Reinforcement Learning Coursework 2

s1575408

1 Question 1

1.1 a

Actor critic method is a Temporal difference method that separate the value function and the policy, but the value function is still influence the policy. Simply, there is two function to make a decision a value function known as critic and the policy stucture known as actor. In my imagination, it's like horse riding (but you leave the horse running without directing it all the time), so the critic is a rider and the actor is a horse, they are separate but actually unify together to go to the optimal solution. So the horse will make an action and the rider will assses/critic the action made by the horse. Assesement based on the TD error, if it's positve than the taken action need to be strengthen for the future, on other hand it's need to be weakened. It's like the rider will slap the horse if the horse going to wrong direction and the horse will learn that going there is not good. In the book they use $P(s_t, a_t)$ as a probability of taking an action in state t, which will be strengthen or weakend by the critic. And the decision is made using softmax function.

One of the implementation of actor critic Is functional electrical stimulation control of a human arm. It implemented by make clinically relevant changes to the dynamics of the arm and test the actor-critic ability to adapt without supervision in a reasonable number of episodes. The advantage of using actor critic is it reduce the dimensionality of the compute function. If we have a problem with m-dimensional state space and n-dimensional action space than Q learning agent will need to compute function $\mathbb{R}m$ x $\mathbb{R}n$ times while actor-critic, this problem is reduced to two lower-dimensional problems: learning the value function $\mathbb{R}m$ and learning the policyf $\mathbb{R}m$ -> $\mathbb{R}n$. Because actor decide the action on the state based on the policy parameter of the actor on the state so we only need to compute maximum action on a state and then take the action and critic will evaluate the action.

1.2 b

Sarsa can be used as a critic function and the actor can be a policy generation function such as greedy.

2 Question 2

2.1 1

θ_t^T is a array of parameter with size [number of features x 1]
$\phi_{s,a}$ is a multi-array with size [number of possible action x number of features]. it contains the array of features for each action.

We don't need to save all the possible state and action. Because we are calculating each $Q(s, a)$ based only the $state_t$ and the features vector $Q(s, a)$.
$Q_t(s, a) = \theta_t^T * \phi_{s,a}$
Our $Q(s, a)$ dimension will be [1 x number of possible action].

2.2 2

In this experiment I use 11 features. each feature will turn on (the value is 1) or turn off (the value is 0) of every [Action with to take, the feature nunber]. we use alpha = $e * 10^{-5}$. gamma = 0.9, epsilon = 0.01. we initialize parameter θ to 0.1

- 1^{st} feature will check enemy in front of our player in the grid. we get the nearest enemy to the player and check the distance with the player (index 0).
  We want our player to brake and turn left or ride instead to avoid collision.
  if there is an enemy and it position is 3 grid or below from our car (vertically). the we will set
  Action Brake - Feature 1 = 1
  Action Left - Feature 1 = 1
  Action Right - Feature 1 = 1
  if there is no enemy in front of our car then we are accelerating
  Action Accelerate - Feature 1 = 1

- 2^{nd} feature will make the player staying in the middle of the road by encouraging it to go left if it's too right and vice versa.
  If the player position on the horizontal grid (row on index 0) is below 3 then it's too left so we encourage it to go right
  Action Right - Feature 2 = 1
  Otherwise, if the player position is more than 8 than we encourage to go left
  Action Left - Feature 2 = 1

- 3^{rd} is predicting the future collision based on the pixel of the enemies an the player, we use collision function. at first we are increasing the y-pixel of our player (-20) this will make the our player move upper. If there is a collision happening than we are incrementing the parameter vector then we encourage brake.
  Action Brake - Feature 3 += 1
  We also encourage to turn left or right, because braking itself will not solve the issue.
  If there is a collision and the enemy is on the left or of our player (enemy-x - player-x < 0). then we are encouraging right Action Right - Feature 3 += 1

  If there is a collision and the enemy is on the right or of our player (enemy-x - player-x > 0). then we are encouraging left Action left - Feature 3 += 1
  If there is no collision we encourage to accelerate Action Accelerate - Feature 3 += 1

- 4^{th} feature will check the speed of our car, if collision happen or we are doing to much brake and left or right action then the speed of our car will be negative, this is not good because we

are encouraging the player to accelerate.
the action at the time the speed become negative is penalize set to 0. while another action is set to 1.
because we don't want to keep penalizing the function, so we use counter to only penalize the 10^{th} first beginning of function.

- 5^{th} feature will check the number of total enemy in the front,left or right of our player and in the right of our player.
  If there is an enemy in the left is more than in the right than we are encouraging to turn right.
  Action Right - Feature 5 = 1
  Otherwise we are encourage to turn left.
  Action Left - Feature 5 = 1

- 6^{th} feature will check if our reward is always zero. this means that we are stuck in one action and we need to move to another action. The counter is used to calculate how many time we got zero score if it's already 30 iteration. then we are penalize the current action and we are encouraging another action by setting the feature to 1.
  Other than current Action - Feature 6 = 1

- 7^{th} feature like the previous feature we are calculating the number of enemy in our left and right, but here we are doing it to the 2 grid before and 2 grid after of our agent. if more player in the left we are encourage to left and otherwise. we also check if we have an enemy in 3 grid infront of our player, if so we are encourage to brake otherwise we are accelerate.
- 8^{th} feature will check if there is two or more an enemy in the left if so we are encourage brake and left, so does if there is the right. otherwise we are accelerate
- 9^{th} feature will check if there is a collition betwen enemy and our player. As before we use pixel representation . the different with the previous version is we make our enemy appear more bigge (width + 5 and height + 10 pixels) and move our enemy closer (player-y - 20).
- 10^{th} is just a bias that will activate all the feature for all actions.

3 3

3.1 a

From figure 1. shows the total reward of linear function approximation get for every epoch. we can see that our agent learning curve sometimes show a learning, for example for epoch 0 - 250 or 1000-1250. But at some point the plot show that our agent reward is decreasing.
If we compare it with q-agent (figure 2.) the q-learning is more able to learn better and increasing every time, while our agent is very spiky and sometimes learn better or sometimes worst.

3.2 b

If we see figure 3. we use initialization 0.1 for theta, and it shows that feature 10 and feature 1 is the most important feature because the value is keep increasing for each epoch, while figure 3 is the most unimportant for all the features. Feature 9, 4, 3 has a decrease value over epoch while the other is increasing. this plot show that the model try to strengthen the believe of some feature to help them get more reward.
If we sort the usefulness of each feature we will get.
feature 10.
feature 1.
feature 8.
feature 7.
feature 5.
feature 2.
feature 9.
feature 4.
feature 3.

3.3 c

If we compare the q-learning and linear function approximation (figure 1 and figure 2), we can see that Q learning converge faster than the linear function. the linear function approximation for 2000 epoch shows no convergence. Probably if we use more features and more epoch we will reach convergence.

we also try to investigate the x position of the enemies and player. if the position is on the left (enemy-x - player-x < 0)

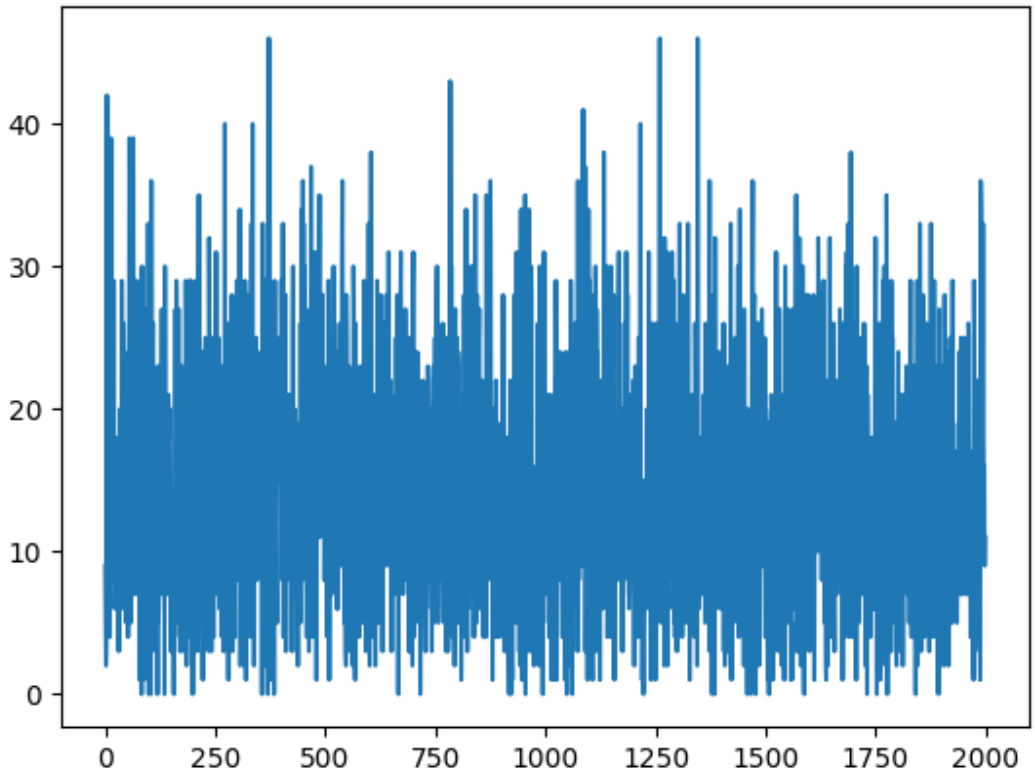

Figure 1: learning curve linear function approximation

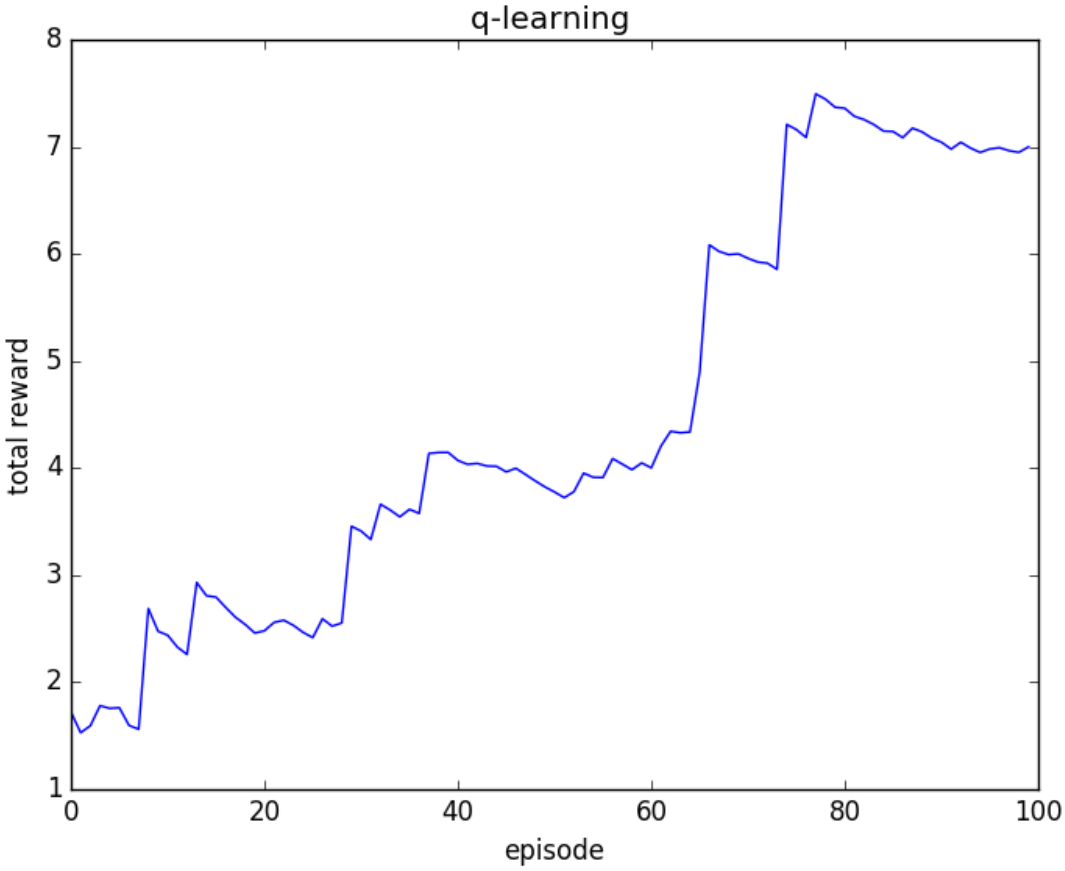

Figure 2: learning curve from q-learning

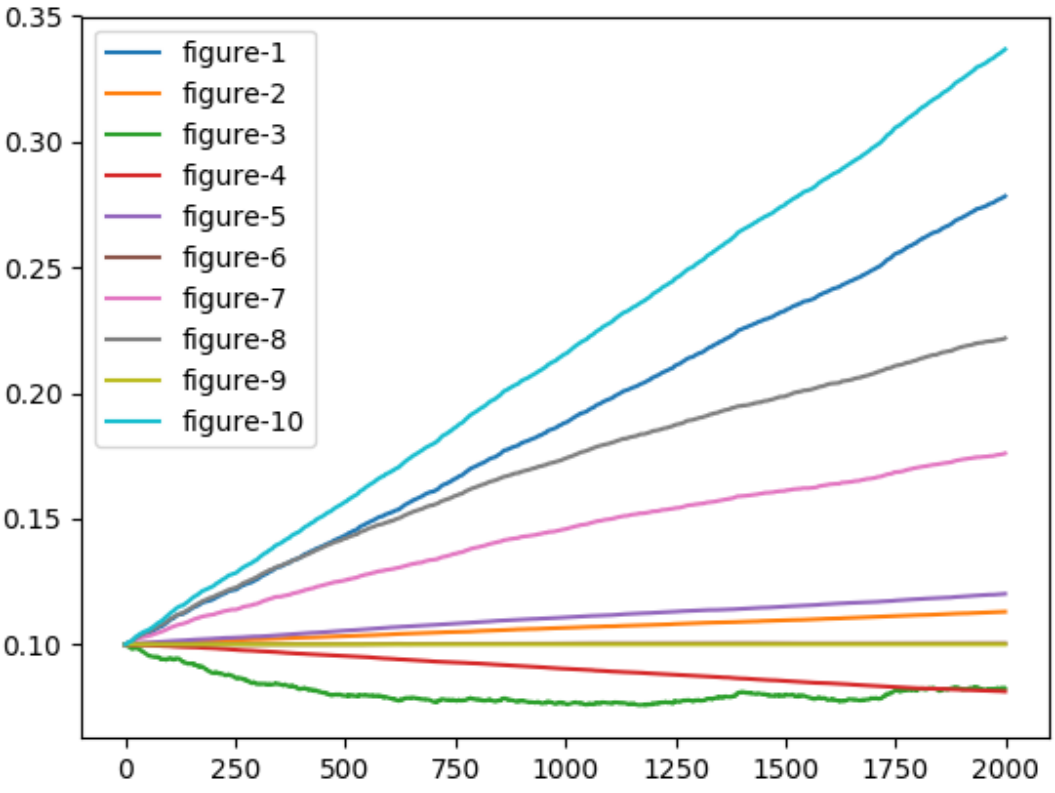

Figure 3: feature weight