INM426: Software Agents
Implementation of Q-learning Algorithm for Table Football

Shawn Ban

March 22, 2018

1 Introduction, aim and motivation

In this project we model a table football game as a Markov Decision Process, and implement a Q-learning algorithm to train an agent to learn efficient policy actions for each state. We then examine the performance of the algorithm under varying conditions of the parameters, such as the learning rate and discount factor. We also evaluate how the algorithm performs under different learning policies, ranging from ϵ-greedy to random.

The motivation of this task is to train an agent to discover efficient policy actions for each state. We initially model the game with simplified assumptions, but these can be relaxed or adapted to represent more complex conditions, such as accounting for the relative strengths and weaknesses of the human players. We conclude by considering a more complex case of state transitions.

2 Problem definition

2.1 Domain and tasks

The domain is a standard 11-a-side table football game configured according to standard International Table Soccer Federation (ITSF) regulations. Each team comprises one goalkeeper, two defenders, five midfielders and three forwards. We also designate a terminal "goal" state, yielding a 12-state space. The agent's task is to learn the optimal path from any point on the network to the goal state so as to optimize its utility function. Figure 1 shows a pictorial representation of one team of a regulation table football game.

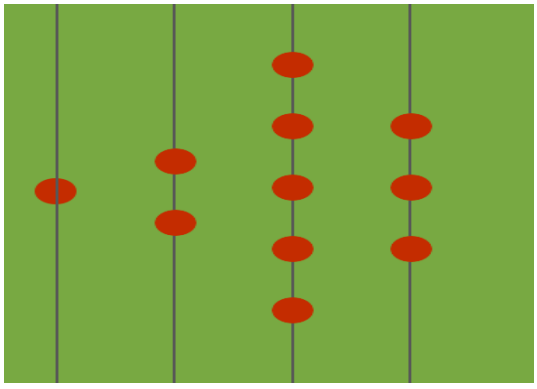

Figure 1: Representation of one team of a table football game

2.2 Representations and assumptions

We model the game as a state space with 12 possible states, representing the 11 players (X_0-X_{10}) and the terminal goal state (X_{11}), with each player being a node. The presence of the agent at a node represents that player's possession of the football. We assume that at each state from X_0 to X_7, a player can either pass to any of the players in the row immediately ahead of it, or attempt a shot on goal. States X_8 to X_{10} represent the most advanced row, and can only attempt a shot on goal.

We further assume that shots on goal from a longer distance have a lower probability of success, and consequently a higher probability of turning the ball over to the other team. As a result, shots from further away will receive a lower expected reward.

2.3 State transition function

We further assume that the agent operates in discrete time and a deterministic environment, so that its state in the next time period, $t+1$, is defined by its action in the current time period, t. More formally, we define the agent's state s as:

$$s_{t+1} = f(s_t, a_t)$$

Considering our earlier assumption that a player can only pass to a player in the row immediately ahead, or attempt a transition to the goal state, yields the following state transitions:

$X_0 \rightarrow \{X_1, X_2, X_{11}\}$
$X_1 \rightarrow \{X_3, X_4, X_5, X_6, X_7, X_{11}\}$
$X_2 \rightarrow \{X_3, X_4, X_5, X_6, X_7, X_{11}\}$
$X_3 \rightarrow \{X_8, X_9, X_{10}, X_{11}\}$
$X_4 \rightarrow \{X_8, X_9, X_{10}, X_{11}\}$
$X_5 \rightarrow \{X_8, X_9, X_{10}, X_{11}\}$
$X_6 \rightarrow \{X_8, X_9, X_{10}, X_{11}\}$
$X_7 \rightarrow \{X_8, X_9, X_{10}, X_{11}\}$
$X_8 \rightarrow \{X_{11}\}$
$X_9 \rightarrow \{X_{11}\}$
$X_{10} \rightarrow \{X_{11}\}$
$X_{11} \rightarrow \{X_{11}\}$

2.4 Reward function

In setting out our reward function, we assign a negative number to every pass to another player, as every pass carries a risk of interception and turning the ball over to the opposing team. Most passes are assigned a reward of -5, with the exception of long cross-field passes from one flank to the other. These passes are characterized as risky and assigned rewards of -10 or -20.

Every transition to the goal state is assigned a positive value belonging to $[10, 25, 50, 75, 100]$. Shots from a further distance are assigned a lower reward, to represent a lower possibility of success. Finally, the transition of X_{11} to itself is assigned a reward of 0, as there is no further reward once a goal is scored. This yields the following reward matrix, R:

R =	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
X0		-5	-5									10
X1				-5	-5	-5	-5	-20				25
X2				-20	-5	-5	-5	-5				25
X3									-5	-5	-10	50
X4									-5	-5	-5	50
X5									-5	-5	-5	50
X6									-5	-5	-5	50
X7									-10	-5	-5	50
X8												75
X9												100
X10												75
X11												0

Figure 2: R-matrix

The rows of the R-matrix represent the current state of the agent and the columns represent the possible actions leading to the next state.

2.5 Choosing a policy

We formally define the learning policy as π, and the associated action a that our agent chooses in a given state s is defined by this policy such that $a = \pi(s)$, with the set of all possible actions denoted by A. One challenge presented in reinforcement learning is balancing the trade-off between exploration and exploitation. To discover the reward function, the agent needs to explore sub-optimal actions, but to obtain the reward, the agent has to exploit actions that it has learned are effective. To manage this balance, we choose an ϵ-greedy policy as the learning policy, where an exploration factor is denoted by $0 \leq \epsilon \leq 1$. In this policy, the agents randomly selects an action a from the set of all possible actions A with probability ϵ (explore), and selects the action with highest reward with probability $1 - \epsilon$ (exploit).

We further add a decay factor $0 \leq \lambda \leq 1$, so that the value of ϵ is updated after each learning epoch as follows: $\epsilon \leftarrow (1 - \lambda)\epsilon$. This allows the agent to explore more often early in the learning, when it knows little about the environment, and exploit more often as the learning progresses. More formally, exploitation is defined by $\pi^*(s) = argmax[Q(s_{valid}, a_{all})]$.

2.6 Graph representation

Figure 2 shows a graphical representation of our state space, with possible state transitions included.

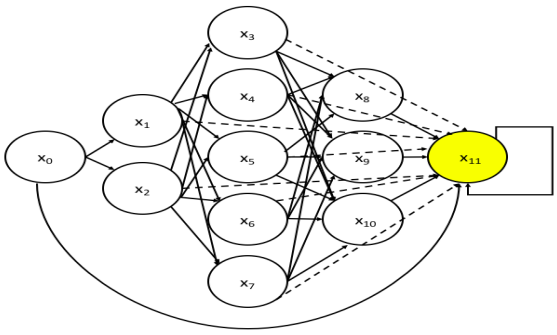

Figure 3: Representation of state transitions

3 Implementing Q-learning

Q-learning is an off-policy temporal difference control algorithm developed by Watkins [1]. It is an off-policy method as the behaviour policy is separate from the estimation policy, which allows

the estimation policy to be greedy while the behaviour policy explores sub-optimal outcomes [2]. In this project, we implement Q-learning in its simplest form, one-step Q-learning. The transition rule of Q-learning is defined by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

3.1 Q-learning parameters

As shown above, the two parameters in the Q-learning algorithm are α, the learning rate, and γ, the discount rate. These parameters affect the behaviour and learning of the agent.

The discount rate $0 \leq \gamma \leq 1$ affects how the agent values future rewards. At $\gamma = 0$, the agent is short-sighted and concerned only with immediate rewards, while at $\gamma = 1$, the agent values future rewards more strongly. The learning rate $0 \leq \alpha \leq 1$ determines to what extent the Q-matrix is updated when the agent learns. At $\alpha = 0$, the agent does not learn at all, while at $\alpha = 1$ the Q-matrix is updated with the full value of the new reward. In this project, we explore how different values of γ and α affect learning.

3.2 Q-matrix

The Q-matrix represents the agent's knowledge, or memory of what it has learned through experience. We first initialize a new Q-matrix of zeroes of the same dimensions as the R-matrix. Initializing to zero denotes that the agent has zero knowledge of the environment at the beginning. As the agent explores the environment and obtains rewards based on its actions, the Q-matrix is updated by the transition rule above.

When the agent takes an action, it first refers to the R-matrix to assess the valid state transitions. It then refers to the Q-matrix to assess the value of being in that state. As the agent explores, the Q-matrix will be updated and populated with information for the agent to exploit. If all state-action pairs are visited and updated, the Q-learning algorithm has been shown to converge to Q^*, the optimal action-value function [1].

3.3 Single learning episode walkthrough

To demonstrate the Q-learning algorithm, we walk through the first learning epoch. At this stage, it is worth reviewing the algorithm in pseudocode:

For each training episode:
  initialize s randomly
  determine available a from s using R-matrix
  choose a using ϵ-greedy policy derived from Q-matrix
  take action a, observe reward r and new state s'
  update $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)]$
  update $s \leftarrow s'$
  update $\epsilon \leftarrow (1 - \lambda)\epsilon$
until s is terminal (X_{11}).

For our numerical example, we set the parameters as follows: $\epsilon = 0.7$, $\lambda = 0.01$, $\alpha = 0.6$, $\gamma = 0.4$. We randomly initialize the state s as X_3. Referring to the R-matrix, the possible actions are to move to X_8, X_9, X_{10} or X_{11}.

We then generate a random number p between 0 and 1. In this case $p = 0.63$ and as $\epsilon > p$, the policy decision is to explore. The agent randomly chooses a state to move to from the possible actions, in this case X_{10} and obtains a reward of -10 from the R-matrix: $R[X_3, X_{10}] = -5$. The

Q-matrix is then updated as shown in Figure 4:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$$
$$Q(X_3, X_{10}) \leftarrow 0 + 0.6 * [-10 + 0.4 * max_{a'} Q(X_{10}, a') - 0]$$
$$Q(X_3, X_{10}) \leftarrow -6$$

Q =	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
X0	0	0	0	0	0	0	0	0	0	0	0	0
X1	0	0	0	0	0	0	0	0	0	0	0	0
X2	0	0	0	0	0	0	0	0	0	0	0	0
X3	0	0	0	0	0	0	0	0	0	0	-6	0
X4	0	0	0	0	0	0	0	0	0	0	0	0
X5	0	0	0	0	0	0	0	0	0	0	0	0
X6	0	0	0	0	0	0	0	0	0	0	0	0
X7	0	0	0	0	0	0	0	0	0	0	0	0
X8	0	0	0	0	0	0	0	0	0	0	0	0
X9	0	0	0	0	0	0	0	0	0	0	0	0
X10	0	0	0	0	0	0	0	0	0	0	0	0
X11	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4: Q-matrix after first step, update highlighted

Q =	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
X0	0	0	0	0	0	0	0	0	0	0	0	0
X1	0	0	0	0	0	0	0	0	0	0	0	0
X2	0	0	0	0	0	0	0	0	0	0	0	0
X3	0	0	0	0	0	0	0	0	0	0	-6	0
X4	0	0	0	0	0	0	0	0	0	0	0	0
X5	0	0	0	0	0	0	0	0	0	0	0	0
X6	0	0	0	0	0	0	0	0	0	0	0	0
X7	0	0	0	0	0	0	0	0	0	0	0	0
X8	0	0	0	0	0	0	0	0	0	0	0	0
X9	0	0	0	0	0	0	0	0	0	0	0	0
X10	0	0	0	0	0	0	0	0	0	0	0	45
X11	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5: Q-matrix after second step, update highlighted

We also update s' as the agent is now in state X_{10} and $\epsilon = (1-0.01)*0.7 = 0.693$. In the second pass of the first training episode, the only possible action is to move to X_{11}. The explore-or-exploit decision becomes irrelevant and both policies lead to X_{11} with a reward of 75: $R[X_{10}, X_{11}] = 75$. We again update the Q-matrix, as shown in Figure 5:

$$Q(X_{10}, X_{11}) \leftarrow 0 + 0.6 * [75 + 0.4 * max_{a'} Q(X_{11}, a') - 0]$$
$$Q(X_{10}, X_{11}) \leftarrow 45$$

As the agent has reached the terminal state X_{11}, this concludes the first learning episode.

4 Experimental results

We choose a base case and compare its performance against 6 other cases with varying parameter values. In order to analyze the results qualitatively, we run each case for 3000 training episodes without implementing early stopping.

4.1 Base case, performance vs episodes

For our base case, we consider the following parameters: $\epsilon = 0.9$, $\lambda = 0.001$, $\alpha = 0.5$, $\gamma = 0.7$. As a performance metric, we calculate the accumulated rewards divided by the accumulated number of steps for all prior episodes plot this against the episode number (iteration). This yields the following plot and optimal paths learned after 3000 episodes:

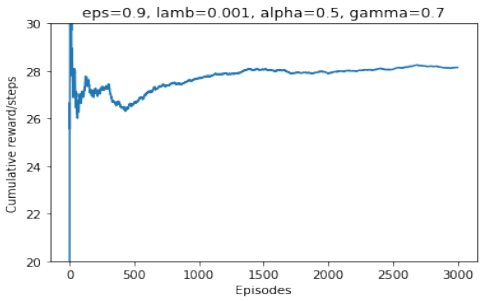

Figure 6: Average cumulative reward vs epsiode

```
0->1->3->9->11
1->3->9->11
2->4->9->11
3->9->11
4->9->11
5->9->11
6->9->11
7->9->11
8->11
9->11
10->11
11
```

Figure 7: Optimal paths learned

These results show that the algorithm converges to a value of about 28 after around 2000 iterations. The updated Q-matrix after 3000 iterations is instructive, as it shows the result of continuing with our earlier walkthrough. We can interpret the updated Q-matrix as the agent's knowledge of the expected rewards of each state transition after 3000 training episodes.

Q =

	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
X0	0	23	23	0	0	0	0	0	0	0	0	10
X1	0	0	0	33	40	40	37	25	0	0	0	25
X2	0	0	0	25	40	40	40	40	0	0	0	25
X3	0	0	0	0	0	0	0	0	46	65	42	50
X4	0	0	0	0	0	0	0	0	47	65	47	50
X5	0	0	0	0	0	0	0	0	47	65	47	50
X6	0	0	0	0	0	0	0	0	47	65	47	50
X7	0	0	0	0	0	0	0	0	42	65	47	50
X8	0	0	0	0	0	0	0	0	0	0	0	75
X9	0	0	0	0	0	0	0	0	0	0	0	100
X10	0	0	0	0	0	0	0	0	0	0	0	75
X11	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8: Updated Q-matrix after 3000 episodes

4.2 Results with different parameter values

We consider 6 other cases to illustrate the effect of varying the parameter values on the learning of the agent. In each case we adjust one parameter to an extreme, and hold the other parameters constant.

- Case 1 (high ϵ, favours exploration over exploitation): $\epsilon = 0.999$, $\lambda = 0.001$, $\alpha = 0.5$, $\gamma = 0.7$.
- Case 2 (low ϵ, favours exploitation over exploration): $\epsilon = 0.001$, $\lambda = 0.001$, $\alpha = 0.5$, $\gamma = 0.7$.
- Case 3 (high α, learn quickly): $\epsilon = 0.9$, $\lambda = 0.001$, $\alpha = 0.999$, $\gamma = 0.7$.
- Case 4 (low α, learn slowly): $\epsilon = 0.9$, $\lambda = 0.001$, $\alpha = 0.001$, $\gamma = 0.7$.
- Case 5 (high γ, favour future rewards): $\epsilon = 0.9$, $\lambda = 0.01$, $\alpha = 0.5$, $\gamma = 0.999$.
- Case 6 (low γ, favour immediate rewards): $\epsilon = 0.9$, $\lambda = 0.01$, $\alpha = 0.5$, $\gamma = 0.001$.

4.2.1 Case 1: high ϵ, favour exploration over exploitation

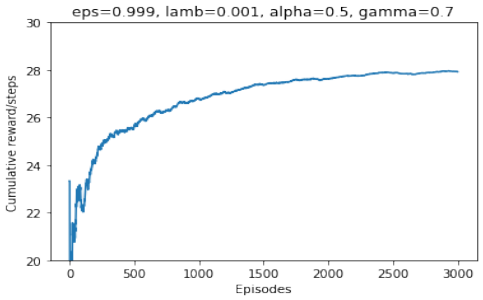

Figure 9: Average cumulative reward vs epsiode

```
0->1->3->9->11
1->3->9->11
2->4->9->11
3->9->11
4->9->11
5->9->11
6->9->11
7->9->11
8->11
9->11
10->11
11
```

Figure 10: Optimal paths learned

In this case, the algorithm converges to the same value as the base case, but at a slower pace. This is because the agent has a higher probability of exploration, even when it has already learned the optimal solution.

4.2.2 Case 2: low ϵ, favour exploitation over exploration

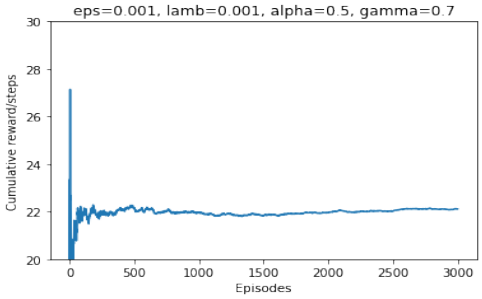

Figure 11: Average cumulative reward vs epsiode

```
0->2->3->8->11
1->3->8->11
2->3->8->11
3->8->11
4->8->11
5->9->11
6->8->11
7->8->11
8->11
9->11
10->11
11
```

Figure 12: Optimal paths learned

These results show the algorithm converging to a lower value than the base case. Performance is poor as the agent has a high probability of exploiting the rewards, and does not explore the state space as much. Note that the paths discovered in this case are different than in the base case, and it has not discovered the optimal route of passing to state X_9.

4.2.3 Case 3: high α, learn quickly

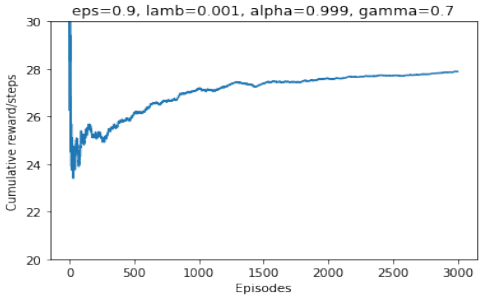

Figure 13: Average cumulative reward vs epsiode

```
0->1->3->9->11
1->3->9->11
2->4->9->11
3->9->11
4->9->11
5->9->11
6->9->11
7->9->11
8->11
9->11
10->11
11
```

Figure 14: Optimal paths learned

The algorithm converges after around 2000 iterations. In this case, there is not much difference from the base case.

4.2.4 Case 4: low α, learn slowly

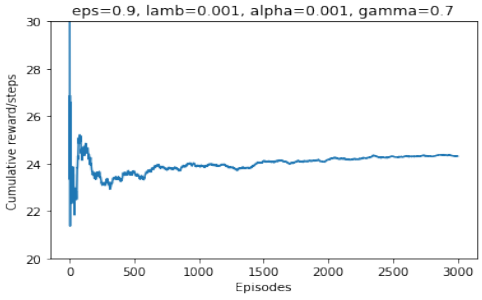

Figure 15: Average cumulative reward vs epsiode

```
0->11
1->11
2->11
3->11
4->11
5->11
6->11
7->11
8->11
9->11
10->11
11
```

Figure 16: Optimal paths learned

At a very low learning rate, not much learning takes place, and the algorithm has yet to converge after 3000 iterations. It has yet to discover optimal paths beyond immediately trying to score a goal.

4.2.5 Case 5: high γ, favour future rewards

The algorithm converges after around 2000 iterations. Again, there is not much difference from the base case, showing that the base case is "far-sighted" enough to learn optimal policies.

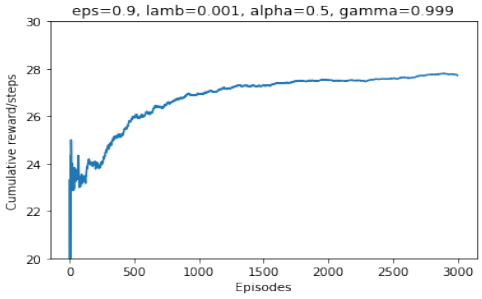

Figure 17: Average cumulative reward vs epsiode

```
0->1->5->9->11
1->5->9->11
2->4->9->11
3->9->11
4->9->11
5->9->11
6->9->11
7->9->11
8->11
9->11
10->11
11
```

Figure 18: Optimal paths learned

4.2.6 Case 6: low γ, favour immediate rewards

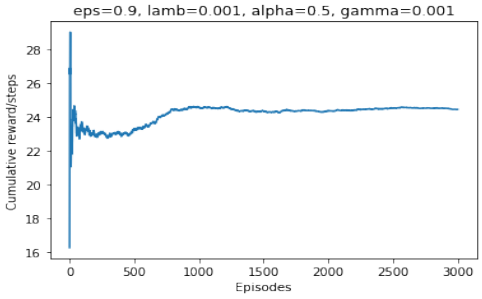

Figure 19: Average cumulative reward vs epsiode

```
0->11
1->11
2->11
3->11
4->11
5->11
6->11
7->11
8->11
9->11
10->11
11
```

Figure 20: Optimal paths learned

The algorithm converges to a suboptimal solution. In this case, it is too myopic and does not allow the agent to traverse longer paths to find a better reward. The agent is incentivised to shoot on goal from every space, i.e. try to get a positive reward as soon as possible.

4.3 Changing the reward function

We now relax some of the earlier assumptions according to the following principles:

- Every player should be able to retain the ball with zero penalty, pass to any other player a negative reward, or attempt a shot on goal, with a positive reward.
- Longer passes should be penalized as they are riskier and carry a higher chance of interception.
- Passing back to the goalkeeper, X_0, should be greatly penalized as this may lead to scoring an own goal.
- The terminal state, X_{11}, remains the same.

While the new state transitions are too complex to be represented graphically, the new R-matrix is as follows:

R =	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
X0	0	-5	-5	-10	-10	-10	-10	-10	-20	-20	-20	10
X1	-50	0	-5	-5	-5	-5	-5	-20	-20	-20	-20	25
X2	-50	-5	0	-20	-5	-5	-5	-5	-20	-20	-20	25
X3	-50	-5	-20	0	-5	-5	-5	-5	-5	-5	-10	50
X4	-50	-5	-5	-5	0	-5	-5	-5	-5	-5	-5	50
X5	-50	-5	-5	-5	-5	0	-5	-5	-5	-5	-5	50
X6	-50	-5	-5	-5	-5	-5	0	-5	-5	-5	-5	50
X7	-50	-20	-5	-5	-5	-5	-5	0	-10	-5	-5	50
X8	-50	-20	-20	-5	-5	-5	-5	-10	0	-5	-5	75
X9	-50	-20	-20	-5	-5	-5	-5	-5	-5	0	-5	100
X10	-50	-20	-20	-10	-5	-5	-5	-5	-5	-5	0	75
X11												0

Figure 21: R-matrix with relaxed assumptions

In this new environment, the algorithm takes longer to converge (more than 3000 iterations) due to the greater number of possible routes. At 3000 iterations, the agent has yet to discover the optimal paths. Note that from state X_1, the agent chooses a patient build-up strategy of passing the ball through midfield to attack, while from state X_2, the agent chooses the long-ball strategy of passing the ball directly to attack, even though the two states are both in defence and have similar reward functions. The agent has also yet to learn to pass the ball from states X_8 and X_{10} to state X_9, choosing to shoot on goal as opposed to passing to the star striker who has a better chance of scoring.

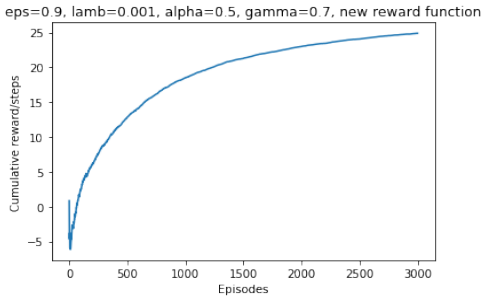

Figure 22: Average cumulative reward vs epsiode

```
0->9->11
1->6->9->11
2->10->11
3->9->11
4->9->11
5->9->11
6->9->11
7->8->11
8->11
9->11
10->11
11
```

Figure 23: Optimal paths learned

4.4 Summary of results

In order to quantitatively analyze the results under different scenarios, we define a stopping criterion for convergence, and terminate the learning algorithm when the variance of the performance (as defined by cumulative rewards divided by steps) of the last 10 training episodes is below 0.00001. We then run each scenario 10 times, and report the mean and standard deviation of the number of episodes for convergence, and the performance at the point of convergence. Figure 24 shows a summary of the results under different parameter values. Note the convergence to a suboptimal performance in cases 2, 4 and 6 as compared to the base case. Case 7 is not directly comparable as it uses a different R-matrix.

						Episodes to converge		Performance	
Case	Description	ε	λ	α	γ	Mean	Std	Mean	Std
0	Base case	0.9	0.001	0.5	0.7	1433.1	124.5	27.4	0.304
1	Explore	0.999	0.001	0.5	0.7	1370.5	150.8	27.2	0.335
2	Exploit	0.001	0.001	0.5	0.7	1199.9	264.8	23.1	0.815
3	Learn fast	0.9	0.001	0.999	0.7	1457.5	120.7	27.4	0.219
4	Learn slow	0.9	0.001	0.001	0.7	1409.9	220.7	24.0	0.436
5	Favour future	0.9	0.001	0.5	0.999	1426.8	97.8	27.2	0.332
6	Discount future	0.9	0.001	0.5	0.001	1666.2	226.9	24.5	0.234
7	New R	0.9	0.001	0.5	0.7	1672.0	265.9	22.9	2.157

Figure 24: Summary of results

5 Other reinforcement learning techniques - SARSA

In this section, we employ SARSA (state-action-reward-state-action), an on-policy temporal difference algorithm. The main difference between Q-learning and SARSA is the way future rewards are found by updating the Q-matrix. In Q-learning, the Q-matrix is updated with the highest possible value of all actions, but in SARSA, the Q-matrix is updated with the value of the actual action that the agent took. Hence SARSA is on-policy in that it accounts for the actual control policy that the agent is following. We first ran the SARSA algorithm with the same parameters as case 0, our base case:

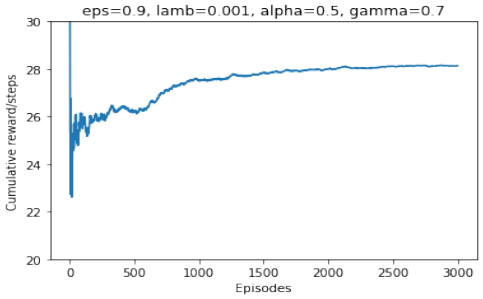

Figure 25: Average cumulative reward vs epsiode

```
0->1->6->9->11
1->6->9->11
2->5->9->11
3->9->11
4->9->11
5->9->11
6->9->11
7->9->11
8->11
9->11
10->11
11
```

Figure 26: Optimal paths learned

Qualitatively, there does not seem to be much difference in learning for this case between Q-learning and SARSA. The performance converges to roughly the same value (28) after a similar number of episodes (2000). However, a large and informative difference emerges when we run SARSA with the same parameters as case 7, with the updated R matrix:

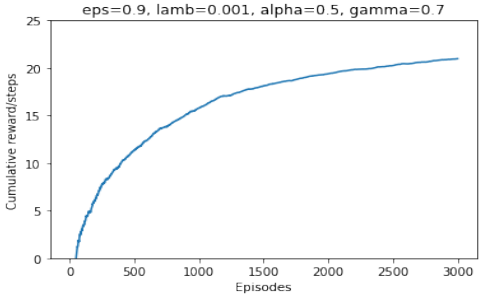

Figure 27: Average cumulative reward vs epsiode

```
0->6->11
1->11
2->10->11
3->11
4->11
5->11
6->11
7->11
8->11
9->11
10->11
11
```

Figure 28: Optimal paths learned

Note that the algorithm performs significantly worse, converging to a much lower value. By examining the paths learnt, we show that the agent has yet to learn the optimal paths, choosing to shoot on goal early and garner an immediate reward. This illustrates one key difference between

Q-learning and SARSA: as the latter accounts for penalties from exploratory moves, it is a far more conservative learning algorithm than Q-learning. It chooses to avoid the dangerous optimal path of passing the ball to the striker as there are negative rewards along the way.

This suggests that SARSA may be more applicable in certain scenarios, where mistakes are real and costly, such as a robot walking on a cliff, while Q-learning may be more appropriate for simulated scenarios where one can iterate quickly at no real cost.

In order to evaluate the performance of Q-learning against SARSA, we again implement early stopping and run 10 trials of each scenario. We perform the experiment for the base case (case 0) and the case with the updated R-matrix (case 7), holding the parameters constant and changing the algorithm used. The results are shown in the following table:

						Episodes to converge		Performance	
Case	Description	ε	λ	α	γ	Mean	Std	Mean	Std
0	Base case, Q-learning	0.9	0.001	0.5	0.7	1433.1	124.5	27.4	0.304
8	Base case, SARSA	0.9	0.001	0.5	0.7	1474.2	349.6	24.4	0.254
7	New R, Q-learning	0.9	0.001	0.5	0.7	1672.0	265.9	22.9	2.157
9	New R, SARSA	0.9	0.001	0.5	0.7	1573.1	209.1	18.7	0.779

Figure 29: Summary of results, Q-learning vs. SARSA

In both cases, SARSA takes roughly the same number of episodes to demonstrate convergence, but it fails to discover the optimal paths and consequently has a worse performance.

References

[1] Watkins, C.J. & Dayan, P. (1992). Q-learning, *Machine learning*, 8(3-4), pp.279-292.

[2] Sutton, R.S. & Barto, A.G. (2017). *Reinforcement learning: An introduction*, Cambridge: MIT Press.