INM426 Software Agents

Catherine Branter and Stephanie Jury

Introduction

In reinforcement learning an agent aims to learn an optimal policy or behavior that maximises the sum of available rewards. The agent is autonomous, making its own decisions about the policy to follow by estimating the long-term cumulative reward it will receive. Directly interacting with an unknown environment, the agent takes a "model-free" learning approach. In this paper, we employ the widely-used Q-learning algorithm introduced by Watkins (C. H. Watkins, 1989). In this algorithm the agent learns the optimal policy by sequentially updating the long-term values of state-action pairs until no further change is observed. Convergence is achieved as the algorithm minimises the temporal difference error over a number of iterations. Watkins proved that the values learned will always converge in cases where there are a finite number of actions and states.

1. Domain and Task (5%)

In our domain an agent tries to learn the optimal route from London to the Lake District based on available train routes. This is an adapted version of the well-known travelling salesman problem, however, in our environment the agent begins its journey in one location and terminates in another. The optimal route depends on the agent's preferences for each leg's duration, cost and number of platform changes. We explore different preferences in this paper, including the optimal route for an agent that prioritises lowest journey cost, one that prioritises shortest journey duration, and one which aims for the best combination of both.

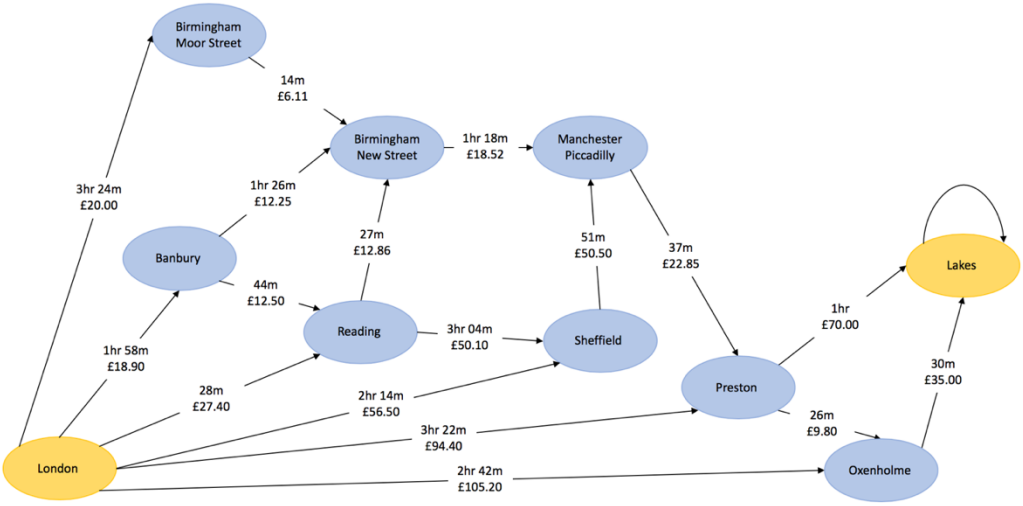

Figure 1: Graphical representation of the problem domain.

In a graphical representation of our environment the nodes represent train stations, which are the states available to the agent. Some states are connected by train lines, which each have an associated journey time and ticket cost that have been source from the National Rail Enquiries website ("National Rail," 2019). The environment is deterministic in the sense that the subset of available environment states is determined entirely by the current state, however the action the agent executes is stochastic. The environment is static as it does not change over time periods, and these time periods are discrete.

2. Environment Space (5%)

In our example, we have a discrete state space in which the agent takes actions to move in fixed steps from one state (train station) to another. The agent always commences its journey in London and the goal is for the agent to reach the Lake District via a train journey that maximises the reward, which is determined by the preferences set at the start.

2.1 State transition function

The state transition function tells the agent the new state it with reach, given its current state and the action it selects. In our train journey environment it is defined as $s_{t+1} = \delta(s_t , a_t)$. s_{t+1} denotes the state to which the agent moves from its

current state, s_t. The agent selects an action, a_t, being a member of A_t, the complete set of actions available from state, s_t. Here, s_{t+1} is a function of the current state, s_t and the action, a selected by the agent. Example actions and state transitions from London are shown in Table 1, and all available transitions from each state are shown in Table 2.

State (s)	Action (a)	New State (s_{t+1})
London	train to Banbury	Banbury
London	train to Birmingham Moor Street	Birmingham Moor Street
London	train to Reading	Reading
London	train to Sheffield	Sheffield
London	train to Preston	Preston
London	train to Oxenholme	Oxenholme

Table 1: State transitions available from London.

State (s)	Available New States
London	Banbury, Birmingham Moor Street, Reading, Sheffield, Preston, Oxenholme
Banbury	Birmingham New Street, Reading
Birmingham Moor Street	Birmingham New Street
Birmingham New Street	Manchester Piccadilly
Reading	Birmingham New Street, Sheffield
Sheffield	Manchester Piccadilly
Manchester	Preston
Preston	Oxenholme, Lakes
Oxenholme	Lakes
Lakes	Lakes

Table 2: Available transitions from each state.

2.2 Reward Function

In our Q-Learning environment the agent receives a reward during each time period, after making a train journey and transitioning to its new state. This reward is an equally weighted sum of the component rewards for the cost, duration, and platform changes involved in that part of the journey. These component rewards are displayed in Table 3. For duration and cost, lower values result in higher rewards. For platform changes, the agent receives a negative valued reward for each time period as it contributes to an additional part of the journey.

Using these component rewards, reward scores were assigned to each possible journey, as shown in Table 4. An additional reward of +150 was provided for the agent reaching the destination, and an additional reward of +10 was provided for the agent reaching a state which is only one additional transition to the destination.

Cost	Reward	Duration	Reward	Platform Change	Reward
GBP ≤ 0	0	mins ≤ 0	0	Yes	-10
0 < GBP ≤ 10	15	0 < mins ≤ 60	15	No	1
10 < GBP ≤ 30	14	60 < mins ≤ 120	10		
30 < GBP ≤ 40	13	120 < mins ≤ 180	5		
40 < GBP ≤ 60	12	180 < mins ≤ 240	1		
60 < GBP ≤ 80	10	240 < mins	-1		
80 < GBP ≤ 90	7				
90 < GBP ≤ 100	4				
100 < GBP ≤ 110	1				
GBP < 110	-1				

Table 3: Reward score components.

Source	Target	Source ID	Target ID	Duration (Minutes)	Cost (GBP)	Platform Change	Reward Score
London	Birmingham Moor Street	0	2	204	20.00	1	5
London	Banbury	0	1	118	18.90	1	14
London	Reading	0	4	28	27.40	1	19
London	Sheffield	0	5	134	56.50	1	7
London	Preston	0	7	202	89.40	1	-2
London	Oxenholme	0	8	102	105.20	1	1
Banbury	Birmingham New Street	1	3	86	12.86	1	14
Banbury	Reading	1	4	44	12.50	1	19
Birmingham Moor Street	Birmingham New Street	2	3	14	6.11	1	20
Birmingham New Street	Manchester Piccadilly	3	6	78	18.52	1	14
Reading	Birmingham New Street	4	3	27	12.86	1	19
Reading	Sheffield	4	5	184	50.10	1	3
Sheffield	Manchester Piccadilly	5	6	51	50.50	1	17
Manchester Piccadilly	Preston	6	7	37	22.85	1	19
Preston	Oxenholme	7	8	26	9.80	1	20
Preston	Lakes	7	9	60	70.00	0	25
Oxenholme	Lakes	8	9	30	35.00	0	28
Lakes	Lakes	9	9	0	0.00	0	

Table 4: Reward scores for each possible state transition.

2.3. Initialising the Reward Matrix and Graphical Representation

Having calculated the reward scores for each possible transition, an R matrix mapping current states and actions can be constructed. This is illustrated in Figure 2, with each action being the journey to the new state. This matrix provides the agent its reward for transitioning from any one state to another, with omitted values representing transitions which are not possible to take. These reward scores as applied to the domain diagram are shown in Figure 3.

Current State \ Actions	London	Banbury	Birmingham Moor St	Birmingham New St	Reading	Sheffield	Manchester Picadilly	Preston	Oxenholme	Lakes
London		14	5		19	7		8	11	
Banbury				14	19					
Birmingham Moor St				20						
Birmingham New St							14			
Reading				19		3				
Sheffield							17			
Manchester Picadilly								29		
Preston									30	175
Oxenholme										178
Lakes										

Figure 2: Reward matrix.

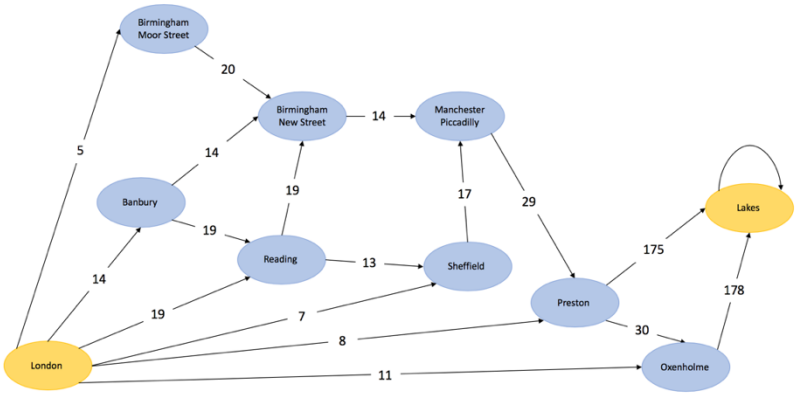

Figure 3: Graphical representation of transition reward scores.

3. Learning Policy

The learning policy, denoted $\pi(s_t) = a_t$, describes the way an agent acts. The policy is a function of a state and an action and returns the probability of taking a specific action given that state. In Q-learning, the goal is for the agent to learn the optimal policy, i.e. to maximise the return in each state. We defined the total number of epochs to be 300, meaning the agent runs through the problem from start to finish (London to Lake District) this many times in full.

Given the agent is in a model-free environment with no prior knowledge of it, the policy needs to involve some exploration, rather than simply using a greedy approach (ε=0) which would always exploit current knowledge in order to maximise the immediate reward. We also need to avoid always choosing randomly (ε=1). Therefore, our ε-greedy policy was selected in line with the findings from Barto and Sutton that "with noisier rewards, it takes more exploration to find the optimal action, and ε-greedy methods should fare even better relative to the greedy method"(Sutton & Barto, 2018). After initializing the ε value, a random number is generated in the range [0,1]. If that number is greater than or equal to ε, the agent exploits the environment by choosing the action with the highest valued reward, otherwise if the random number is less than ε, the agent will explore by selecting an action at random. We set our initial ε to be 1 to explore initially, and set the decay factor to 0.9, so that $\varepsilon_{t+1} = 0.9 \times \varepsilon_t$. The ε-decay policy should lead the agent to initially explore its environment and then progressively exploit its knowledge learned over the episodes.

4. Parameter Values for Q-Learning

We set our initial parameter values for Q-learning as shown in Table 5. These also serve as the default values for parameters during our grid search experiments.

Q-Learning Parameter	Value
γ	0.5
α	0.5
ε	1
π	ε-greedy
λ	0.90

Table 5: Parameter values for Q-learning.

The discount rate, (γ) determines the present value of future rewards and takes a value in the range [0,1]. When the discount rate is zero, the agent is "myopic" and seeks only to maximise its immediate return, R_{t+1} when determining the action to take. As the discount rate increases to one, the agent becomes increasingly "farsighted" and takes into consideration longer-term rewards when determining the action to take.

The learning rate or step size, (α) determines the extent to which newly acquired information overrides previously learned information, and takes a value in the range [0,1]. When the learning rate is zero, the agent does not update the Q-matrix with information learned in the current episode, instead retains only prior knowledge. When the learning rate is one, the environment becomes deterministic and the agent is sensitive to only the most recent information, completely overwriting prior knowledge with that learned in the most recent episode.

The learning policy, (π) as previously defined, is epsilon-greedy (ε-greedy). This policy takes an initial exploration factor, (ε) which is reduced by multiplication with decay factor, (λ) after every completed episode. As the number of elapsed episodes increases and the unexplored portion of the agent's environment reduces, it follows that the probability of further exploration should be reduced and the probability of reward exploitation should increase. ε equates to the probability of the agent selecting an action randomly from a uniformly distributed set of available actions, excluding that which provides the maximum reward state, and takes a value in the range [0,1]. (1-ε) equates to the probability of the agent selecting the action which will maximise its reward given its current state. When ε is set to one, the agent explores the state space indefinitely, and when set to zero, the agent exploits its known rewards.

5. Updating the Q-matrix in a Learning Episode

We make updates to our Q-matrix using the Bellman equation,

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

which specifies that the expected long-term reward given an action is equal to the immediate reward from the current action plus the expected reward, at a given point in time. This is obtained from the reward matrix. We assume that our agent has no initial knowledge, as such we start with an empty Q-matrix at t_0.

Current State \ Actions	London	Banbury	Birmingham Moor St	Birmingham New St	Reading	Sheffield	Manchester Picadilly	Preston	Oxenholme	Lakes
R-matrix										
London		14	5		19	7		8	11	
Banbury				14	19					
Birmingham Moor St				20						
Birmingham New St							14			
Reading				19		3				
Sheffield							17			
Manchester Picadilly								29		
Preston									30	175
Oxenholme										178
Lakes										

Current State \ Actions	London	Banbury	Birmingham Moor St	Birmingham New St	Reading	Sheffield	Manchester Picadilly	Preston	Oxenholme	Lakes
Q-matrix at t_0										
London	0	0	0	0	0	0	0	0	0	0
Banbury	0	0	0	0	0	0	0	0	0	0
Birmingham Moor St	0	0	0	0	0	0	0	0	0	0
Birmingham New St	0	0	0	0	0	0	0	0	0	0
Reading	0	0	0	0	0	0	0	0	0	0
Sheffield	0	0	0	0	0	0	0	0	0	0
Manchester Picadilly	0	0	0	0	0	0	0	0	0	0
Preston	0	0	0	0	0	0	0	0	0	0
Oxenholme	0	0	0	0	0	0	0	0	0	0
Lakes	0	0	0	0	0	0	0	0	0	0

1. Define the initial state, s_0.
   Example: s_0 = London

2. Define the set of possible actions, A_0, that can be taken from s_0.
   Example: A_0 = {Banbury, Birmingham Moor Street, Reading, Sheffield, Preston, Oxenholme}

3. Choose an action, a_0 from A_0, according to the ϵ-greedy policy.
   Example: a_0 = train to Preston

4. a_0 defines the next state, s_1.
   Example: s_1 = Preston

5. Define the set of possible actions, A_1, that can be taken from s_1.
   Example: A_1 = {Oxenholme, Lakes}

6. Update the Q-matrix at t_1 with the score of the selected state action pair, $Q(s_0, a_0)$.
   Example:

$$Q(London, Preston) \leftarrow Q(London, Preston) + \alpha\left[r(London, Peston) + \gamma \max[Q(Preston, Oxenholme), Q(Preston, Lakes)] - Q(London, Preston)\right]$$

$$Q(London, Preston) \leftarrow 0 + 1 \times \left[8 + 0.5 \times \max(0,0)\right] \leftarrow 8$$

Current State \ Actions	London	Banbury	Birmingham Moor St	Birmingham New St	Reading	Sheffield	Manchester Picadilly	Preston	Oxenholme	Lakes
London	0	0	0	0	0	0	0	8	0	0
Banbury	0	0	0	0	0	0	0	0	0	0
Birmingham Moor St	0	0	0	0	0	0	0	0	0	0
Birmingham New St	0	0	0	0	0	0	0	0	0	0
Reading	0	0	0	0	0	0	0	0	0	0
Sheffield	0	0	0	0	0	0	0	0	0	0
Manchester Picadilly	0	0	0	0	0	0	0	0	0	0
Preston	0	0	0	0	0	0	0	0	0	0
Oxenholme	0	0	0	0	0	0	0	0	0	0
Lakes	0	0	0	0	0	0	0	0	0	0

7. Choose an action, a_1 from A_1, according to the ϵ-greedy policy.
   Example: a_1 = train to Oxenholme

8. a_1 defines the next state, s_2.
   Example: s_2 = Oxenholme

9. Define the set of possible actions, A_2, that can be taken from s_2.
   Example: A_2 = {Lakes}

10. Update the Q matrix at t_2 with the score of the selected state action pair, $Q(s_1, a_1)$.
    Example:

$$Q(Preston, Oxenholme) \leftarrow Q(Preston, Oxenholme) + \alpha\left[r(Preston, Oxenholme) + \gamma \max[Q(Oxenholme, Lakes)] - Q(Preston, Oxenholme)\right]$$

$$Q(Preston, Oxenholme) \leftarrow 0 + 1 \times \left[30 + 0.5 \times \max(0)\right] \leftarrow 30$$

Current State \ Actions	London	Banbury	Birmingham Moor St	Birmingham New St	Reading	Sheffield	Manchester Picadilly	Preston	Oxenholme	Lakes
London	0	0	0	0	0	0	0	8	0	0
Banbury	0	0	0	0	0	0	0	0	0	0
Birmingham Moor St	0	0	0	0	0	0	0	0	0	0
Birmingham New St	0	0	0	0	0	0	0	0	0	0
Reading	0	0	0	0	0	0	0	0	0	0
Sheffield	0	0	0	0	0	0	0	0	0	0
Manchester Picadilly	0	0	0	0	0	0	0	0	0	0
Preston	0	0	0	0	0	0	0	30	0	0
Oxenholme	0	0	0	0	0	0	0	0	0	0
Lakes	0	0	0	0	0	0	0	0	0	0

11. Choose an action, a_2 from A_2, according to the ϵ-greedy policy.
    Example: a_2 = train to Lakes

12. a_2 defines the next state, s_3.
    Example: s_3 = Lakes

13. Define the set of possible actions, A_3, that can be taken from s_3.
    Example: A_3 = { }

14. Update the Q matrix at t_3 with the score of the selected state action pair, $Q(s_2, a_2)$.
    Example:

$$Q(Oxenholme, Lakes) \leftarrow Q(Oxenholme, Lakes) + \alpha\left[r(Oxenholme, Lakes) + \gamma \max[Q(OLakes, Lakes)] - Q(Oxenholme, Lakes)\right]$$

$$Q(Oxenholme, Lakes) \leftarrow 0 + 1 \times \left[178 + 0.5 \times \max(0,0)\right] \leftarrow 178$$

Current State \ Actions	London	Banbury	Birmingham Moor St	Birmingham New St	Reading	Sheffield	Manchester Picadilly	Preston	Oxenholme	Lakes
London	0	0	0	0	0	0	0	8	0	0
Banbury	0	0	0	0	0	0	0	0	0	0
Birmingham Moor St	0	0	0	0	0	0	0	0	0	0
Birmingham New St	0	0	0	0	0	0	0	0	0	0
Reading	0	0	0	0	0	0	0	0	0	0
Sheffield	0	0	0	0	0	0	0	0	0	0
Manchester Picadilly	0	0	0	0	0	0	0	0	0	0
Preston	0	0	0	0	0	0	0	30	0	0
Oxenholme	0	0	0	0	0	0	0	0	0	178
Lakes	0	0	0	0	0	0	0	0	0	0

6. Quantitative and Qualitative Analysis of Results

The agent's behaviour is determined by the parameters supplied to the Bellman equation, and the ε-greedy policy. We conducted several experiments sequentially varying these and analysing their effect on the speed at which the maximum mean episode Q score was reached, the number of episodes taken for the algorithm to converge, and the success with which the agent exploited the optimum policy. For each experiment we set all parameters to the default values described in Section 4. and varied the parameters to investigate as described below. Programmatically, we set a constant seed value, so the experiments yielded repeatable results and were suitable for direct comparison. We defined "convergence" as being the first episode after which the Q-matrix did not change for a further 15 consecutive episodes. To reduce the effect of random variation in our results, for each combination of tested parameters we ran each learning episode 50 times and recorded the average of the returned episode reward scores.

6.1. Varying Learning Rate (α)

We investigated learning rates of 0.00, 0.25, 0.50, 0.75 and 1.00, which were held constant over each episode. Values of 0.00 and 1.00 resulted in the agent failing to converge on an optimal policy within 300 episodes, as illustrated in Figure 2. When the learning rate was set to 0.00, the Q-matrix was never updated and effectively converged instantly on a Q-matrix of zeroes. When the learning rate was 1.00, the agent learned the state space extremely quickly, exploiting optimal policy effectively at the expense of exploring all of the Q-matrix. Figure 3 illustrates that changes in the learning rate had significant impact on the rate at which the agent maximised its mean reward. As larger learning rates were applied, the mean reward curve traced in the early episodes became incrementally steeper and more jagged. This reflects the greater speed with which the initially empty Q-matrix was overwritten with new information. With lower learning rates, the updates made to the Q-matrix were more gradual, resulting in a smoother, shallower learning curve. Accordingly, convergence was achieved much more quickly with higher learning rates.

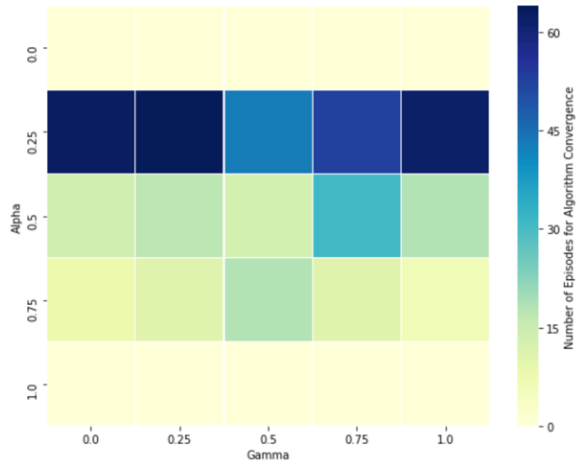

Figure 2: Heatmap of the number of episodes required for convergence on the optimal policy. The most was observed with α = 0.25 and γ = 0.25, at 64 episodes. The fewest was observed with α = 0.75 and γ = 0.25, at 17 episodes.

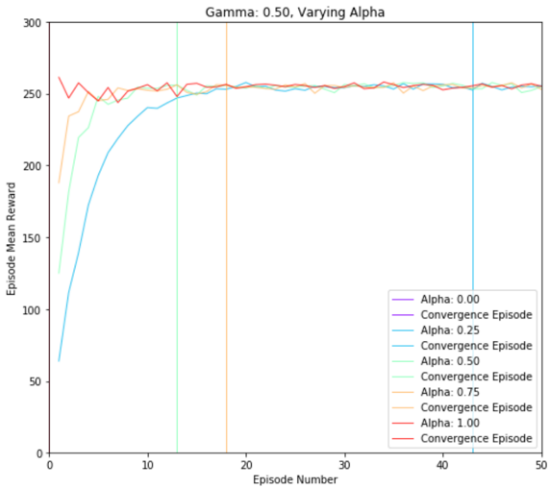

Figure 3: Learning curve with γ = 0.50 and α = 0.00, 0.25, 0.50, 0.75 and 1.00. No line is observed when α = 0.00 as the Q-matrix is never updated. It remains a matrix of zeroes, thus plots an episode mean reward which never rises above zero.

6.2. Varying Discount Rate (γ)

We investigated discount factors of 0.00, 0.25, 0.50, 0.75 and 1.00, which were held constant over each episode. We did not test discount factors greater than 1.00, to prevent the action values diverging. As illustrated in Figure 4, varying the value of gamma had little perceivable impact on the rate at which exploitation of the optimal policy was first achieved, with all curves exhibiting similar initial gradients. The number of episodes to convergence was not correlated with discount rate, contradicting our expectation that higher discount rates might result in sequentially faster convergence with more sophisticated agents that consider information about future rewards. We suspect this is due to the small number of possible transitions available to the agent in a single episode, meaning that there is little additional benefit conferred to an agent that is looking far ahead.

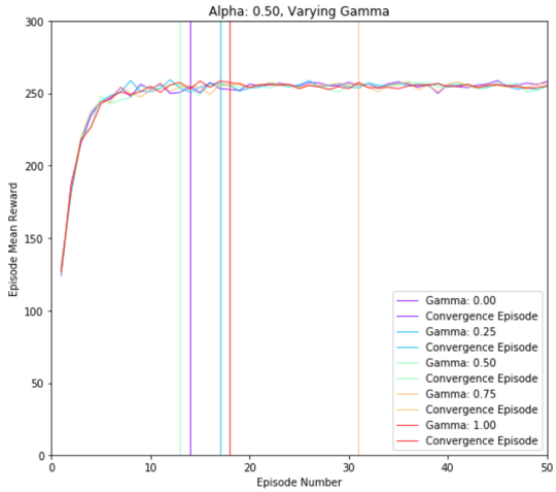

Figure 4: Learning curve with α = 0.50 and γ = 0.00, 0.25, 0.50, 0.75 and 1.00.

6.3. Varying Exploration Parameters (ε and λ)

We investigated epsilon values of 0.00, 0.25, 0.50, 0.75 and 1.00 and constant decay values, (λ), of 1.00, 0.975, 0.950, 0.925 and 0.900. The decay values were applied to each epsilon by multiplication at the end of each learning episode. Programmatically, a random number between zero and one was generated for each state transition and compared to the value of epsilon to determine whether the agent should explore or exploit in that episode.

In Figure 5 it is apparent that a random policy, with epsilon set to 1.00 and not reduced by a decay factor (decay factor set to 1.00), resulted in the fastest convergence, at only 12 episodes. This makes intuitive sense as the agent explores at every state transition, and the Q-matrix converges as soon as it has explored every possible complete journey.

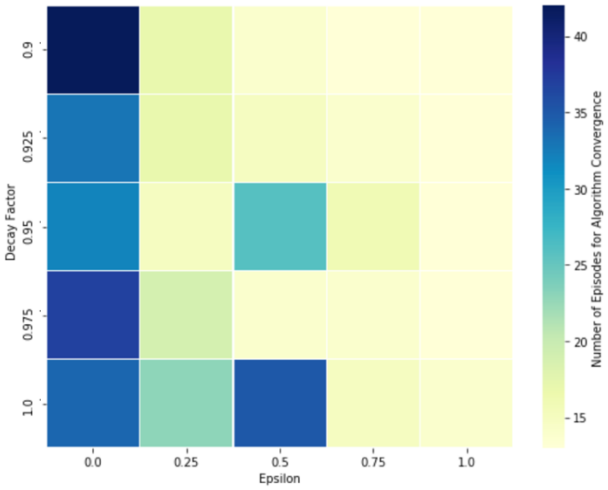

Figure 5: Heatmap of the number of episodes required for convergence on the optimal policy. The most was observed with ε = 0.00 and λ = 0.90, at 43 episodes. The fewest was observed with ε = 1.00 and λ = 1.00, at 12 episodes.

This fast exploration is reflected in Figure 6, where the steep gradient in episode mean reward was achieved because the agent did not stop exploring to exploit sub-optimal policies. Greedier epsilon policies, of 0.25 and below, resulted in the agent exploiting early on, thus requiring more episodes to learn the rewards for every state transition. The decay factor is not meaningful when epsilon is initialised at 0.00, but averaged over these five trials, convergence was achieved in 37 episodes. The slower exploration is reflected in Figure 7, with the shallower gradients in trials caused by the agent exploiting sub-optimal policies early on. While the initialised epsilon value affects the gradient of the episode mean reward curve, the size of the decay factor, i.e. the rate at which the agent becomes biased towards exploitation, determines the agent's ability to select the best learned policy consistently. In Figure 7, the least variance in episode mean reward after the initial learning phase, was achieved with the largest decay factor, 0.900. This was in line with our expectation that setting a low epsilon as quickly as possible after the initial learning phase will result in repeated optimal policy selection.

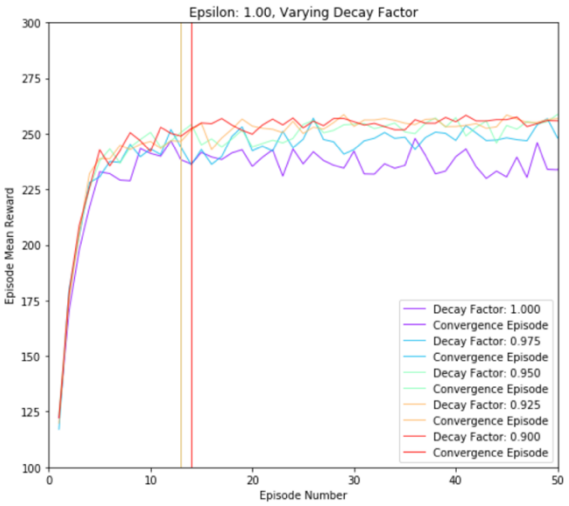

Figure 6: Learning curve with ε = 1.00 and λ = 1.00, 0.975, 0.950, 0.925 and 0.900.

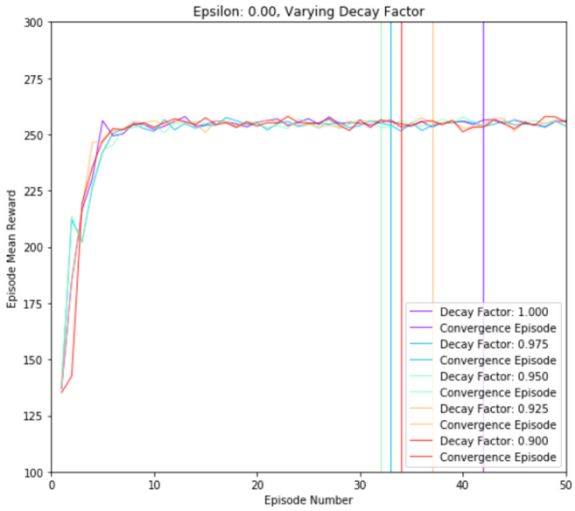

Figure 7: Learning curve with ε = 0.00 and λ = 1.00, 0.975, 0.950, 0.925 and 0.900.

6.4. Modifying the State Transition Diagram

To increase the complexity of the problem space, we modified the state transition diagram to allow the agent to travel back on itself, instead of only progressing towards the Lake District. We repeated the experiments which used an equally weighted combination of rewards obtained from journey cost and duration but, this time, imposing a penalty of -50 on the agent for each retrograde step. Once again, we initialised the algorithm with the default parameters described in Section 4. and repeated the grid searches over learning rate, discount rate, epsilon and decay factor.

6.4.1 Varying Learning Rate (α) and Discount Rate (γ)

Comparing equivalent learning curves between the original and more complex problem spaces, for example Figure 3 with Figure 8, and Figure 4 with Figure 9, it is clear that the agent requires more learning episodes to learn the optimal policy episode in the larger problem space. The shallower learning curves and larger number of episodes required to reach convergence are unsurprising as the agent has a greater number of state transitions to explore. Despite this, in many cases the agent is still capable of exploring the state space and selecting the optimal policy within 50 episodes.

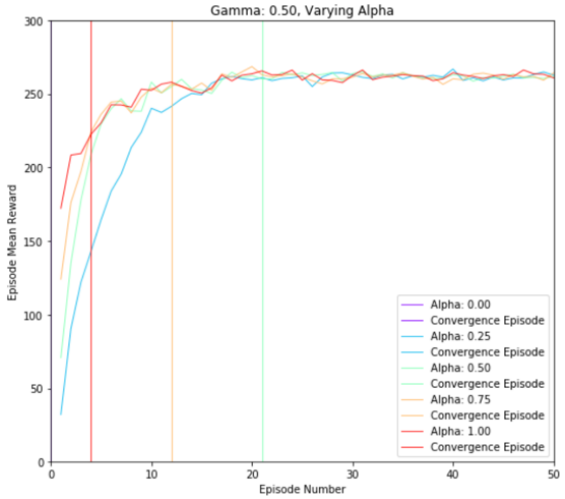

Figure 8: Learning curve with γ = 0.50 and α = 0.00, 0.25, 0.50, 0.75 and 1.00 in the larger problem space.

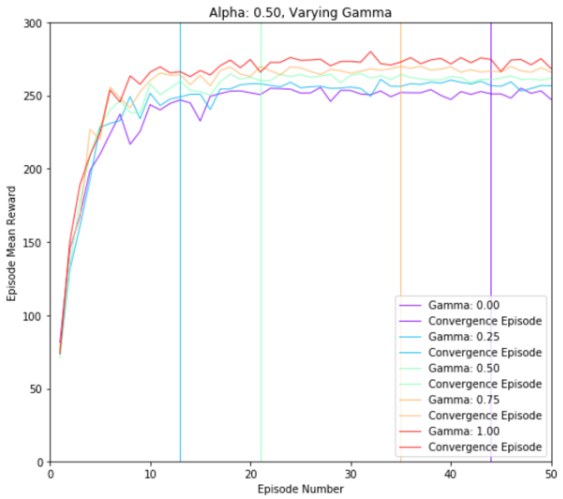

Figure 9: Learning curve with α = 0.50 and γ = 0.00, 0.25, 0.50, 0.75 and 1.00 in the larger problem space.

In the original problem space, the discount rate had little impact on the optimal policy learned by the agent, however, Figure 9 illustrates that in the larger problem space, as gamma was reduced from 1.00 to 0.00 and the agent became progressively less focused on seeking long-term rewards, convergence occurred at increasingly sub-optimal policies.

6.4.2 Varying Exploration Parameters (ε and λ)

In comparing the impact of the exploration parameters between the original problem space, Figure 6, and the larger problem space, Figure 9, the most notable difference occurs when using an epsilon of 1.00. Because the agent will incur penalties for transitioning in the direction of London, if it is conditions to explore constantly, the agent can achieve very large, negative episode mean reward scores as it continues to travel back and forth in the state space.

When varying the decay factor, a similar pattern is seen in the larger problem space as in the original, with more aggressive decay factors resulting in shallower learning curves. When the agent is set to explore indefinitely, with a decay factor of 1.00 applied to an epsilon of 1.00, it does not encounter the optimum policy within our limit of 5,000 episodes. As a consequently of the larger state space, the learning rate is much more sensitive to the exploration decay parameter and, in this case, λ = 0.900 achieves the fastest convergence on the optimal policy.

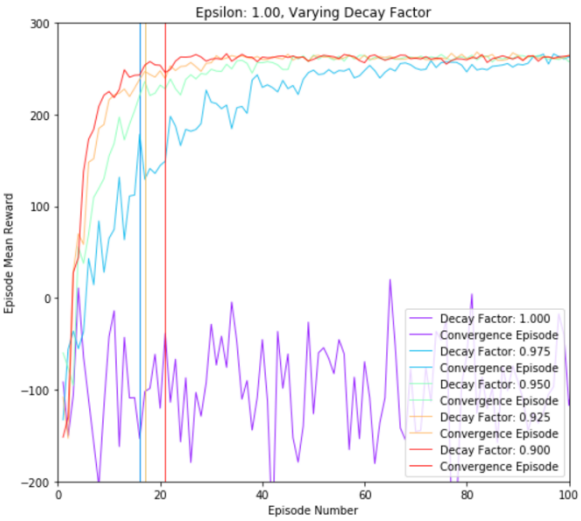

Figure 9: Learning curve with ε = 1.00 and λ = 1.00, 0.975, 0.950, 0.925 and 0.900 in the larger problem space.

7. Advanced Reinforcement Learning Techniques

The implementation of deep Q-learning has the potential to improve upon the performance of other reinforcement learning approaches and can be applied to domains beyond those which are "fully observed, low-dimensional state spaces", which has been noted as a limitation in the application of algorithms such as Q-learning (Mnih et al., 2015). The values and states within a reinforcement learning problem can be learned using a neural network and by incorporating deep learning, we no longer need to explicitly define a Q-matrix, meaning the agent is able to generalise to states it has not seen before and can facilitate the agent learning more rapidly, particularly in higher dimensional environments.

An additional problem with Q-learning is the the over-estimation problem of Q-values, which contributes to a bias in the model's learning due to the fact that the Q-learning algorithm approximates the maximum expected total reward by using the distribution of maximum total reward, rather than the actual distribution of total reward (Fox, Pakman, & Tishby, 2015). We anticipated that the performance of deep Q-learning would be unlikely to improve upon the results obtained through Q-learning for our specific problem, due to the limited size of the state space and implied lower likelihood that the overestimation problem is affecting our results. While neural networks can allow increased flexibility for reinforcement learning problems, they often do so at the expense of computational power and robustness (or stability) in results (Hasselt, Guez, & Silver, 2016).

An area we would like to explore in future would be the incorporation of multi-step temporal difference learning, in which we update the value of the Q-matrix using the rewards from more than one time period. Whereas Monte Carlo methods update based on an entire sequence of observed rewards, in temporal difference with 1-step, the value of the next state (one step away) is used as a proxy remaining rewards (Sutton & Tanner, 2005). An *n*-step approach could improve learning and reduce the bias of the temporal method as it looks further into the future than a 1-step approach by using an intermediate number of rewards.

8. Parallelisms Between the Q-Learning Algorithm and Error Correction Models in Psychology

Within psychology, the Rescorla-Wagner model presented a powerful means for modelling associative learning through its error-correcting learning rule (Rescorla & Wagner, 1972). The model is based on the notion that learning is driven by error and surprise, i.e. an animal learns when events violate its expectations (Niv, 2009). Expectations are built up when a significant event follows the presentation of a stimulus and these expectations are modified when subsequent events disagree with the learned expectation.

The Rescorla-Wagner model built upon previous approaches, originating from classical conditioning theory developed by Pavlov in which “the magnitude and timing of the conditioned response changes as a result of the contingency between the conditioned stimulus and the unconditioned stimulus” (Pavlov, 1927). The model introduced a global error term in which the change in prediction value of the conditioned response is proportional to the difference between the actual and predicted outcome. This enabled to model to assume that the conditioned stimulus could be predicted by summing the associative strength of all of the conditioning stimuli. The model also provided an explanation for blocking, and overshadowing, and predicted the concept of over-expectation (Rescorla & Wagner, 1972).

Q-learning was first introduced by Watkins in 1989 (C. J. C. H. Watkins, 1989). In Q-learning, agents learn the optimal way to act (policy) in Markovian settings, by interacting with their environment and determining the long-term values of state-action pairs (J. C. H. Watkins & Dayan, 1992). The algorithm aims to converge by minimising the temporal difference error over a number of iterations. Temporal difference is a learning algorithm introduced by Barton and Sutton and builds upon the Rescorla-Wagner model. It explains the same phenomena as the Rescorla-Wagner model, but real-time prediction takes place at each time step, rather than only at the end of a trial (Sutton & Barto, 1990). In Equations 1 and 2, we can see that the time component, *t*, present in the temporal difference model is not present in Rescorla-Wagner. This implies that the Rescorla-Wagner model cannot discriminate on the basis of a temporal structure, as can temporal difference and Q-learning models. Conversely, the Markov decision process assumed in Q-learning contributes to the lack of generalisability of the Q-learning computational approach when compared to Rescorla-Wagner.

$$\Delta V_i^n = \alpha_i \cdot \beta_{US} \left(\lambda^n - \sum_{i=A}^{Z} V_i^{n-1} \cdot X_i^n\right)$$

Equation 1: Rescorla-Wagner equation for the change in the predictive value of a stimulus.

$$\partial v^t{}_i = \alpha_i \cdot \left[\lambda^t - \left(\sum_i \sum_j \left(X_{i,j}{}^{t-1} \cdot v_{i,j}^t\right) - \gamma \sum_i \sum_j \left(X_{i,j}{}^t \cdot v_{i,j}^t\right)\right)\right]$$

Equation 2: Temporal difference predictions and learning rule.

An important parallel to be drawn between the two models is the relevance of the global error term within each of them. In temporal, the actual outcome, λ, is compared to the total change in the prediction outcome between the two most recent predictions, whereas in Rescorla-Wagner the actual outcome is compared to the total outcome prediction on a given trial, as seen respectively in Equations 3 and 4.

$$\Delta V_i^n = \alpha_i \cdot \beta_o \cdot (\lambda^n - P_o^n)$$

Equation 4: Rescorla-Wagner learning rule.

$$\partial v^t{}_i = \alpha_i \cdot \beta \cdot [\lambda^t - (P_o^{t-1} - \gamma P_o^t)]$$

Equation 5: Temporal difference learning rule.

An additional commonality of Q-learning and error correction models such as Rescorla-Wagner, is the presence of eligibility traces. In temporal difference models, component eligibility traces are sequentially activated in time and their activation slowly decays, which is reminiscent of the concept of memory tracing in psychology (as memories decay over time). They are regulated by the discount factor and activation decay. The discount factor, γ, in the temporal difference error of Q-learning takes a value in the range [0,1] and explains the value the agent places on future versus current rewards. Similarly, in Rescorla-Wagner, predictors closer to the unconditioned stimulus are given more weight as they are considered to be more informative. Parallels can also be drawn between the rewards associated with state-action pairs in Q-learning, and the associations developed between unconditioned and conditioned stimuli in associative learning. The rewards act as a positive reinforcement of certain behaviour within Q-learning, which can lead to learning of the optimal policy, in a similar way to how an animal learns to associate an unconditioned and conditioned stimulus because of an association made over time, due to surprise.

In Q-learning the optimal policy depends upon the structure of the rewards i.e. the behaviour the agent learns is dependent on the rewards available for state-action pairs, however in the Rescorla-Wagner model, rewards and outcomes are regarded as one and the same, unlike in Q-learning where they are distinguished between (Alonso, Mondragón, & Kjall-Ohlsson, n.d.). Another significant difference is the lack of generalisability for Q-learning due to the Markov decision process condition, which is not present in the Rescorla-Wagner model.

10. Error Correction Models as Reinforcement Learning Architectures

At present, drawbacks of using Q-learning to model reinforcement learning problems include the computational framework's inability to take on high-dimensional environments, as well as the over-estimation problem of Q-values, and the lack of generalisability when the temporal component of classical conditioning is included in computational models (Fox et al., 2015), (Luzardo, 2018). Limitations of reinforcement learning computational methods more broadly, when compared to human performance in learning, include the need for large amounts of training data and the restrictions on applying learned skills into different domains or environments (Wang et al., 2016).

By incorporating error correction models into reinforcement learning algorithms, we could develop a computational approach that more closely mimics the process by which humans and animals efficiently derive meaning from their environments and use it to apply their learnt behaviour to novel situations (Mnih et al., 2015). The success of incorporating the prediction error term from the Rescorla-Wagner model into other models for understanding reinforcement learning processes was evident when used in the context of dynamic causal modelling. den Ouden et al. established that associations are learned, regardless of whether they are related to the task at hand, and neurological changes were seen to co-occur with prediction error changes (den Ouden, Friston, Daw, McIntosh, & Stephan, 2008). Additionally, progress has also been made with regards to explaining the lack of generalisability of Q-learning results by combining the Rescorla-Wagner model with Q-learning (Alonso et al., n.d.). In real-life applications of reinforcement learning, such as in the field of robotics or for specific tasks such as traffic light control, a function approximator is needed for Q-learning to be implemented (Levine, Finn, Darrell, & Abbeel, 2015), (Arel, Liu, Urbanik, & Kohls, 2010).

Within the field of deep learning, a convolutional neural network (CNN) is a type of neural network which consists of convolution and subsampling steps and a single hidden layer for classification using backpropagation, as displayed in Figure 10.

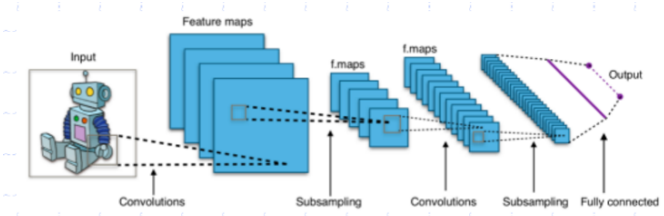

Figure 10: Graphical representation of a CNN (Gulli & Pal, 2017)

CNNs were inspired by connectivity patterns between neurons in the visual cortex, and are recognised for their success at image and video recognition (Krizhevsky, Sutskever, & Hinton, 2012.) During learning, CNNs discover features, which can be thought of as the state space, and these extracted features are fed into the lower layers of the network for classification. Given the successful efforts of researchers to unite computational models with the neurological systems, such as the prediction error hypothesis of dopamine and the existence of a decision making mechanism in the brain, it follows that the combination of these approaches with a model such as a CNN could provide a realistic representation of decision making through prediction error (Niv, 2009). The output of error correction models, in which a stimulus predicts an outcome, could be incorporated into the classification process as a synthetic feature and supplement the model's learning, and the error correction rule from Rescorla-Wagner could be used to train the weights for classification within a CNN. Backpropagation is the process by which the weights in an artificial neural network are updated based on the error term and here, the global delta error term from an error correction model could be incorporated. It has already been proven that reinforcement learning combined with neural networks can help relieve the human problem of having to create specific features related to the task at hand, and their use cases have expanded in recent years (Barto, Thomas, & Sutton, n.d.).

Additionally, the output of error correction models could be used in Long-Short Term Memory (LSTM) neural networks. LSTMs are a type of recurrent neural network (RNN) with feedback connections, and they have proved particularly useful in processing both images and sequential data, such as videos or word recognition (Li & Wu, 2014). Researchers have already drawn some interesting parallels between computational models for understanding classical conditioning and RNNs, through recurrence and eligibility traces (Kokkola, 2017). Ultimately, incorporating a prediction error term could help with accelerating the rate at which reinforcement methods adapt to new tasks, through providing an additional feedback loop. The output of the error correction models such as Rescorla-Wagner could be therefore be applied to enhance our understanding of many wide-ranging practical applications of reinforcement learning, and for applying an agent’s learned behaviour to novel environments.

Bibliography

Alonso, E., Mondragón, E., & Kjall-Ohlsson, N. (n.d.). Pavlovian and Instrumental Q-Learning: A Rescorla-Wagner-based approach to generalization in Q-learning. Retrieved April 2, 2019, from https://cal-r.org/mondragon/home/Papers/AISB-1.pdf

Arel, I., Liu, C., Urbanik, T., & Kohls, A. G. (2010). Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems*, *4*(2), 128. https://doi.org/10.1049/iet-its.2009.0070

Barto, A. G., Thomas, P. S., & Sutton, R. S. (n.d.). *Some Recent Applications of Reinforcement Learning*. 6.

den Ouden, H. E. M., Friston, K. J., Daw, N. D., McIntosh, A. R., & Stephan, K. E. (2008). Dual Role for Prediction Error in Associative Learning | Cerebral Cortex | Oxford Academic. Retrieved April 2, 2019, from https://academic.oup.com/cercor/article/19/5/1175/301790

Fox, R., Pakman, A., & Tishby, N. (2015). Taming the Noise in Reinforcement Learning via Soft Updates. *ArXiv:1512.08562 [Cs, Math]*. Retrieved from http://arxiv.org/abs/1512.08562

Hasselt, H. van, Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. *Thirtieth AAAI Conference on Artificial Intelligence*. Presented at the Thirtieth AAAI Conference on Artificial Intelligence. Retrieved from https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389

Kokkola, N. (2017). *A Double-Error Correction Computational Model of Learning. (Unpublished Doctoral thesis, City, University of London)*. 279.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 25* (pp. 1097–1105). Retrieved from http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2015). End-to-End Training of Deep Visuomotor Policies. *ArXiv:1504.00702 [Cs]*. Retrieved from http://arxiv.org/abs/1504.00702

Li, X., & Wu, X. (2014). Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition. *ArXiv:1410.4281 [Cs]*. Retrieved from http://arxiv.org/abs/1410.4281

Luzardo, A. (2018). The Rescorla-Wagner Drift-Diffusion Model. *Doctoral Thesis, City, University of London*, 156.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., … Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

National Rail. (2019). Retrieved April 3, 2019, from http://www.nationalrail.co.uk/

Niv, Y. (2009). *Reinforcement learning in the brain*. 38.

Pavlov, I. P. (1927). *Conditional reflexes: an investigation of the physiological activity of the cerebral cortex.* Oxford, England: Oxford Univ. Press.

Rescorla, R. A., & Wagner, A. R. (1972). *A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. Classical Conditioning II Current Research and Theory. Ed. by A H Black and W F Prokasy*.

Sutton, R. S., & Barto, A. G. (1990). *Time-Derivative Models of Pavlovian Reinforcement*. 41.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction* (Second edition). In *Adaptive Computation and Machine Learning Series* (Second edition). Cambridge, Massachusetts: The MIT Press.

Sutton, R. S., & Tanner, B. (2005). Temporal-Difference Networks. In L. K. Saul, Y. Weiss, & L. Bottou (Eds.), *Advances in Neural Information Processing Systems 17* (pp. 1377–1384). Retrieved from http://papers.nips.cc/paper/2545-temporal-difference-networks.pdf

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., … Botvinick, M. (2016). Learning to reinforcement learn. *ArXiv:1611.05763 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1611.05763

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards, Ph.D. thesis*. Retrieved from http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

Watkins, J. C. H., & Dayan, P. (1992). *Machine Learning*. Retrieved from http://www.gatsby.ucl.ac.uk/~Dayan/papers/cjch.pdf