Lunar Lander Agent:

Shailesh Tappe: stappe3 **Git Hash: ad14b27a4e6fac2a49c367ce532b6ac5e554872f**

Objective: The objective of this project report is to explore and understand reinforcement learning (RL) model free learning algorithm (Q Learning) and nonlinear function approximation using artificial neural network (ANN). The experiment is to design model for Lunar Lander game provided by OpenAI Gym environment. The input for model is series of pixel images generated by Lunar Lander environment, which then used in double DQN (Deep Q Network) to produce action from action space of the game.

Introduction: OpenAI Gym's Lunar Lander is 2D environment with small rocket initiated at top middle position with continuous state space with state vector as (x, y, vx, vy, θ, vθ, left-leg, right-leg) and discrete action space comprised of {do nothing, fire left orientation engine, fire main engine, fire right orientation engine} . An episode finishes if lander crashes or comes to rest, receiving additional -100 or +100 points respectively. Each leg ground contact is worth +10 points and firing main engine incurs penalty of -0.3 points for each occurrence. The goal of experiment is to land rocket within designated location and achieve score of 200 points or higher for average over 100 connective runs.

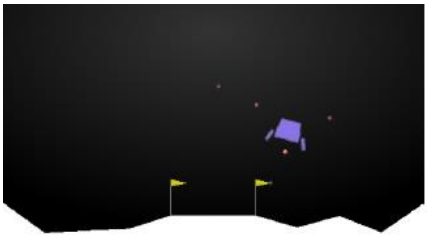


Figure 1: Illustration of Lunar Lander environment

Approach: The Lunar Lander can be formalized as Markov decision process (MDP), in which for any state vector S, an action a is defined. Agent can be solved using model free RL algorithm (Q-Learning), which does not require definitive MDP and policies are generated based on Q function values for any state action pair Q(s,a) and reward is predicted by Q function $Q(s,a): S \times A \rightarrow \mathbb{R}$. It explores and learn optimal policy $\pi(s)$ from changing environment by maximizing rewards $\pi(s) = \text{argmax}_a\, Q(s,a)$. This learned policy directs the agent what action to take at each state.

Since the game has large continuous state space, a naturally suitable solution would be using function approximation to model the Q function Q(s,a) using artificial neural network (ANN). For this experiment Double deep Q-Network (DQN) is used using Keras and Tensorflow libraries for solving the game.

Double DQN is used in experiment over DQN to solve this environment. DQN is widely experimented and successfully generalized its learning to solve Q function, but the problem with DQN is that it tends to overestimate Q values. This is because max operator in Q learning equation uses same values for both selecting and evaluating an action, Due to this in many state action estimates, higher value action is preferred over optimal action. This leads in selecting suboptimal action instead of optimal action.

To solve this problem Double DQN is useful, as it uses two separate Q functions, each learning independently. One Q function is used to select an action and other to evaluate an action. This is done by updating and adjusting the target function of DQN $y_i^{DQN} = r + \gamma max_{a'} Q(s', a'; \theta')$ to $y_i^{DoubleDQN} = r + \gamma Q(s, argmax Q(s, a; \theta^-); \theta')$. In this equation Q function with weights θ' is used to select action of other Q function with weights θ^- to evaluate action.

Algorithm 1 Double Q-learning

Initialize Q^A, Q^B, s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
  else if UPDATE(B) **then**
    Define $b^* = \arg\max_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end

Figure 2: Double Q-Learning algorithm by H. van Hasselt

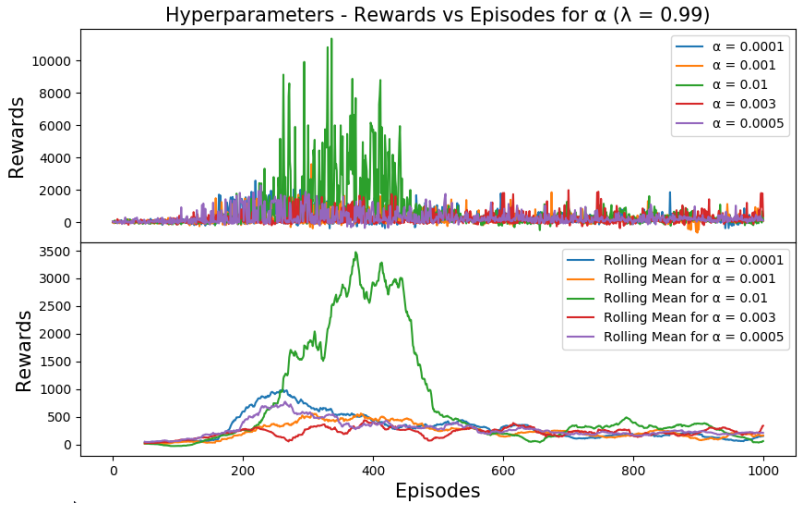
Experiments: Experiments were implemented using python 3, and with Keras and Tensorflow based feed-forward neural network and double DQN algorithm. Dense network of 2 hidden layers of 128 node each were used with rectified linear (ReLU) activation function. Lost function mean squared error is used to gauge performance of neural network.

$$J = \frac{1}{2}(y - \hat{y})^2$$

J → Loss Function y → Actual Value $\hat{y}$ → Predicted Value

Different hypermeters (learning rate(α) and discount factor(λ)) on environment with 1000 episodes each were analyzed to tune parameter for environments. The training sessions were performed for 2000 episodes until rolling average of rewards reached above 200. This trained model is then used to 100 trained episodes until rolling average of rewards reached above 200.

1. **Hyperparameter Analysis:** Learning rate and discount factor with different values were experimented with 1000 episodes to determine optimal parameter for running an agent. During exploration it is observed that if learning rate α is higher then network tends to constantly overshoot the minimum and degrades or prevents in converging. Using smaller learning rate of 0.0001 shows better result with discount factor of 0.99.



Analyzing discount factor for a smaller value, an agent was impaired of learning rewards. For lesser discount i.e. 0.9 agent couldn't solve temporal credit assignment for more than 10 steps

ahead (1/(1- λ) = 1/(1-.9) = 10), but for higher learning rate closer to 1 i.e. for .99 ((1/(1- λ) = 1/(1-.99) = 100)) allowing agent to credit more rewards for landing correctly.

2. **Experiment Results for training:** This experiment is carried out with learning rate = 0.0001 and discount factor = 0.99 for 2000 episodes and tested model on 100 trained episodes
From figure 3 it is clear that agent is very erratic for about first 100 episodes, but after exploring it becomes less intermittent because it starts exploring on what it has learned.

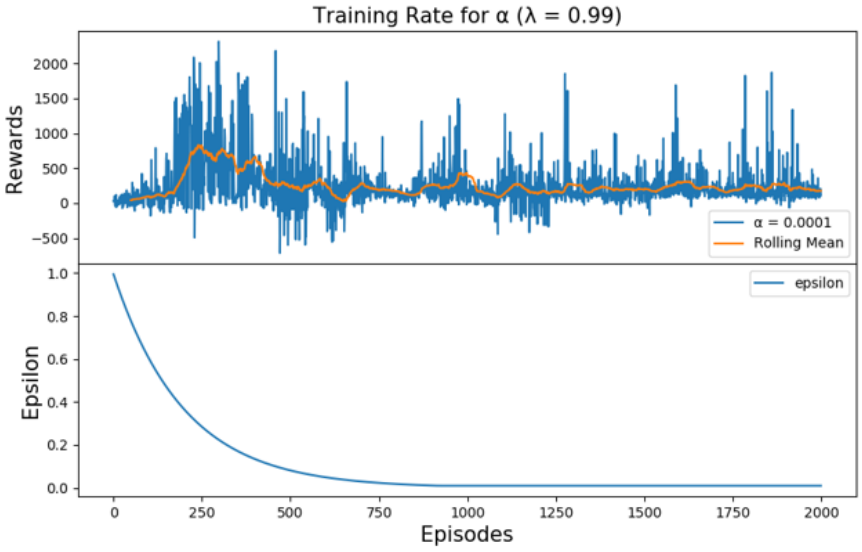


Figure 3

3. **Experiment Results for testing Trained Model:** The trained model is then tested with 100 episodes and it is observed that average reward for test episodes is above 200.

4. **Issues and Resolution:** During experiment, extremely large continuous hyperparameter space such as discount factor, learning rate, network node size, memory deck size and batch size, epsilon was producing impaired results. Most of parameters except discount factor and learning rate was adjusted with intuition and experimentation.

Conclusion: Overall project experience was very unique and interesting and outcome of experiment is to learn Q-Learning model with function approximation using artificial neural network (ANN). Model is able

to transfer over to another deterministic environment for RL problem solving. Preliminary results are promising and can significantly improve with more exploration on large space of hyperparameters. In future I intent to explore these hyperparameters and also explore and experiment Dueling DQN algorithm which was not feasible due to time and space constraint during this experimentation.

Double DQN appears to be more robust for challenging evolution, suggesting that appropriate generalization occurs and explored solutions do not exploit the deterministic environment. This is more useful as it suggests progress towards finding generalize solutions rather than deterministic sequence of steps that would be less robust.

References:

Richard Sutton Book: Reinforcement Learning - Chapter 9 - On-policy Prediction with Approximation

Dr Charles Isabel and Dr Michael Littman: Lesson 9: Generalization

Hado van Hasselt, Arthur Guez, and David Silver: Deep Reinforcement Learning with Double Q-Learning

Hado van Hasselt: Double Q-Learning

TA’s during office hour and piazza forum.