Comparison of Deep Reinforcement Learning Approaches for Intelligent Game Playing

Anoop Jeerige, Doina Bein
Department of Computer Science
California State University, Fullerton
Fullerton, CA, USA
anoopjeerige@csu.fullerton.edu, dbein@fullerton.edu

Abhishek Verma
Department of Computer Science
New Jersey City University
Jersey City, NJ 07035
averma@njcu.edu

***Abstract*— In Reinforcement Learning, a category of machine learning, learning is based on evaluative feedbacks without any supervised signals. The paper presents work aimed to understand the deep reinforcement learning approaches to creating such intelligent agents, by reproducing existing research and comparing their results. The project uses the Atari 2600 game called Breakout, in which the agent will learn control policies using deep reinforcement learning approaches to achieve a high score. The project explores two deep reinforcement learning approaches, Asynchronous Advantage actor-critic and Deep Q-Learning, both proposed by the DeepMind team, to train intelligent agents that can interact with an environment with automatic feature engineering thus requiring minimal domain knowledge.**

Keywords—Deep Reinforcement Learning; Neural Network; Policy Gradient; Deep Q-Learning Network; Asynchronous Advantage Actor-Critic; OpenAI Gym.

I. INTRODUCTION

Reinforcement Learning (RL) is a class of machine learning (ML) models where the learning process is based on evaluative feedbacks without any supervised signals [1, 8]. RL aims to create agents similar to the humans, which learn for themselves by trial-and-error, solely from rewards or punishments, to develop successful strategies that eventually lead to the largest long-term rewards. In the past, the success of reinforcement learning tasks for an agent operating on a domain has mostly relied on the hand-crafted feature representations of that domain, thus varying the performance based on the quality of the representation. We can conclude that learning to control an agent solely from high-dimensional inputs such as vision data is one of the main challenges of reinforcement learning due to the large amount of data needed for training. These agents need to observe the state of the environment and take actions in this environment. The agent, for each of the actions taken, receives either a reward or penalty using which it learns to maximize the long-term rewards thus allowing it to be successful in the environment.

We have used in our research the Atari 2600 game called Breakout, in which the agent will train and learn control policies using deep reinforcement learning approaches to achieve a high score. In this paper we compare two existing approaches, Asynchronous Advantage actor-critic and Deep Q-Learning, both proposed by DeepMind team. Our goal is to to create a deep learning model that successfully learns control policies directly from high-dimensional sensory input (such as image data) using reinforcement learning and understand the deep reinforcement learning approaches to creating such intelligent agent.

The paper is organized as follows. In Section II we define reinforcement learning and present significant deep reinforcement learning methods. Description of the project functionality is given in Section III. Simulation results and analysis are presented in Section IV. Concluding remarks and future work are presented in Section V.

II. REINFORCEMENT LEARNING

In RL, an agent learns from evaluative feedback without receiving any supervised signals. An RL agent interacts with an environment as follows: at each time t, the agent receives a state s_t in a state space S and selects an action a_t from an action space A by following a policy $\pi(a_t|s_t)$. The policy is defined to be the behavior of the agent; the policy acts as a mapping from state s_t to action a_t. By performing the action, an RL agent receives a reward r_t and transitions to the next state s_{t+1} based on the environment model. The model consists of the environment's reward function $R(s,a)$ and the state transition probability $P(s_{t+1}|s_t,a_t)$. An RL agent continues the interaction with the environment until it reaches some terminal state and then it restarts. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ of each state is the discounted, accumulated reward with the discount factor $\gamma \in (0,1]$. The agent aims to maximize the expectation of the long-term return from each state. The value function is the central concept to reinforcement learning, as it measures the goodness of a state or a state-action pair. It can be either a state value $v_\pi(s)$ – the expected return for following a policy π from state s or an action value $q_\pi(s,a)$ – the expected return for selecting action a in state s and then following policy π. It is also used as a prediction of the expected, accumulated, discounted future reward. The goal of the agent is to find an optimal policy π^* to maximize the expectations of long-term rewards.

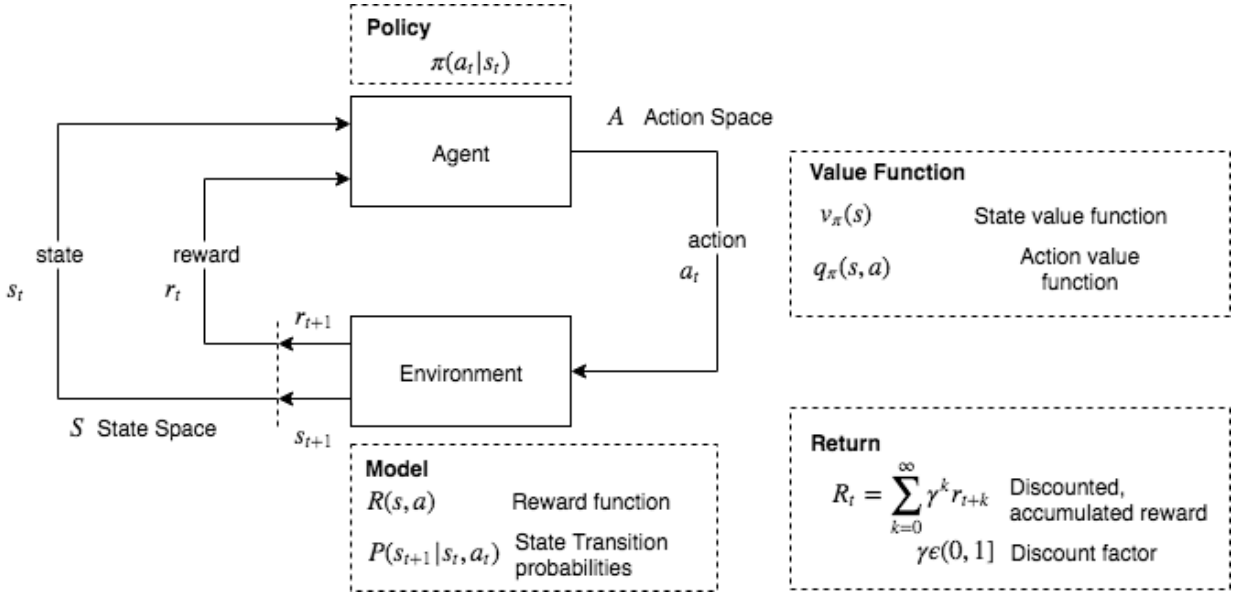

Figure 1. RL problem

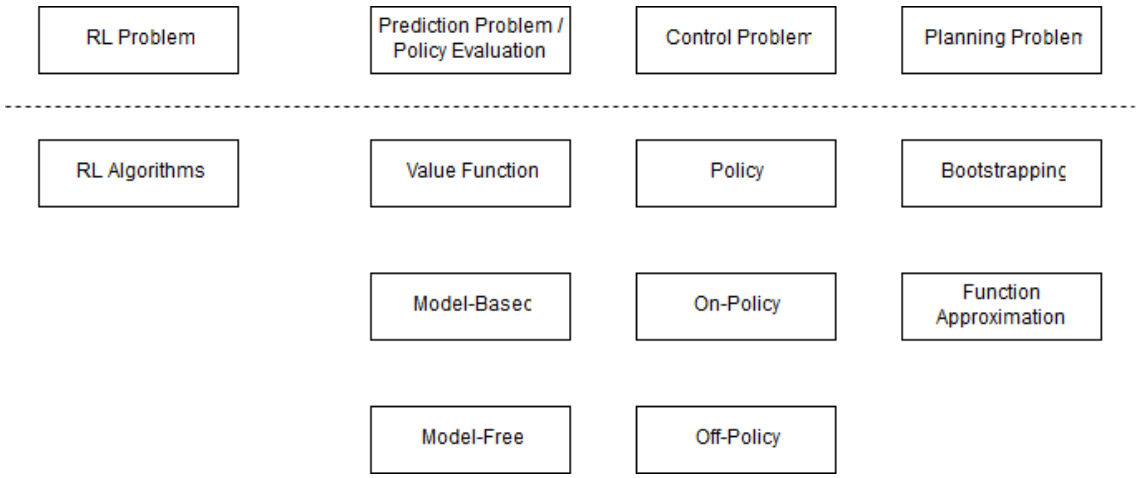

Figure 2. RL problem and algotirthm types

The RL problem (Fig. 1) defined by the 5-tuple (S, A, P, R, γ) can be formulated as a Markov Decision Process (MDP) or a Partially Observable Markov Decision Process (POMDP) and can be further divided into three problems: prediction or policy evaluation problem, control problem, and planning problem. Prediction or policy evaluation problem involves computing the state or the action value for a policy. Control problem involves finding the optimal policy. Planning problem involves construction of a value function or a policy with a model. In case a model is available, dynamic programming methods (policy evaluation, policy or value iteration to find optimal policy) are used. In case a model is not available (a so-called model free environment) the RL methods such as Temporal Difference Learning, Q-Learning, and Actor-Critic are used (Fig. 2).

The RL algorithms design to solve the RL problems can be some value function and/or policy based, model-free or model-based, on-policy or off-policy based, some function approximation or not. Usually the RL methods are used with tabular spaces of state and action that tend to be small. To generalize to much larger or continuous spaces, the concept of function approximation is used. A function approximation is an attempt to construct an approximate of an entire function using examples of the function and is a more general method.

OpenAI Gym [2] provides a toolkit for reinforcement learning research. It consists of a collection of environments that are modelled as POMDPs. At each step the RL agent takes an action, receives an observation and a reward from the environment (Fig. 3).

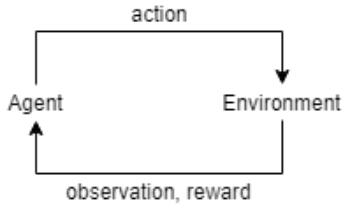

Figure 3. Agent-environment interaction

The agent continues to interact with the environment until it reaches the terminal state and then restarts. The goal of this episodic reinforcement learning is to maximize the expectation of the total reward for each episode. The OpenAI Gym's Atari environment offers the Atari 2600 games as testbeds for developing AI agents. The project uses the Breakout game environment for evaluating the RL algorithms. The Breakout game environment is created using the Gym's make module, and initialized to get the first observation of the environment, which consists of a 210 x 160 pixels RGB image that represents the state of the environment. The created environment provides a step function that allows for an action, taken from game's action space, to be inputted and returns the new observation, a float reward, a Boolean flag, and a dictionary info. The new observation represents the new state of environment after performing the action and corresponding reward received for that action. The Boolean flag denoted the terminal state of the environment thus notifying if an environment restart is required. The info dictionary provides additional information helpful for debugging.

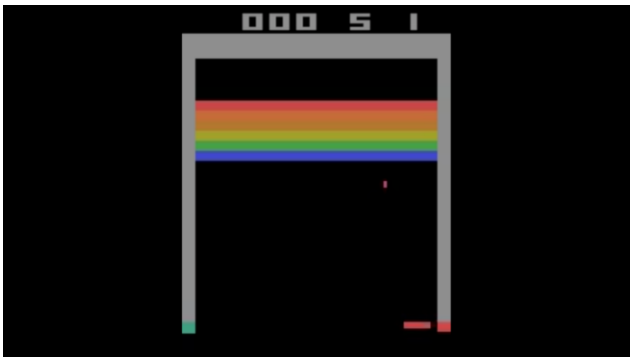

Figure 4. RGB image representing state of environment

The RBG image is cropped, resized into 84 x 84 and converted to grayscale as a part of the preprocessing step (see Fig. 5). This reduces the number of computations required by the learning network and speed up the training process.

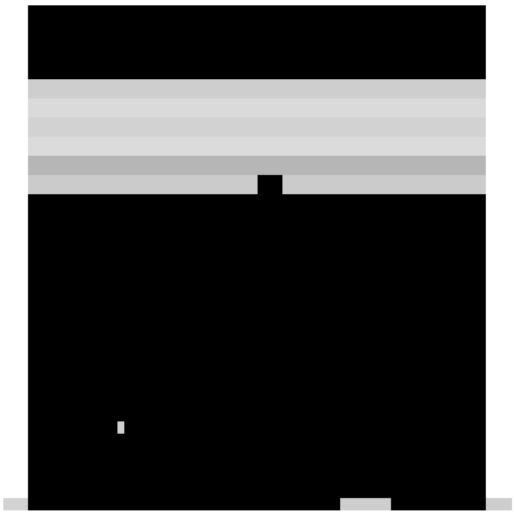
Figure 5. Preprocessed observation

Deep Reinforcement Learning methods [3,4,5,6,7] use deep neural networks as function approximator to approximate either the value function $\hat{v}(s;\theta)$ / $\hat{q}(s,a;\theta)$, or the policy $\pi(a|s;\theta)$, or the model (state transition function and reward function). The parameter θ represents the weights of the deep neural network and stochastic gradient descent, optimization technique, is used to update the weight parameters during the learning steps. The project explores two deep reinforcement learning approaches, Deep Q-Learning and Asynchronous Advantage actor-critic, proposed by the DeepMind team.

For each of the two methods, we follow the steps below:

1. Environment loading:
    a. Selecting the environment where the agent will learn. The environment presents the agent with observations, performs actions specified by the agent and returns the rewards.
    b. Setting the parameters of the environment as required for the learning process.
    c. Preprocessing the observations to create the state.
2. Network implementation:
    a. Implementing the primary network – to choose an action
    b. Implementing the target network – to generate Q-values for that action
    c. Creating helper functions to:
        i. Implement experience replay to help the network train from experience.
        ii. Update the parameters of target network with primary network.
3. Network training:
    a. Setting the training parameters. The parameters include – number of experiences to use for each training, frequency of training step, discount factor on the target Q-values, path to save the model, and many others.
    b. Training the network. With the set parameters the networks are trained, training metrics are logged. throughout the process to help check and debug, and periodically the model is saved.
4. Visualizing results:
    a. Plotting graphs for the various stats to understand the network performance.
    b. Viewing the TensorBoard readings to check and debug the model.

Q-Learning is an off-policy control method used to find the optimal policy. It learns the action value function using the update rule, al;so called the Bellman Equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a')\text{-}Q(s,a)]$$

Q-Learning refines the policy greedily with respect to the action values using the max operator. This learning is applied to smaller spaces where the value function or the policy is stored in tabular form. So, the action space A and state space S can be represented as a two-dimensional array (A*S) and Q-Learning employs dynamic programming methods to update the values of the array. But the learning does not generalize well to estimate value for unseen states.

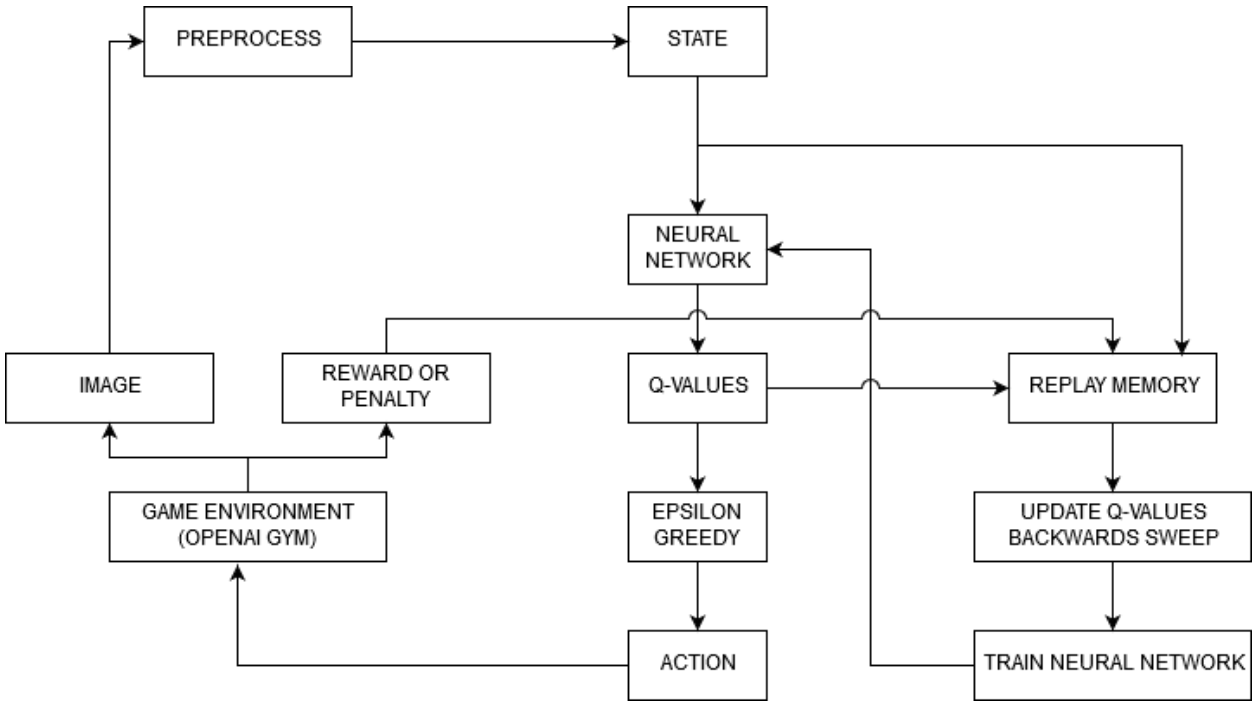

Figure 6. Steps for DQN

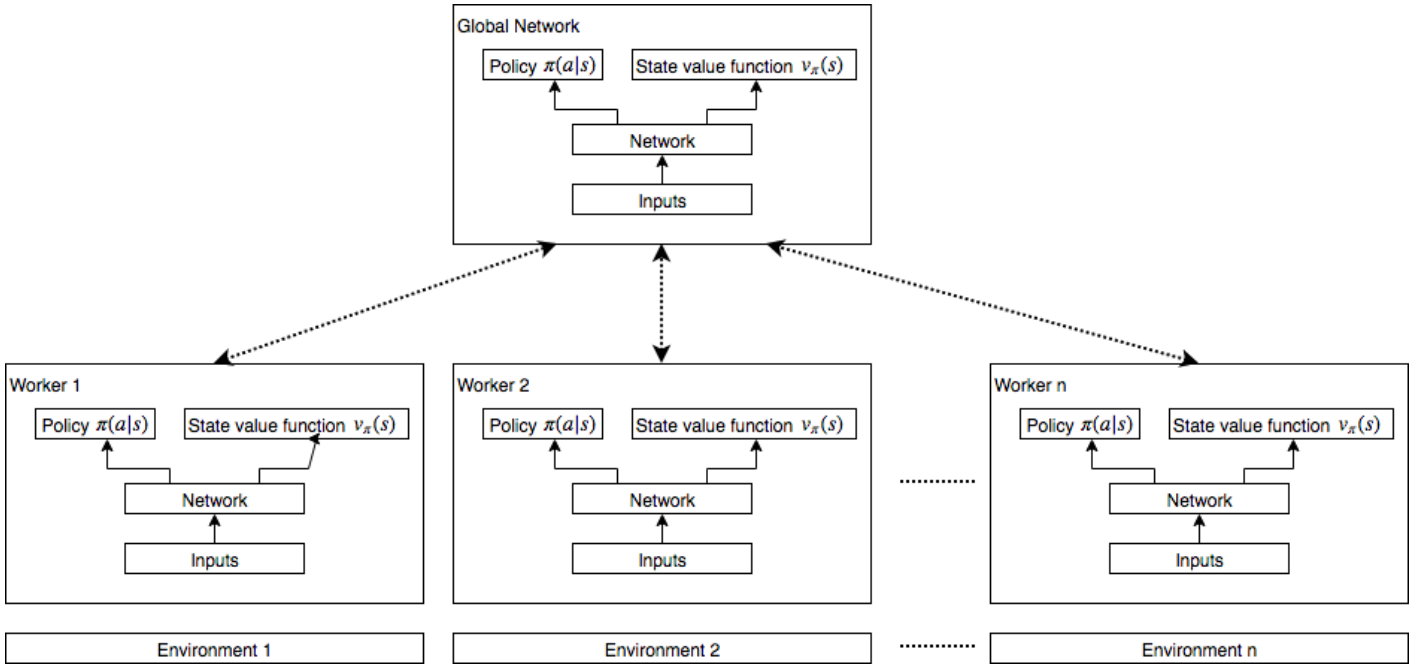

Figure 7. A3C algorithm model

To overcome this issue, Deep Q-Learning Network (DQN) replaces the tabular array with a deep neural network and leverages this network to estimate the action value function. DQN uses a Convolution Neural Network (CNN) as the function approximator of the raw pixels of the Atari 2600 breakout game. DQN stabilizes the training of the value function approximation with CNN using a replay memory and target network. By using only, the raw pixels of the game state there is no need for feature engineering and minimal domain knowledge is required. CNN function approximator, also called the Q-Network uses three convolution layers and two fully connected layer as a part of it network architecture.

III. PROJECT DESCRIPTION

The project currently uses the Atari 2600 game called Breakout, in which the agent will learning control policies using deep reinforcement learning approaches to achieve a

high score. A few roadblocks have been encountered during the various steps of the project. Their mitigation has been listed in Table 1. The main steps of DQN are shown in Fig. 6.

Recall that a policy is the mapping between state and action and value function measures the goodness of a state or state-action pair. DQN learns an action value function and defines the policy from that value function. The Asynchronous Advantage Actor-Critic (A3C) algorithm learns both the policy and value function. The actor representing the policy and critic representing the value function will be implemented as fully connected layers on top of the network. The learning will use the value function (the critic) to update the policy (the actor) more intelligently. The advantage estimate gives a measure of how much better the actions taken in a state turned out to be than expected. Multiple worker agents, each with their own set of network parameters, explore their own copy of the environment to learn the experience and update the global network with the diverse experience (see Fig. 7). This is much more intuitive than the discounted rewards which plainly measure if an action was good or bad.

Table 1. Project roadblocks and mitigation

Roadblocks	**Mitigation**
Reinforcement Learning Environment • Environment domain knowledge and feature engineering	OpenAI Gym • Collection of environments (Partially observable Markov Decision Processes) POMDPs • Simple interface • Monitoring tools
Deep Reinforcement Learning Agents • Mathematical complexity • Design complexity • Implementation complexity	• Baseline on existing research • Early proof-of-concept
Design-Implement-Test-Extend cycle too long	Choose simpler environment Understand RL algorithms (WIP)
Custom environment	Understand existing environment wrappers (WIP)

IV. Research results and analysis

In both DQN and A3C approaches the agent successfully learned to play the Atari Breakout game well and was able to reach a high score of 79 with A3C and 44 with DQN (Fig. 8).

The A3C algorithm proved to be much better than DQN in terms of training time, stability and higher score. As seen in the rewards graph (Fig . 9 and 10) the DQN rewards are much noisier and dips overtime, while A3C rewards increases gradually. The A3C algorithm employs asynchronous methods that can run efficiently on a multi – core CPU with multiple workers and environments interacting concurrently whereas DQN is a single agent single environment that needs a powerful GPU to train faster and is much slower a CPU.

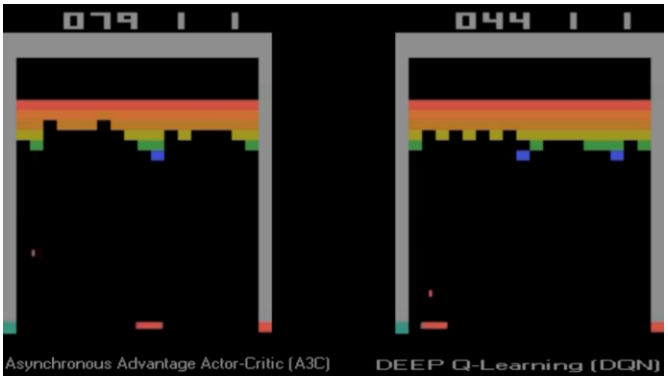

Figure 8. Breakout high scores for DQN and A3C

IV. Conclusion and Future Work

The trained models and checkpoints, source code, agent gameplay, and references to original authors has been placed in a GitHub repository [9].

The project explored and compared reinforcement learning approaches to train intelligent agents that can interact with an environment with automatic feature engineering thus requiring minimal domain knowledge. That can learn control policies for the given task to enable them to maximize the expected long-term rewards for that task in the environment thus succeeding in the task.

References

[1] Britz, D. (2016). Learning Reinforcement Learning. Retrieved from http://www.wildml.com/2016/10/learning-reinforcement-learning/

[2] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. CoRR, abs/1606.01540.

[3] Gron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Inc.

[4] Juliani, A. (2016). Simple Reinforcement Learning with Tensorflow Part 4: Deep Q-Networks and Beyond..

[5] Li, Y. (2017). Deep Reinforcement Learning: An Overview. CoRR, abs/1701.07274.

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. (2013). Playing Atari with Deep Reinforcement Learning. CoRR, abs/1312.5602.

[7] Mnih, V., Puigdomènech Badia, A., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. CoRR, abs/1602.01783.

[8] Pedersen, M. E. (n.d.). Reinforcement Learning (Q-Learning). Retrieved from TensorFlow Tutorial#16:https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/16_Reinforcement_Learning.ipynb

[9] GitHub, https://github.com/anoopjeerige/deep-learning.

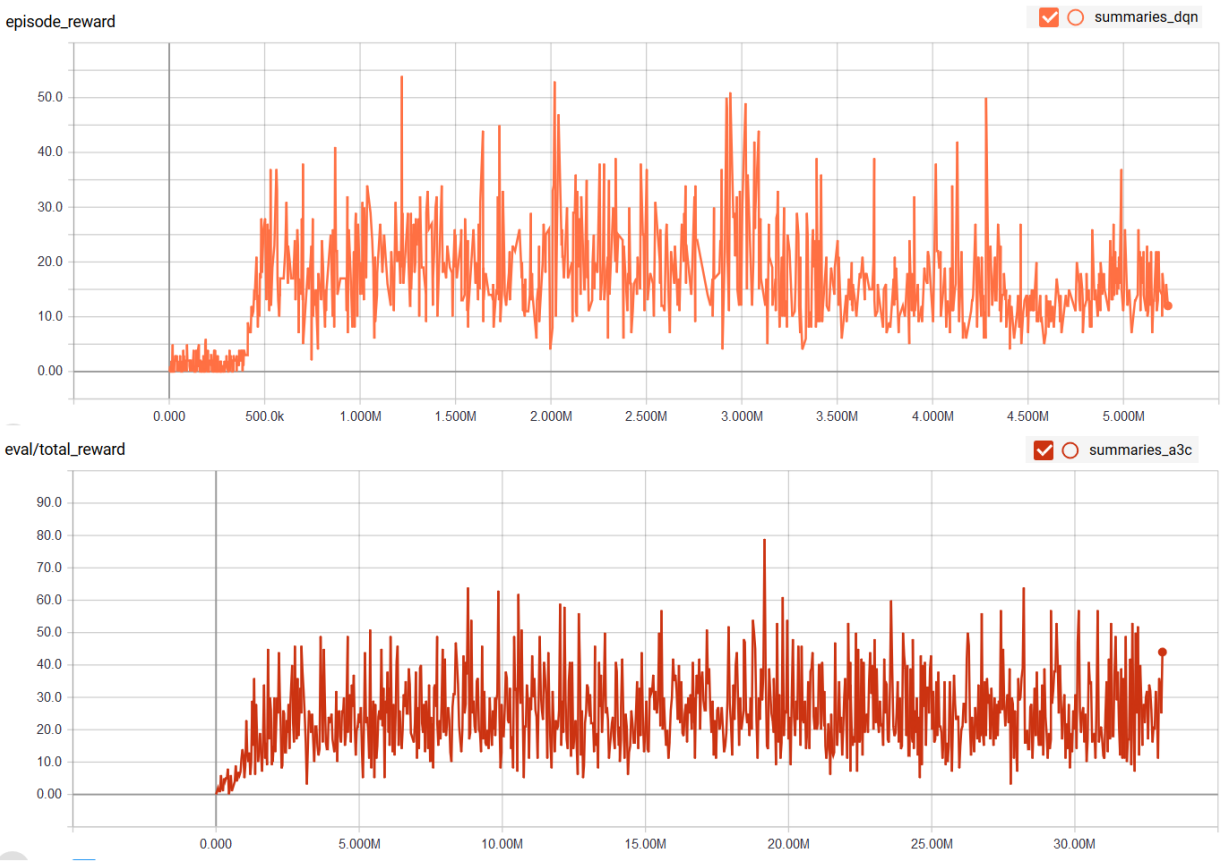

Figure 9 - Episode reward with steps

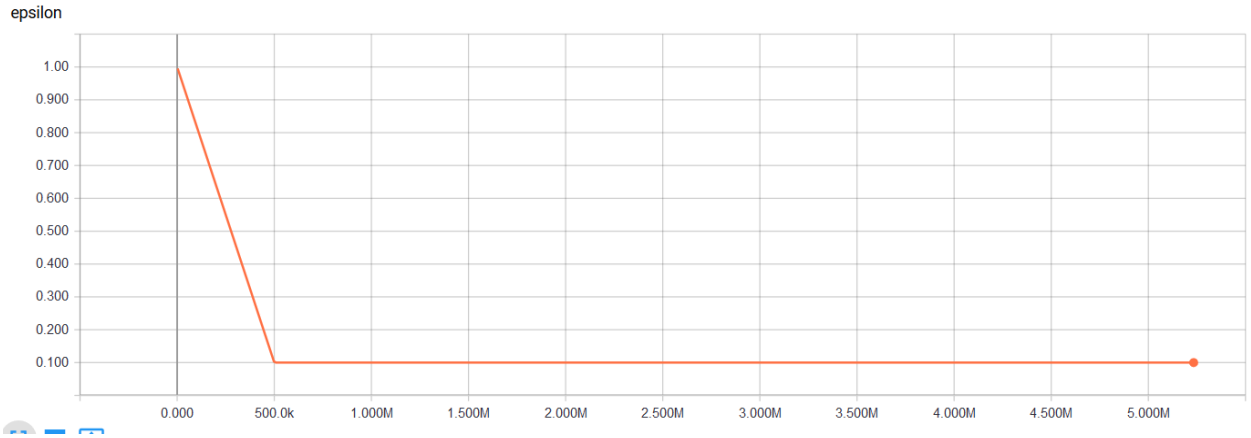

Figure 10 - Epsilon for DQN