Project 3 Report

(Yogesh Luthra. ID: yluthra3)

(Link to Video)

Summary: Lunar Lander is a continuous space, discrete action environment. Since Neural Nets are considered universal function approximators, with a non-linear activation function, they are used as "agent's brain". Rectified Linear Units (ReLU) as activation units were used due to non-diminishing gradient benefits and low evaluation time cost. Double DQN [1] algorithm was implemented with an online and a target network. Reason for this choice and selection of parameters is explained in subsequent sections. The implemented algorithm was able to successfully solve the domain in short number of iterations (100-episode average reward > 200).
(Some evaluations on OpenAI: D-DQN 1, D-DQN 2, DQN)

Architecture: State vector = I(Observation Vector) where I: identity transform; Observation Vector $\in R^8$
3 layer fully connection Multi-layer Perceptron Neural Network with 2 hidden layers activated by ReLU units were used and last layer was a simple linear layer. Following weights and biases were used.
1st layer Weights.shape=(8, 50) 2nd layer Weights.shape=(50,50) 3rd layer Weights.shape=(50, 4)
Both, online network and target networks, used above structure. Target network was a simple periodically sampled replica of online network, with period being C iterations. Different values of C were tried as explained below.
Higher number of layers and units per layer were tried but above values seemed optimal.

Experiments run and parameter settings:

Gamma: was set to 0.99 in all experiments as lower value (like 0.9) caused to either converge too slowly or stuck in a local minima (Lander never touched the ground). Lower gamma is assumed to limit the horizon or in other words, seemed to restrict how far an effect of taking some action the agent could see. With gamma=0.9, agent approximately could see effect of taking current action upto 10 steps into future. Whereas, with gamma=0.99, agent could see effect of current action upto about 100 steps into future. However too high of a value, like 0.999, caused too long for agent to converge, as it possibly caused too much noise in Q-value evaluations.

State Vector Transformation: Sequence of Observation Vectors were rolled into a State Vector but didn't seem to help much for this domain. Possibly because Observation Vector contained enough information of sequence that caused current observation (like coordinates, x/y velocity, angular velocity, left/right leg to ground information).
Each dimension in State Vector was centered around zero mean and normalized by standard deviation. Mean and Standard deviations for each dimension were extracted from initial 3000 uniform random iterations in the environment.

Algorithms: Deep Q-Learning (DQN) [2] and Double Deep Q-Learning(D-DQN) [1] were implemented, both with an online network and a target network. D-DQN 100-episode average reward was seen to be much smoother than DQN; and D-DQN converged slightly earlier than DQN (result comparison in video). Hence D-DQN was chosen for all subsequent experiments. Replay memory was an important consideration in these algorithms. Neural networks were trained by sampling uniformly at random from this Replay memory, assuming this breaks correlation between samples. However an important general requirement for machine learning systems, that samples should be Independent and Identically Distributed (I.I.D) may not have been satisfied due to nature of agent's experiences.

Replay Memory Size: Initially the size was set too small (200000) which probably caused memory to be filled with many similar experiences. Some episodes were observed to last 1000 iterations. Only 200 such similar episodes broke assumption of samples being un-correlated. This clearly caused model to be biased to certain experiences. Some explanation of this effect is in video summary. When memory size was increased to 1000,000 related issue was resolved.

Minibatch Size: Through experimentation, minibatch size=200 was found to be best for solving the domain in shortest possible number of episodes. However lower numbers like 40, 80, 100 were also tried, but were seen to take more time to solve.

Regularization: Network weights were tried to be regularized by l2_loss (in TensorFlow). But didn't seem to help much.

Exploration Strategies: e-greedy, epsilon-first, epsilon-decay (linear) and Gibbs sampling. For all, best was found to be epsilon-decay. Various shapes of epsilon-decay were tried by varying initial exploration probability and ending to 0.0 in 600 episodes. Final choice was epsilon-decay from 0.7 to 0.0 in 600 episodes. Exploration shape seemed to have a relation with learning of neural network, as will be discussed later.

Optimizer for Neural Network: RMSProp, Adam, SGD optimizers were tried of which best seemed to be RMSProp.

Learning Rate of Neural Network: Learning rates tried were 2.5e-5, 2.5e-4 and 2.5e-3. Best seemed to be 2.5e-4. Interestingly this is about the same value used in [1], [2] with same optimizer, that was discovered best for this domain as well.

Results

Due to space constraints, only results of experiments with Learning Rate and Exploration Shape will be discussed. But first, following graph shows results of one of the successful runs which solved the domain. Learning Rate=2.5e-4 for RMSProp optimizer, Epsilon-decay (linear): 0.7 to 0.0 in 600 episodes.

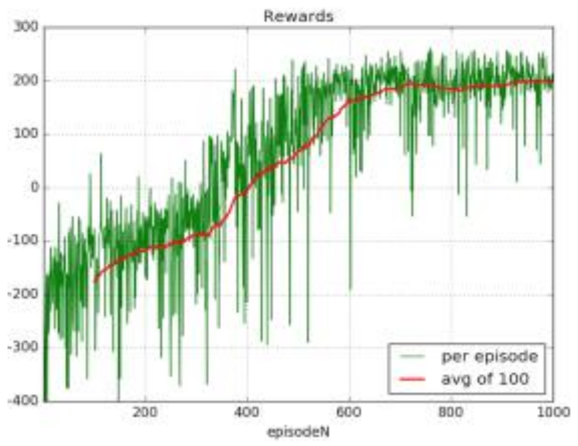

Fig 1. Awards per episode (Green), rolling mean (Red) of 100 episodes.

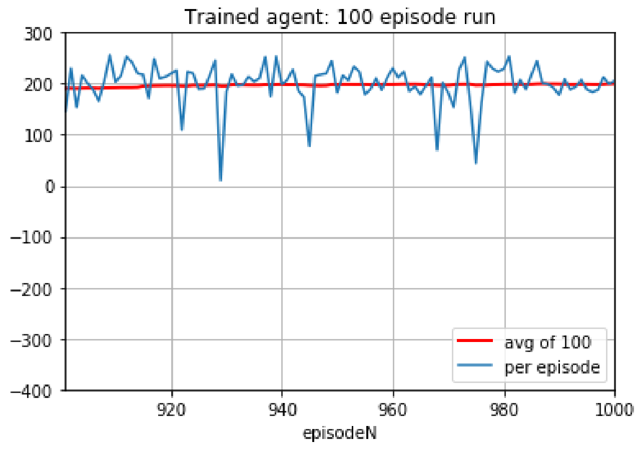

Fig 2 Rolling mean of awards over 100 episodes of trained agent

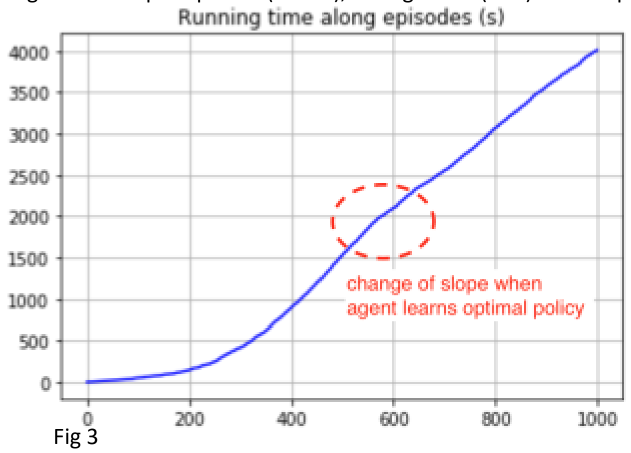

Fig 3

Fig 1 shows rewards per episode as well as rolling mean over 100 episodes in Red. Exploration ends at episode 600. Fig 2 shows rolling mean over 100 episodes of the trained agent (no exploration).

Fig 3 shows an interesting aspect about agent's learning behavior. Between episodes 200-550, time per episode was increasing non-linearly (most of episodes 1000 iteration long). However once agent learns optimal policy close enough, episodes afterwards take almost similar time per episode exhibited by linear line after 550 episodes.

Figure 4 shows the reason for choice of hyper params used to generate above plot. Two important statistics were extracted from each combination of hyper-params. Rolling mean and rolling standard deviation from past 100-episodes. Rolling mean tells us average improvement along with episodes, whereas rolling stddev tells how stable current policy is. Low stddev may just mean the policy is locally optimum, however taking statistics over wider window may show more deviations, if any.

It was intended to find hyper-params such that highest gains can be achieved in least number of episodes. This was important keeping in mind long run times in this domain. With that in mind, maximum number of episodes were set to 1000 and those params which gave best results were chosen.

1st column, learning rate (alpha)=2.5e-5 is clearly too slow of the network to learn, as evidenced by low value of rolling mean and non-monotonic variance trend. 3rd column, learning rate=2.5e-3 is also clearly not appropriate as, although rolling mean does show a trend to approach average reward of 200, but stddev trend shows too much uncertainity in policy, as it is highly non-monotonic.

From trends in Fig 4, few interesting observations can be made. In 2nd column, e=0.7 (green lines) corresponds to less exploration. Green lines have higher rolling mean, lower and almost monotonically decreasing stddev. However e=1.0 (blue line), in which the agent supposedly explored more, has much variety of experience in its replay memory. But e=1.0 case shows both lower lower rolling mean towards the end of episodes, as well as, non-monotonic stddev of episodic rewards. But e=0.4 is clearly worst among three. **This could suggest that for a certain learning rate, there is an optimum exploration magnitude for the agent to learn fastest.**

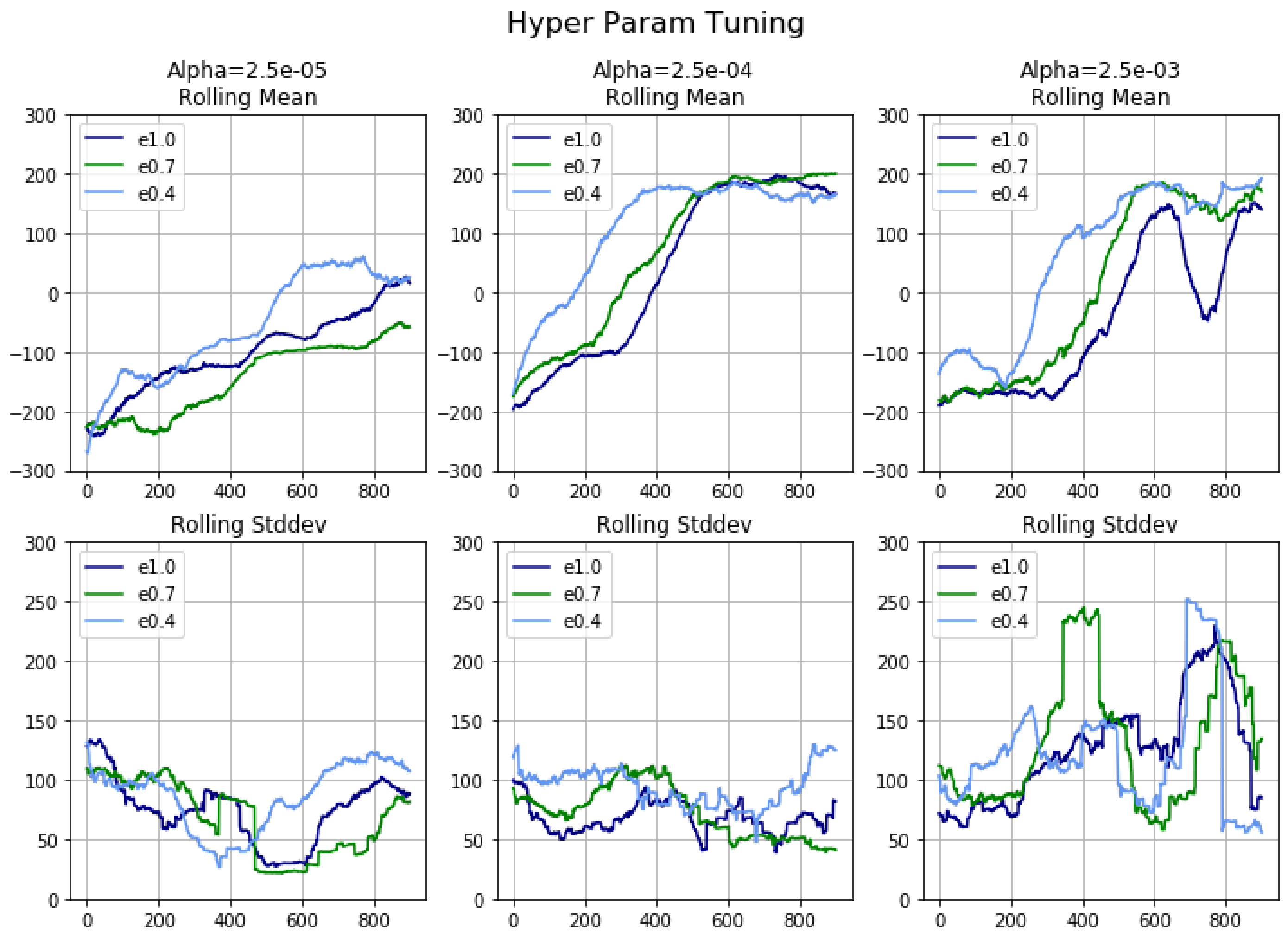

Fig 4. Rolling mean and stddev of episode rewards.
‘**Alpha’**: Learning Rate; ‘**e**’: epsilon value at start of linear decay.

References:
[1] : https://arxiv.org/pdf/1509.06461.pdf
[2] : http://www.nature.com.prx.library.gatech.edu/nature/journal/v518/n7540/pdf/nature14236.pdf