) Roboschool-HalfCheetah with DNN hidden layers = (f) [64, 64], (g) [128, 128] and (h) [64, 128, 128, 64], and (i)-(l) with DNN hidden layers = (j) [64, 64], (k) [128, 128] and (l) [128, 128, 128].

B. Convolutional Neural Network Agents

In an environment, where the observation cannot be easily encoded in to a vector, such as Atari Arcade games, convolutional neural network that takes inputs of 2D images is used in the control agent. In this project, we only applied the CNN network structure reported in [4], because of limited computing resources and time.

V. Results and Discussions

Based on the environments in OpenAI Gym and the Stable Baselines framework, we are able to explore different reinforcement algorithms in different control tasks. Overall, we have completed A2C and DQN training on Atari-Breakout, DDPG training on Roboschool-HalfCheetah, Roboschool-Ant and Roboschool-Walker2d, and their variations in neural network structures. The results and discussions of these experiments are summarized in this section.

A. RL Algorithm Dependency

According to our simulations, different reinforcement algorithms lead to drastically different agent performance. For example, A2C and DQN trainings are applied in Atari-Breakout (**Fig. 8a,b**) in 6 million timesteps, and agents trained by A2C and DQN give > 25 and < 2 average rewards per episode, respectively. The reason for the weak agent by DQN is the unoptimized hyperparameters. In other words, the training efficiency is very sensitive to the hyperparameters and training settings. Other experiments also support this in DDPG trained Roboschool-Halfcheetah agent (**Fig. 8c,d**). There is a large agent performance gap (average reward per episode: < 800 *vs.* $>$ 1,500) for without and with parameter noise. The parameter noise is to implement noise in network parameters (weights and biases) to more effectively explore the policy search space.

B. Network Dependency

Average rewards per episode during training are plotted for Roboschool-Ant, Roboschool-HalfCheetah and Roboschool-Walker2d with different DNN network widths and depths are plotted in **Fig. 9**. There is an improvement of the average rewards by increase the network width from 64 to 128, for all the three environments, indicating the original hidden layers of [64, 64] are not sufficient to provide the non-linearity in the Roboschool control tasks. However, there is no obvious improvement by increasing the depth of the DNN. We suspect this is because of the less efficient training in back propagation with deeper networks, in reinforcement learning, where training dataset is limited.

VI. Conclusions

In this project, multiple reinforcement learning algorithms, including policy gradient, DQN, A2C and DDPG, are explored in training control agents for both discrete and continuous action spaces, thanks to the availability of open-source frameworks [5],[6]. It is identified that neural network agent and reinforcement learning are necessary to train a strong agent in non-linear control tasks. Reinforcement learning is sensitive to the hyperparameters and training settings, according to simulation results. Wider DNN is found to improve training efficiency, whereas deeper DNN is not shown such benefits.

VII. Contributions and Acknowledgements

Shengjun Qin and Junkai Jiang shared joy of working together and contributed equally to this project. The authors would like to thank their TA Jay Whang for helpful discussions.

REFERENCES

[1] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction (Second Edition). Cambridge, MA: The MIT Press, 2018.

[2] H. Mao, M. Alizadeh, I. Menache and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of ACM Workshop on Hot Topics in Networks*, Nov 2016, pp. 50-56.

[3] J. Kober, J. A. Bagnell and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238-1274, 2013.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484, 2016.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "Openai gym," *arXiv: 1606.01540*, 2016.

[6] A. Hill, et al., "Stable baselines," *Github*, 2018, https://github.com/hill-a/stable-baselines.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. of International Conference on Learning Representations*, 2016, pp. 1-14.