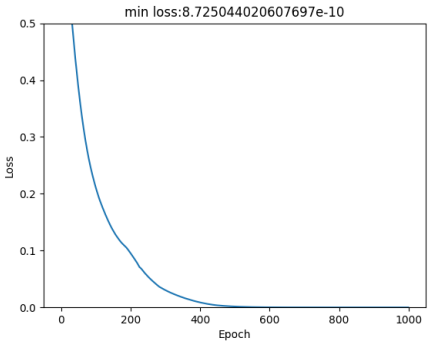
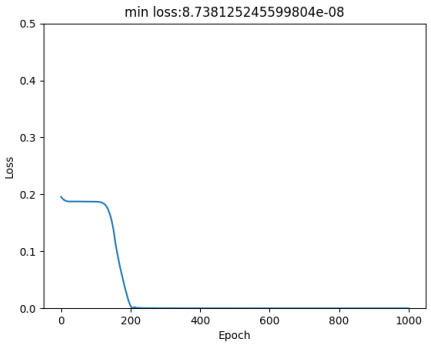
I a.

1. 

2. Model converges at epoch 400 as it flattens out in subsequent epochs.

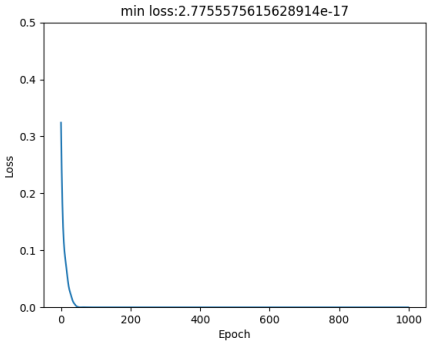
I b.

1. 

2. Increasing the number of layers accelerates converges by almost 200 epochs.

3. Yes, we do observe a phase (between 0 to roughly 150 epochs) where the model doesn't improve and sort of plateaus in terms of its loss. This could be because of the phenomena called vanishing gradients as the number of hidden layers or the layers through which our loss back-propagates through increases. It takes some time for our optimizer to find the optimal path and is stuck since the gradient values aren't that significant. Once they accumulate, they're enough to drive the model out of the local minima.

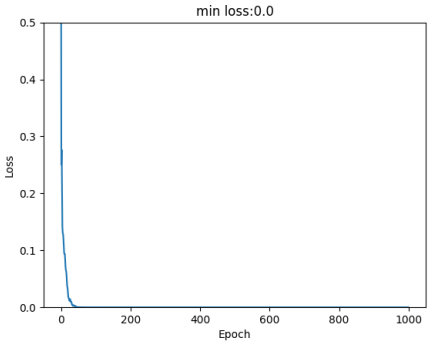
I c.

1. 

2. Increasing the number of hidden dimensions does accelerate convergence to below 50 epochs.

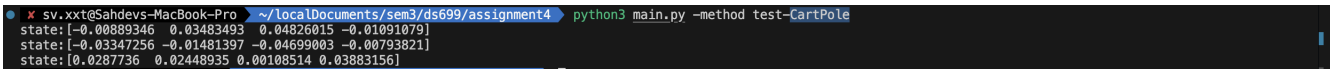
P.T.O

I d.

1. 

2. Increasing the learning rate does accelerate convergence to below 50 epochs again because our optimizer takes larger steps towards the global minimum.

II a.

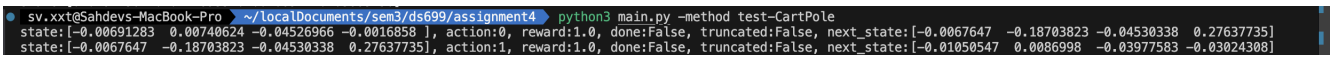
1. 

2. It is a random initial state.

3. Position : -0.00889346, Velocity: 0.03483493, Pole Angle: 0.04826015, Pole Angular Velocity: -0.01091079.

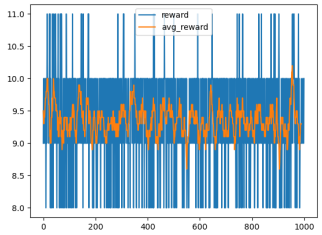
4. We should push the cart to the right as the pole is leaning towards the right side (+0.04826015). By moving the cart along the right side, we move the pivot point under the pole's center of mass, ensuring that the pole stays vertical.

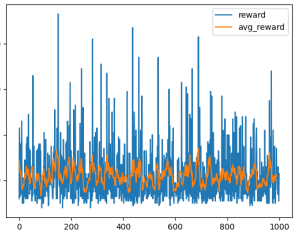
II b.

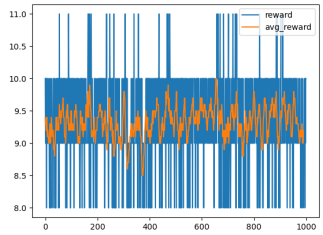
1. 

2. In the CartPole simulation, equal and opposite discrete forces fail to restore the initial state because the system's coupled, second order dynamics propagate both linear and angular momentum, and the fixed step integrator accrues truncation error. A rightward push instantaneously alters the cart's velocity and induces a nonzero pole angular velocity, and the subsequent leftward force applied to a moving mass with residual angular momentum cannot precisely reverse those changes.

II c.

1. Left : 

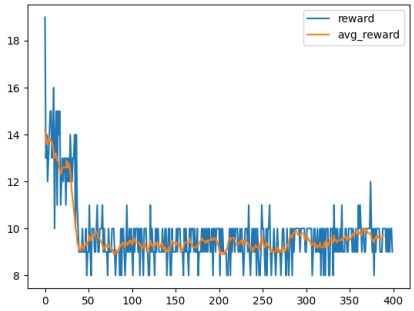
Random : 

Right : 

2. The random policy is better since the episodes are longer on average. There's a +1 reward for every action on any state, including reaching the terminal state. So, the y axis simply represents episode length. The longer the episode, the higher the policy balances the pole.

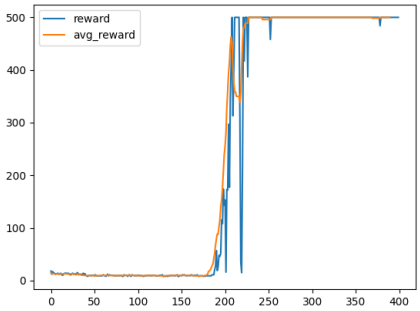
P.T.O

III a.

1. 

2. No.

III b.

1. 

2. Between Episodes 200 and 250.

3. The default environment reward is sparse and uninformative. It gives equal weightage to all the variations from the threshold values, making each state equally favorable and thus, poorly distinguishing between just upright and perfectly centered. However, that shouldn't be the case and is handled by the ratio reward, which is more dense and has shaped feedback. It also aligns the learning with values of position and angle, thus, enhancing the representation of the state. With ratio reward, small deviations receive a higher reward.

III c.

1. The frequency of target Q network's weight update affects the bias-variance tradeoff. If it's too small, then the target Q value is updated is more frequently, driving our model to chase this moving objective during backpropagation, which is observed as highs and lows from episode 130+ to 400. It has low bias but high variance. If the frequency is large enough, the learning is more stable as the target Q value is consistent for a few episodes so backpropagation enables thorough learning with a clean climb from 200 to 250 and is consistent. This has higher bias and lower variance.

III d.

1. Large buffer size leads to a more reliable convergence. The smaller one has highly correlated samples (violating i.i.d assumption as successive samples can be from the same episode), lacks diversity, is non-stationary as

P.T.O

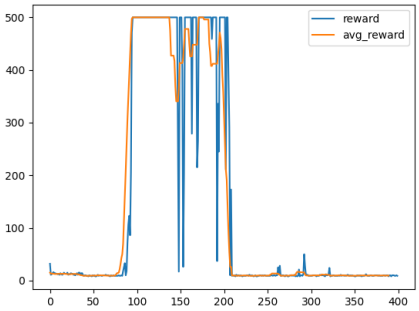
older experiences are immediately replaced, is low in variance and high in bias offering very few samples for generalizing the value function. The larger one tends to offer older and newer experiences in one batch which is more varied and thereby, smoothens out the learning signal.

III e.

1. As hidden dimensionality increases, the model tends to capture more non-linear patterns that existing between target network's q value and main network's q value. It also reduces the bias and makes the value approximation function adequately parameterized. These factors contribute towards a faster episodic convergence.

III f.

1. No, it isn't fair. With batch_size = 60, for 100 learning steps, we're passing 6000 transitions through our network. However for batch_size=10, we're passing only 10*100 = 1000 transitions through the network in 100 learning steps.
   For comparable performance, Steve should call the learn function 6 times per iteration.

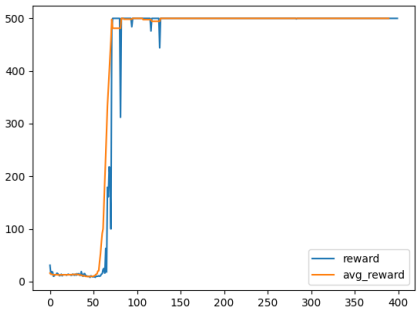
2. 

3. It converges earlier then diverges because these quick early updates are noisy and are less representative of the full experience (state).

III g.

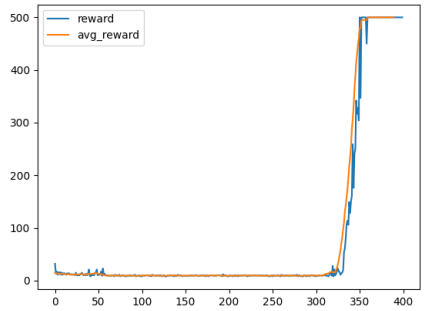
1. Increasing learning rate leads to large weight updates, which can in-turn lead to overshooting during back-propagation, thereby missing the minima and cause unstable learning. This leads to a poor estimation of the q value function. In the situation depicted here, I think it's a case of exploding gradients with fluctuations and divergence in training.

III h. Setting batch size back to 60 and comparing it with III b. 1.

1. 

P.T.O

2. The model converges earlier with soft update.

3. 

4. Yes, it slows down convergence because of slower propagation of new weights/knowledge from the main network and increased staleness of the weights of the target network. It increases bias since target network weights don't represent new knowledge and reduces variance as they're quite stable.