Assignment 2

Reinforcement Learning, WS2022/23

Team Members		
Tufan	Mohammad Milad	11902863
Palnak	Fuzail	12129320

1 a,b,c

see python code for that

2 d)

2.1 deterministic Q

2.1.1 influence of ϵ

For finding the perfect ϵ value we just created a range from (0.0 to 1.0) and let the agent train. After 200 episodes the reward was averaged and saved. The following 3 combinations of α and ϵ had the best average reward over 200 episodes for the $\alpha - \epsilon$-tuning, where γ is 0.9.

- $\alpha = 0.1$, $\epsilon = 0.5$, reward = -13.0
- $\alpha = 0.1$, $\epsilon = 0.4$, reward = -15.0
- $\alpha = 0.1$, $\epsilon = 0.2$, reward = -17.0

for the γ and α tuning, where ϵ is 0.5 we got the following results:

- $\alpha = 0.1$, $\gamma = 0.5$, reward = -13.0
- $\alpha = 0.1$, $\gamma = 0.7$, reward = -15.0
- $\alpha = 0.3$, $\gamma = 0.4$, reward = -17.0

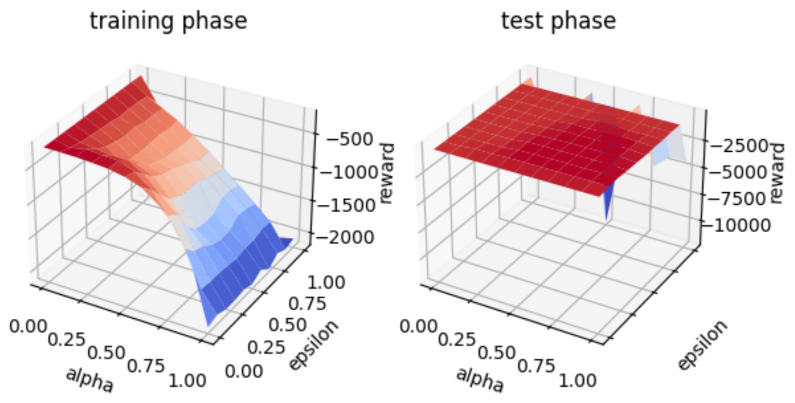

Figure 1: $\alpha - \epsilon$-tuning. Deterministic Q

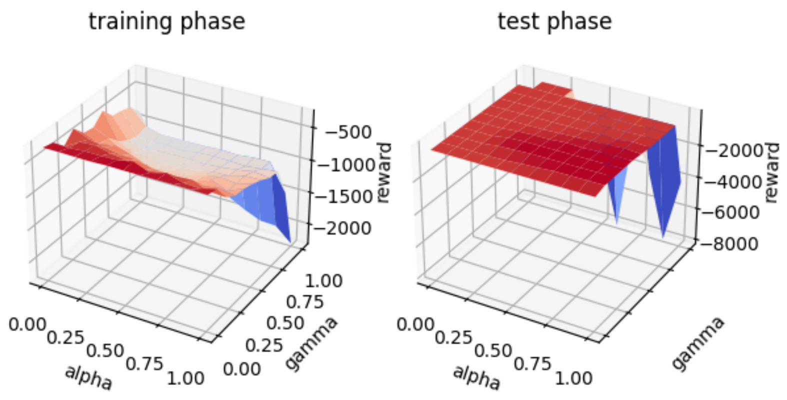

Figure 2: $\alpha - \gamma$-tuning. Deterministic Q

2.2 deterministic SARSA

The following 3 combinations of α and ϵ had the best average reward over 200 episodes for the $\alpha - \epsilon$-tuning, where γ is 0.9.

- $\alpha = 0.2$, $\epsilon = 0.1$, reward = -19.0
- $\alpha = 0.9$, $\epsilon = 0.1$, reward = -21.0
- $\alpha = 0.0$, $\epsilon = 0.0$, reward = -200.0

for the γ and α tuning, where ϵ is 0.1 we got the following results:

- $\alpha = 0.1$, $\gamma = 1.0$, reward = -17.0
- $\alpha = 0.0$, $\gamma = 0.0$, reward = -200.0
- $\alpha = 1.0$, $\gamma = 1.0$, reward = -4160.0

In this case 200 episodes is too little for the deterministic SARSA to be as good as the Q-learning cliff walking example. Nevertheless the top model is nearly as good as the Q-learning models.

2.3 expected SARSA

The following 3 combinations of α and ϵ had the best average reward over 200 episodes for the $\alpha - \epsilon$-tuning, where γ is 0.9.

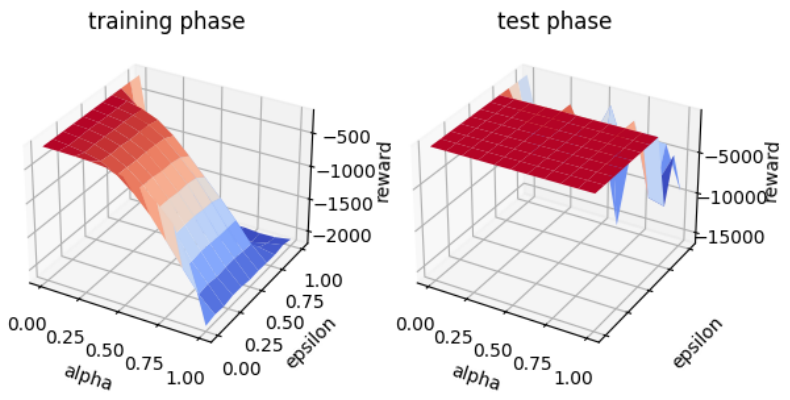

Figure 3: $\alpha - \epsilon$-tuning. Deterministic SARSA

- $\alpha = 0.4$, $\epsilon = 0.1$, reward = -15.0
- $\alpha = 0.4$, $\epsilon = 0.0$, reward = -17.0
- $\alpha = 0.4$, $\epsilon = 0.3$, reward = -21.0

for the γ and α tuning, where ϵ is 0.1 we got the following results:

- $\alpha = 0.4$, $\gamma = 0.9$, reward = -17.0
- $\alpha = 0.0$, $\gamma = 0.0$, reward = -200.0
- $\alpha = 0.8$, $\gamma = 0.9$, reward = -2972.0

In this case our top model reaches the goal, since a reward of -17 means we did not fall into the cliff.

2.4 Fixing ϵ

The best value for ϵ varies from algorithm to algorithm. For Q-learning the best ϵ was 0.5. For deterministic SARSA it was 0.1 and for the expected SARSA it was 0.0. In general an ϵ of 0.0 delivered a reward of -200 which did not solve the cliff walking problem. The exception to that is the Expected SARSA where an ϵ of 0.0 actually solved our problem, yielding a reward of -17.0.

In general fixing ϵ to the best value as tuned in the α-ϵ-tuning, delivered worse results for the deterministic SARSA than just having γ as 0.9. For the Q-learning it actually did not change the rewards and for the expected SARSA

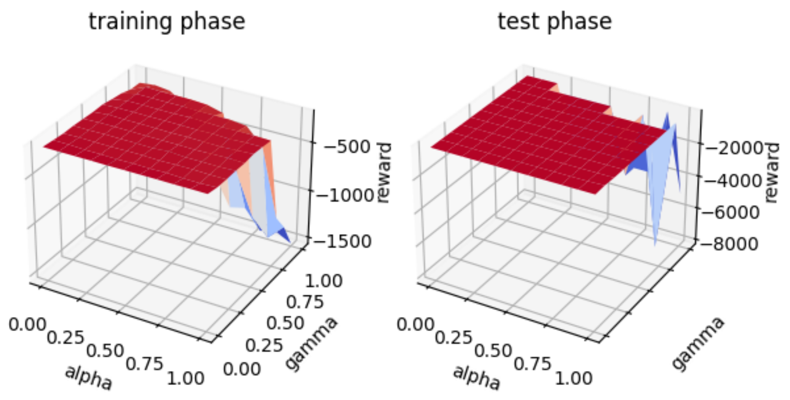

Figure 4: $\alpha - \gamma$-tuning. Deterministic SARSA

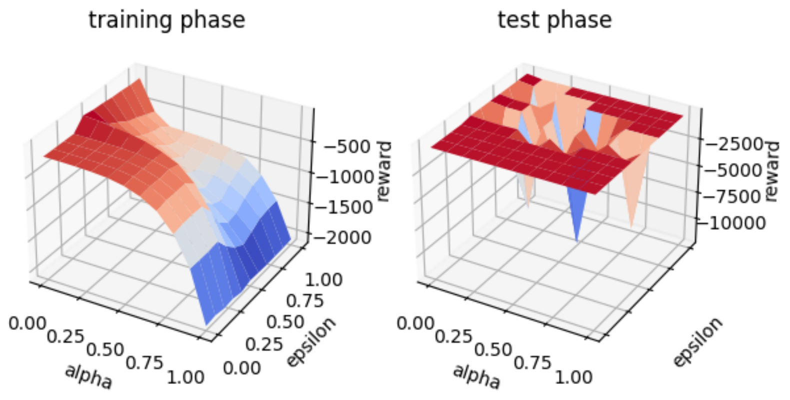

Figure 5: $\alpha - \epsilon$-tuning. Expected SARSA

it's also worse than without fixing the ϵ.

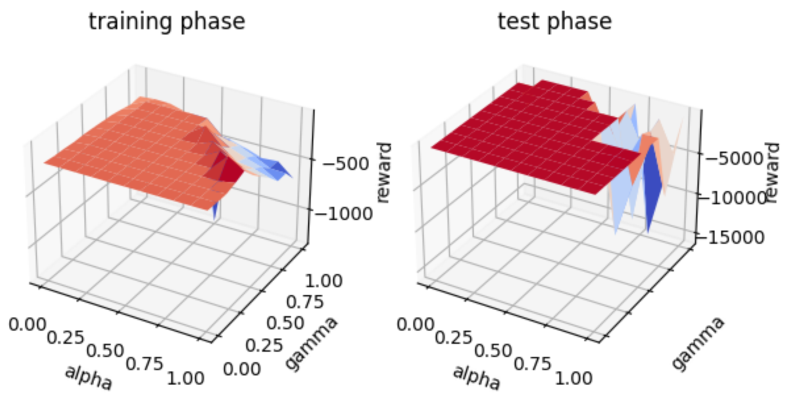

Figure 6: $\alpha - \gamma$-tuning. Expected SARSA

2.5 In theory, for a continuing MDP, convergence of the policy is only ensured for $\gamma < 1$, but here $\gamma = 1$ works also in practice and in theory. Why?

The discount factor γ determines the importance of future rewards compared to immediate rewards, and a value of $\gamma < 1$ means that future rewards are discounted and therefore less important. As a result, the agent will be more likely to converge on a policy that maximizes the expected sum of discounted future rewards.

The the grid world environment is typically finite, with a limited number of states and actions, and the agent will eventually visit all states and take all actions. As a result, the expected sum of discounted future rewards will eventually become equal to the expected sum of undiscounted future rewards, and the value of γ will not have an impact on the policy, therefore, setting $\gamma = 1$ can also lead to convergence of the policy in grid world environments.

TASK 2

3 (a)

see attached handwritten proof

4 (b)

4.1 What is the parameter eps_decay? How will it affect the conver- gence of the algorithm?

The parameter *eps_decay*, decays the *eps* parameter, which is the exploration vs exploitation rate, it controls on how the agent should choose between exploring and exploiting, In the epsilon-greedy policy.

If the exploration rate is set too high, the agent may spend too much time exploring and may not learn an optimal policy. This can lead to slow convergence or even non-convergence of the algorithm. On the other hand, if the exploration rate is set too low, the agent may not explore enough and may become stuck in a sub-optimal policy. This can also lead to slow convergence or non-convergence.

5 (c)

5.1 Can you find a set of parameters that works better? Find a suitable learning rate for the Adam optimizer

From our experiment we conclude that linear SARSA fails to solve to problem as evident from the figure (7), with non linear SARSA we observe when learning is set to $1e-02$ and *max_episode_length=200* (8) gives better result as compared to $1e-01$ (10) and $1e-03$ (9)

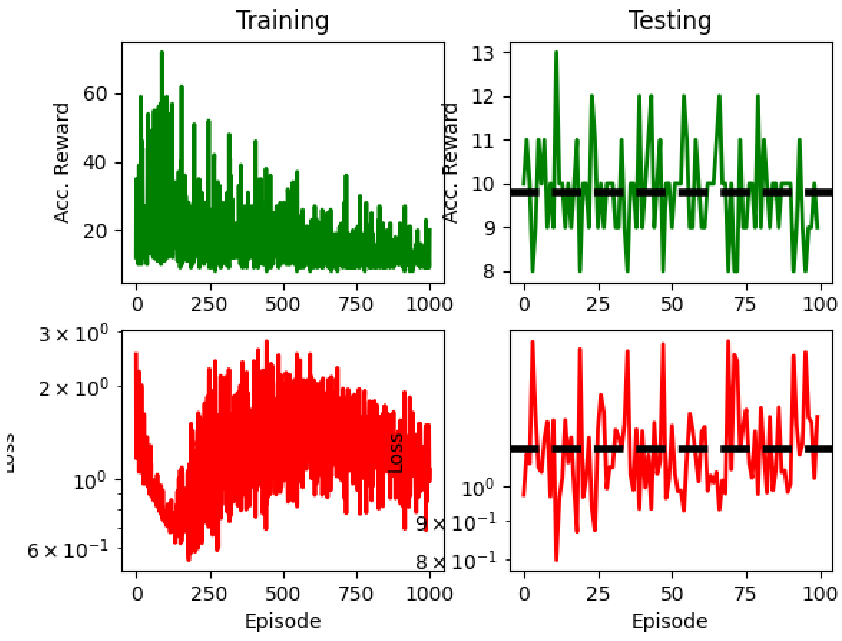

Figure 7: linear SARSA

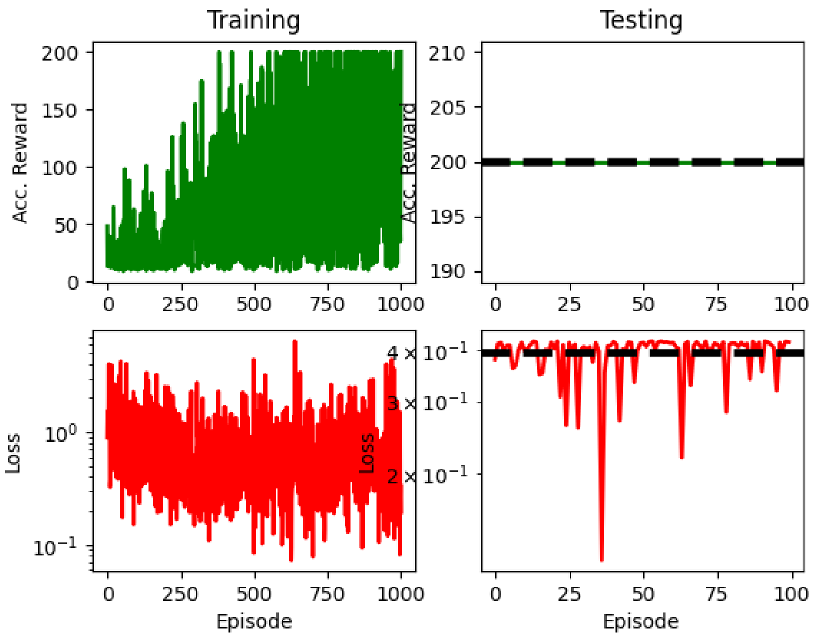

Figure 8: Non linear SARSA: learning rate $1e-02$

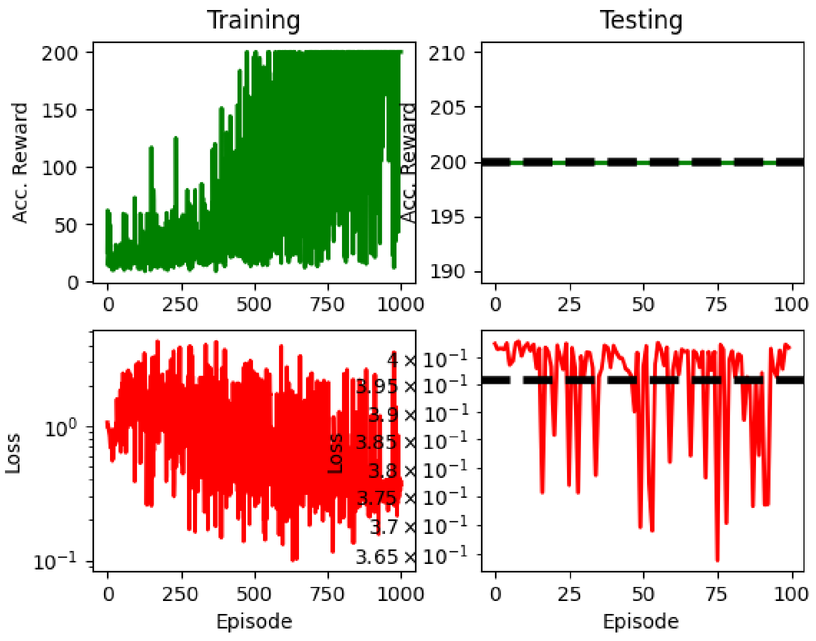

Figure 9: Non linear SARSA: learning rate $1e-03$

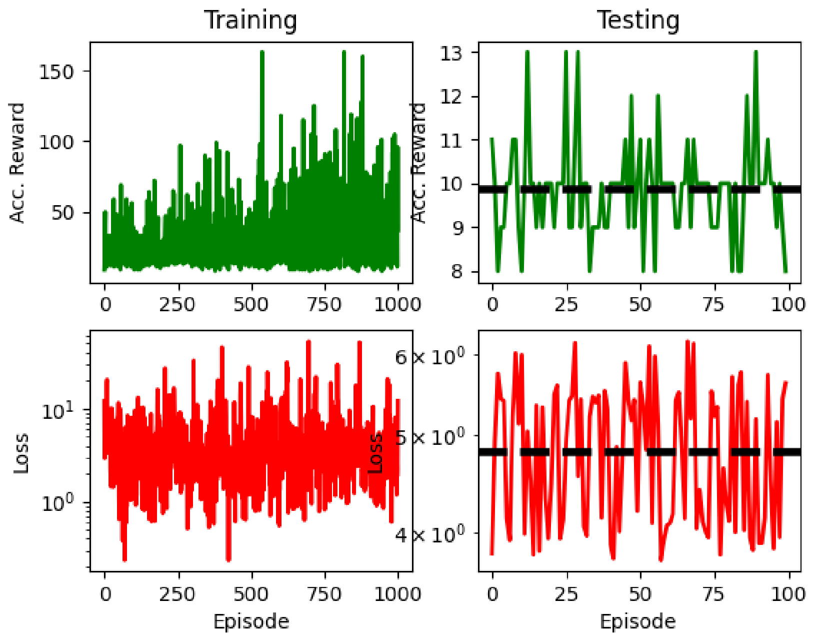

Figure 10: Non linear SARSA: learning rate $1e-01$

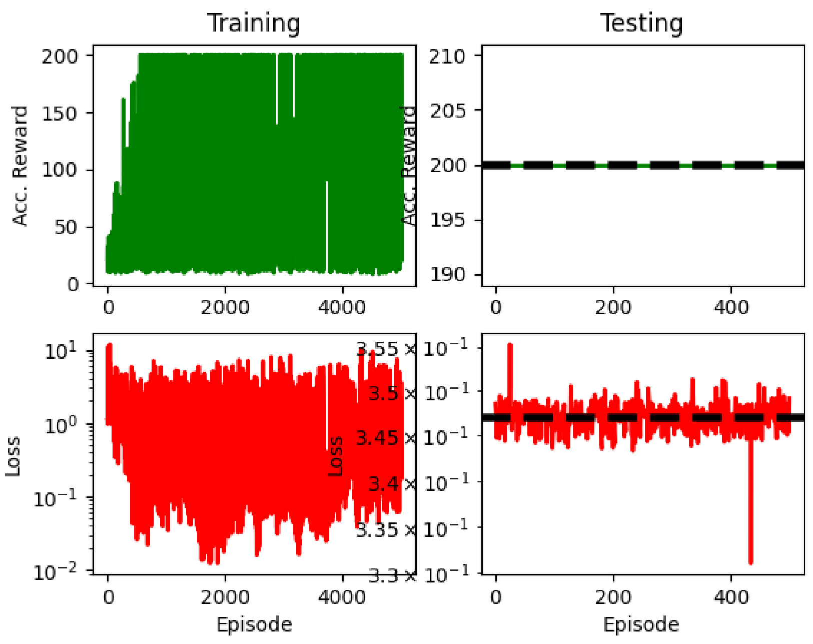

Figure 11: Non linear cart pole with Q-learning, learning rate $1e-03$

6 (d)

6.1 "Cart-Pole with Q-learning Off Policy"

We trained a 2 layer, non linear cart-pole agent using Q-learning, with the following parameters (listing: 1) and obtained the following results (11).

```
# PARAMETERS 1
alpha=1e-03, eps=1, gamma=0.9, eps_decay=0.999,
max_train_iterations=5000, alpha_decay=0.999,
max_test_iterations=500, max_episode_length=2000
```

Listing 1: parameters

We trained a linear cart pole agent using Q-learning, with the following parameters (listing: 2) and obtained the following results (12).

```
# PARAMETERS 1
alpha=1e-03, eps=1, gamma=0.9, eps_decay=0.999,
max_train_iterations=2000, alpha_decay=0.999,
max_test_iterations=500, max_episode_length=500)
```

Listing 2: parameters

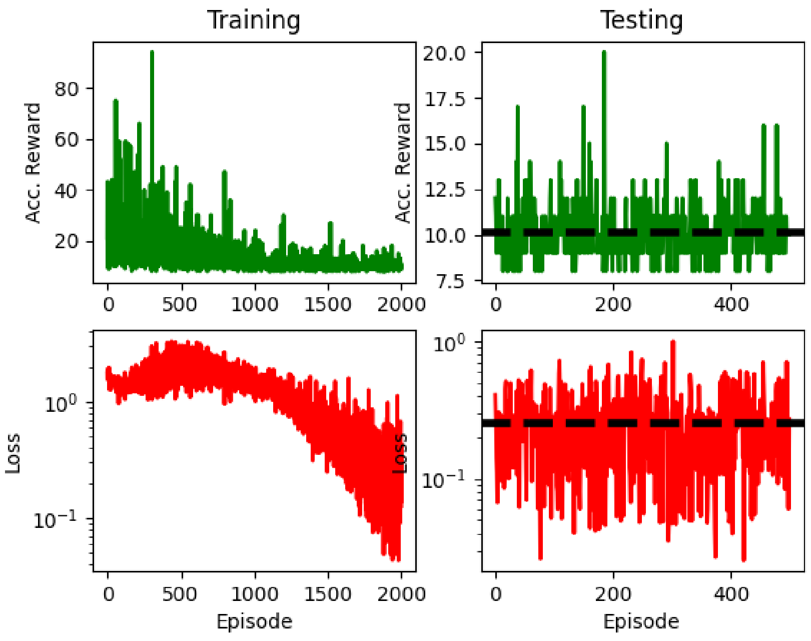

Figure 12: Linear cart pole with Q-learning, learning rate $1e - 03$

6.2 ”Mountain Car”

We trained mountain with SARSA with the reward (Listing : 3) and here are our findings.

Reward 1

In Listing : 3 the idea is to accelerate the agent based on its velocity.

```
    reward = float(reward + 10 * abs(next_state[1]))
```

Listing 3: reward-1

Non-Linear SARSA

We trained the mountain car agent using Non-Linear SARSA with the parameters (Listing : 4) and we observed (13) that the agent takes time to learn the environment and optimize the policy that could solve the task.

```
# PARAMETERS
alpha=1e-3,
eps=1,
gamma=0.9,
eps_decay=0.999,
max_train_iterations=10000,
alpha_decay=0.999,
max_test_iterations=100,
max_episode_length=5000,
```

Listing 4: parameters

Linear SARSA

```
# PARAMETERS
alpha=1e-3,
eps=1,
gamma=0.9,
eps_decay=0.999,
max_train_iterations=25000,
alpha_decay=0.999,
max_test_iterations=100,
max_episode_length=5000,
```

Listing 5: parameters

We trained the mountain car agent using Linear SARSA with the parameters (Listing : 5) and we observed (14) that the agent did not make any progress in optimizing the policy.

We observe that when we add non-linearity (13) the agent starts to make progress in its learning and we also observed that the agent also requires more training iterations.

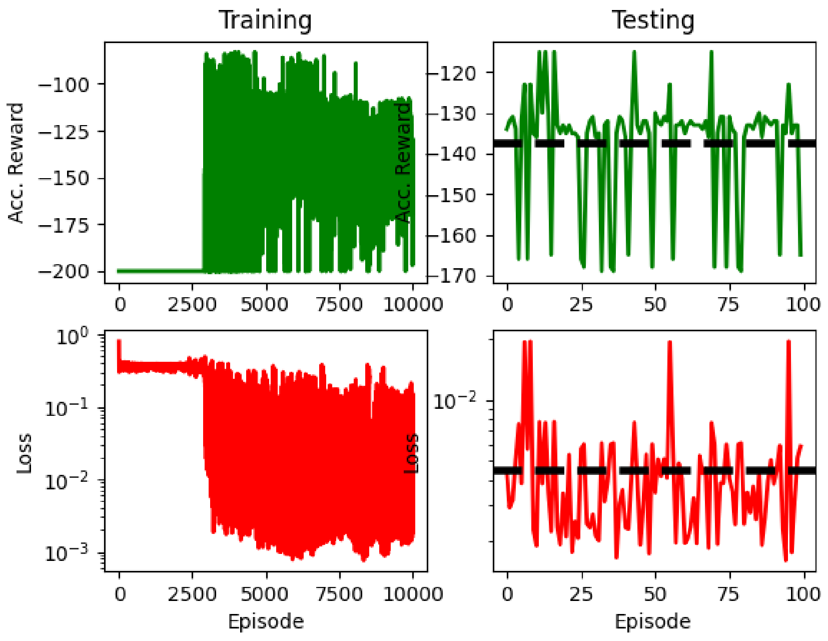

Figure 13: mountain car with non-linear SARSA

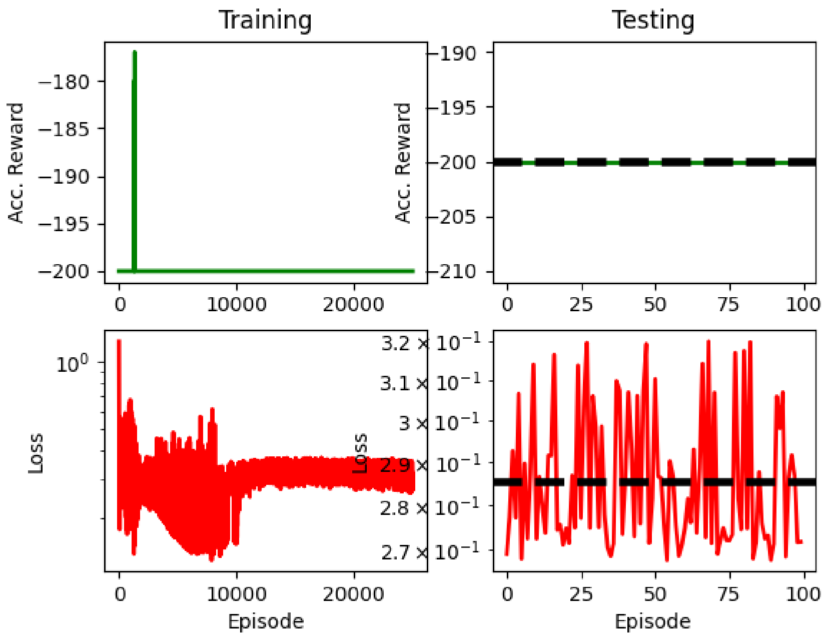

Figure 14: mountain car with linear SARSA

$$E = \frac{1}{2}(Q(s,a) - y)^2$$

$$Q_\theta(s,a) = \theta_a^T s$$

Therefore,

$$E = \frac{1}{2}(\theta_a^T s - y)^2$$

1) "y" is assumed to be constant.

$$\frac{\partial E}{\partial \theta} = \frac{\partial}{\partial \theta}\left(\frac{1}{2}(\theta_a^T s - y)^2\right)$$

$$= (\theta_a^T s - y) \cdot s$$

The update rule is given as

$$\theta \leftarrow \theta - \alpha \frac{\partial E}{\partial \theta}$$

$$\boxed{\therefore \theta_a \leftarrow \theta_a - \alpha (\theta_a^T s - y) s}$$

2) If $y = r + \gamma \theta_\theta(s', a')$

a) $$\frac{\partial E}{\partial \theta_a} = \frac{\partial}{\partial \theta_a}\left(\frac{1}{2}\left(\theta_a^T s - (r + \gamma \theta_{a'}^T s')\right)^2\right)$$

$$\frac{\partial E}{\partial \theta_a} = \theta_a^T s - (r + \gamma \theta_{a'}^T s') * s$$

Therefore using the update rule.

$$\theta_a \leftarrow \theta_a - \alpha \left(\theta_a^T s - (r + \gamma \theta_{a'}^T s') * s\right)$$

$$\boxed{\theta_a \leftarrow \theta_a - \alpha\,(\theta_a^T s - y) * s}$$

b) $$\frac{\partial E}{\partial \theta_{a'}} = \frac{\partial E}{\partial \theta_{a'}}\left(\tfrac{1}{2}\left(\theta_a^T s - (r + \gamma \theta_{a'}^T s')\right)^2\right)$$

$$= \left(\theta_a^T s - (r + \gamma \theta_{a'}^T s')\right) * (-\gamma s')$$

$$= -\gamma\left(\theta_a^T s - (r + \gamma \theta_{a'}^T s')\right) * s'$$

Using the update rule.

$$\boxed{\theta_{a'} \leftarrow \theta_{a'} + \alpha\gamma\left(\theta_a^T s - (r + \gamma \theta_{a'}^T s')\right) * s'}$$

The parameter update.

$$\boxed{\begin{aligned} \theta_a &\leftarrow \theta_a - \alpha\,(Q_\theta(s,a) - y)\, s \\ \theta_{a'} &\leftarrow \theta_{a'} + \alpha\gamma\,(Q_\theta(s,a) - y)\, s' \end{aligned}}$$

is equivalent to the Q-Learning algorithm with a standard Q-table when the state rewritten as a one-hot encoded vector in a grid world environment.

The standard Q learning update rule is given as -

$$\boxed{Q(S,a) \leftarrow Q(S,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))}$$

If we substitue $Q_\theta(S,a) = \theta_a^T s$ in the parameter update rule we get

$$\boxed{\begin{aligned} \theta_a &\leftarrow \theta_a + \alpha\left(\theta_a^T s - r + \gamma(\theta_{a'}^T s')\right)s \\ \theta_{a'} &\leftarrow \theta_{a'} + \alpha\gamma\left(\theta_a^T s - r + \gamma(\theta_{a'}^T s')\right)s' \end{aligned}}$$

Now assuming "s" is a one-hot encoded vector, then the dot product $\theta_a^T s$ and $\theta_{a'}^T s'$ will be θ_a^T and $\theta_{a'}^T$ respectively

Substituting these values we get

$$\boxed{\begin{aligned} \theta_a &\leftarrow \theta_a + \alpha(\theta_a - r + \gamma(\theta_{a'}))s \\ \theta_{a'} &\leftarrow \theta_{a'} + \alpha\gamma(\theta_a - r + \gamma(\theta_{a'}))s' \end{aligned}}$$

which is now the same form as the standard Q-Learning update rule, with θ_a and $\theta_{a'}$ taking the place of $Q(S,a)$ and $\max_{a'} Q(S',a')$

Therefore, this parameter update is equivalent to the Q-Learning update rule with a stdard Q table.