ECE 448 MP4 Report

Ashish Kumar Sathish(asathis2) 1.1

Sooraj Kumar(sskumar5): 2.2, EC

Xi Chen(xichen30) 2.1

1.1

1.

A high discounting factor γ value from 0.75-0.85 worked best for the TD agent. This worked because we need to increase the likelihood of higher future rewards by placing a higher importance on it for an optimal policy. Too high of a γ value 0.85-0.95 in the SARSA agent resulted converged to policies with ~7-8 as the score, I got the best score ~10 with 0.5. This is probably because SARSA uses a near optimal policy and requires it's current options to be weighed more greatly, it needs to be more opportunistic.

A learning rate α-constant value from 40-50 worked for both best since we required a policy that did not override old information too greatly but also provides importance to the number of times we have seen a state.

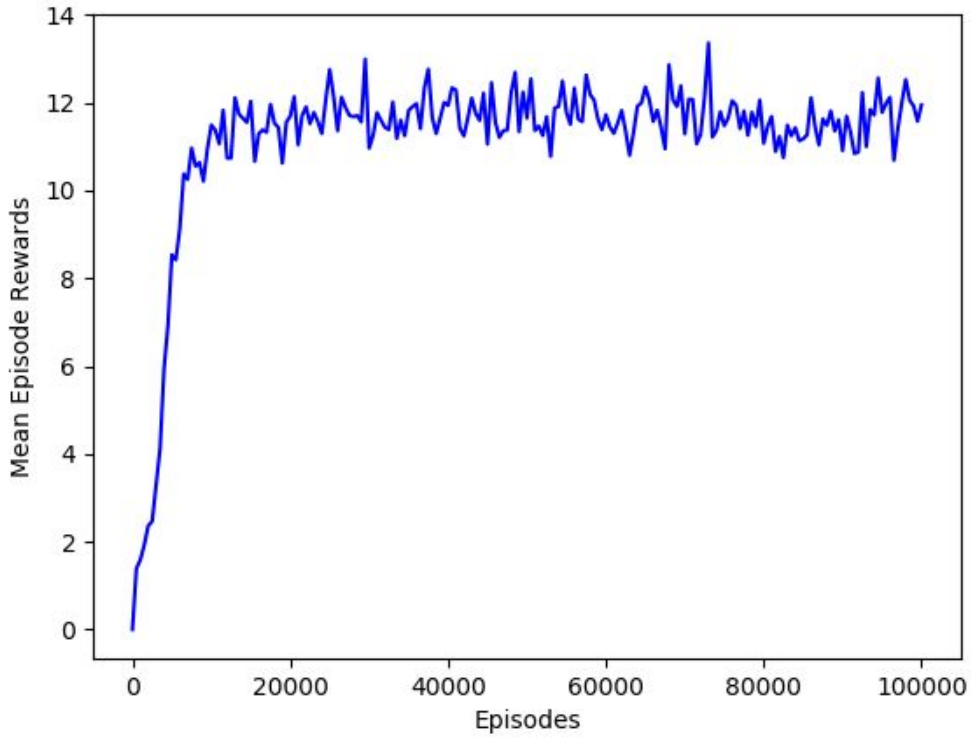
An exploration policy ε value of 0.03 worked best. Since we are increasing long term rewards in a fixed system, a low exploration policy would work best.

If a good policy is termed as reaching an average score of >9 and remaining around that value, then it usually took around 25,000 games for the TD learning agent and 45,000 games for the SARSA learning agent. Training time for a SARSA agent was longer than that of a TD-learning agent.

2.

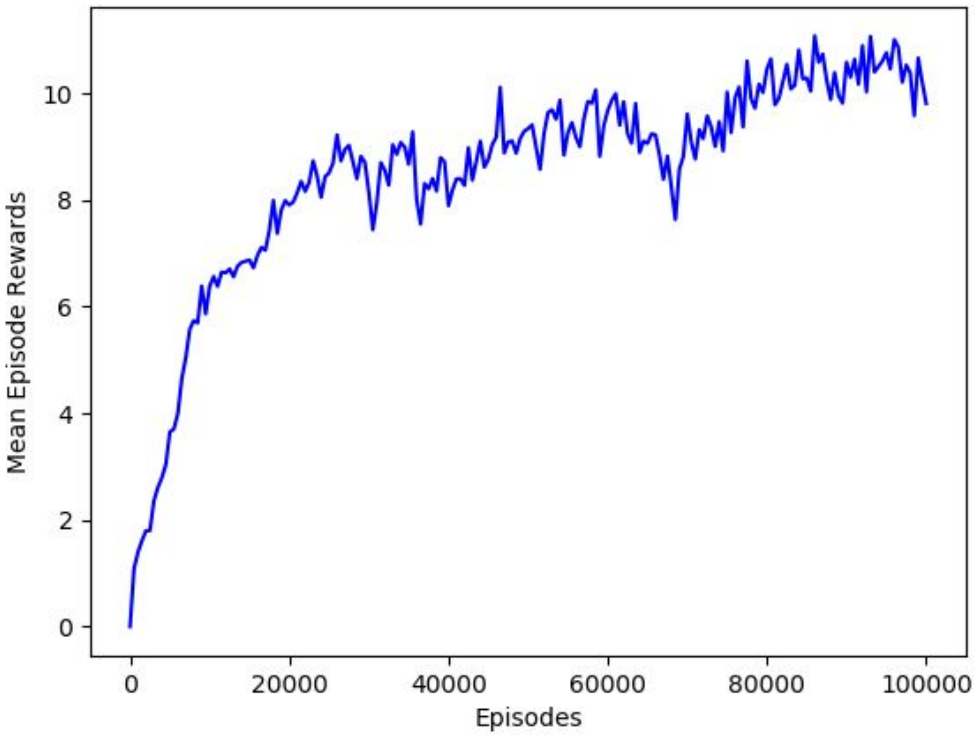
The best result I got from TD learning agent 200 test games average score of 12.32. I trained the TD table on 100,000 games.

The best result I got from SARSA learning agent 200 test games was an average score of 10.475. I trained the Q table on 100,000 games.



3.

TD-Learning (α-constant=40, γ =0.85, ε =0.03)



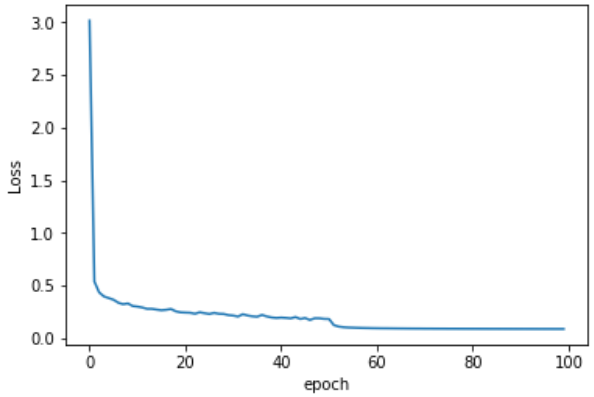
SARSA-Learning (α-constant=40, γ =0.5, ε =0.03)

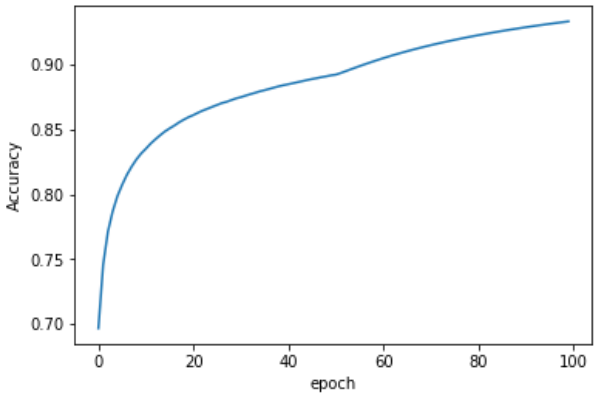
From the plots we observe that the TD learning agent converged quicker than the SARSA agent. The TD plot has a sharper growth. However it seems like the SARSA agent tends to avoid too many fluctuations by not choosing dangerous policies, it is more conservative. The TD agent would take the risk of a large negative reward as long as there is an optimal policy close by. The SARSA agent would be better to display rewards gained *while* learning.

2.1 behavior cloning and deep learning

1. Advantage of a deep network over Q-table: the state space is very large in Pong game (the ball can be in any position within the arena and can have different velocity in different direction, and the paddle position is varying as well), it will be a huge task for the Q learning algorithm to optimize the decision. Deep learning is for inferring complex functions and thus is more suitable for such situation.
2. Implementation:

A. Neural network architecture: We used a four-layer neural network, with 256 units per hidden layer. The input layer has 5 units and the output layer has 3 units. We used ReLu as our activation function.
B. Initialization: weights are initialized randomly in range [-0.5, 0.5], bias terms are initialized to be zeros.
C. Learning rate decay function: we had a initial learning rate of 0.05. The learning rate remains constant for half of the training process. After finishing training half of the epoches, the learning rate will be decided by learning rate decay function rate = initial_rate/(round - 0.5 * n_epoch + 1)
D. Preprocessing: in order to achieve faster convergence, we normalized the input data before we start training. In testing the input data needs to be normalized with mean and variance computed on the training set.
E. Loss and accuracy plot over number of epochs





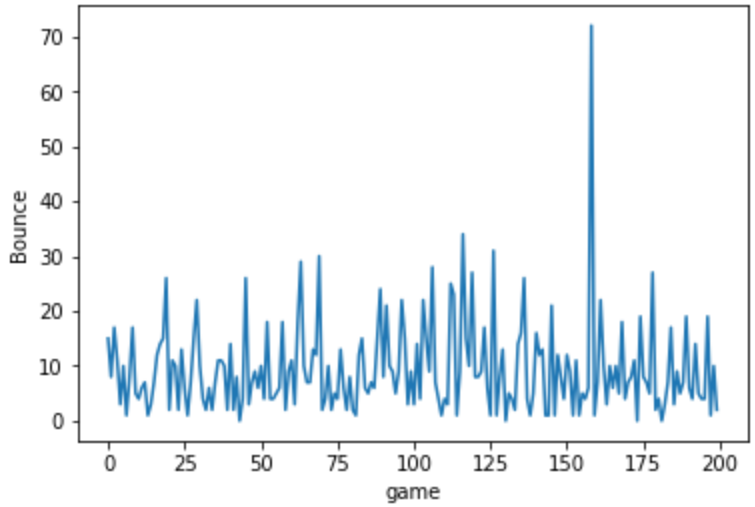
F. Confusion matrix:

[[0.92407618 0.02420216 0.05172166]
[0.03585574 0.92495092 0.03919334]
[0.032644 0.02284756 0.94450844]]

G. Performance:

We achieved 93.3114% accuracy after 100 epoches. Misclassification error: 6.6886%.

Number of bounce over 200 games



Average bounce: 9.415

2.2 **Behavior Cloning Using Advantages Estimation**

The implementation for this section mostly builds off of the implementation of the previous part. However, there are a few major changes now that we are dealing with advantage values rather than classes. To change between predicting classes to predicting real numbers per move, we made the following changes:

1. We changed the loss function from the cross entropy between our predicted Q(s,a) values and the targets to the mean square loss between our predicted advantage values A(s,a) and the targeted advantage values.
2. We calculated accuracy by taking the argmax of the A values for each state and the argmax for the targeted A value and compared the results to test for accuracy.

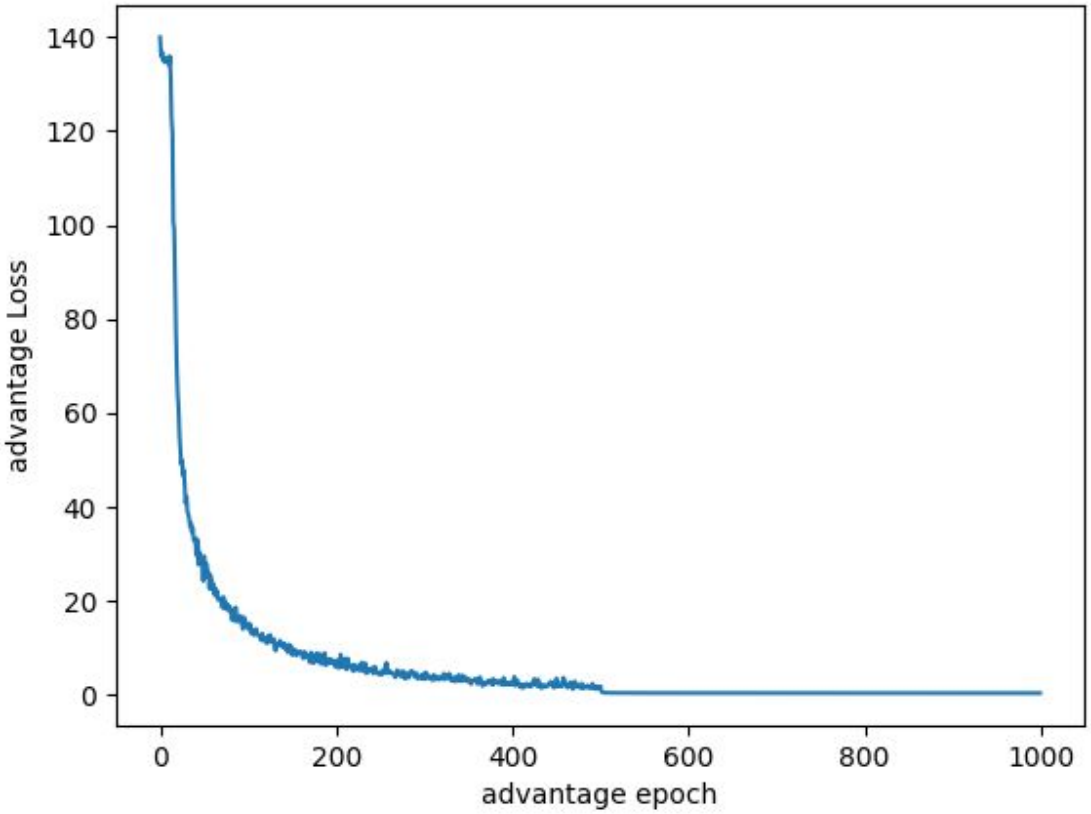
The choice for the best hyperparameters was a learning rate of 0.5, 1000 epochs and a weight scale of 0.01

Hyperparameters vs. Accuracy:

Learning Rate	# of epochs	Weight Scale	Accuracy	Bounces
0.05	100	0.01	66.209	3.82

0.1	500	0.01	88.6538	9.785
0.5	1000	0.01	94.24157	10.74
0.1	300	0.01	0.854491	9.14
0.5	1000	0.1	0.4007394	-1

Loss Curve: (Learning Rate = 0.5, Epochs = 1000)



Average number of bounces: 10.74

Part 2: Extra Credit

We implemented Batch Normalization for the extra credit section of part 2. We received 8.97 bounces with an accuracy of 83.7706