MHBF Reinforcement Learning Project:

Alexander Moore & Johannes Rieke

Our project required us to use Reinforcement learning, specifically the SARSA algorithm, to perform a spatial navigation task. Our environment took the shape of a T maze, with a pickup area and a target area at the end of two branches. The agent received a large positive reward if it entered the target area after first entering the pick up area, it also received a small negative reward every time it hit a wall. The agent began each run completely naive, having to learn about the values of certain state action pairs via exploration and later on exploit more efficient paths to a reward.

Neural Network:

To implement the above we created a two dimensional neural network. Our input layer received and encoded information about the location of the agent, where as the output layer played a vital role in action selection.

The cells in the input layer were designed to mimic place cells, which have been extensively researched in the rodent medial temporal lobe. Equally distributed as a grid covering the entire maze each cell had a gaussian activity profile, with width 5cm. Each place cell would be maximally activated if the agent was stood exactly on its centre, the activity decays as the agent moves further away from the centre. Fig 1 shows the distribution of place cells and an example of their activity for a given agent location. As the agent is equidistant between four place cells they all display equal activity, some activity is observed in the next four closest cells, while the rest remain at baseline. The number of place cells activated for a given location is dependent on the sigma parameter, wider gaussians are sensitive to locations that are further away. As can be seen in Fig 1, with a value of sigma= 5cm the place cells have overlapping gaussians; as a result, each location is encoded by the activity of multiple cells in the input layer. This allows for a robust and continuous representation of space. The input layer also encodes whether or not the agent has entered the pickup area, via switching between two identical populations of neurons.

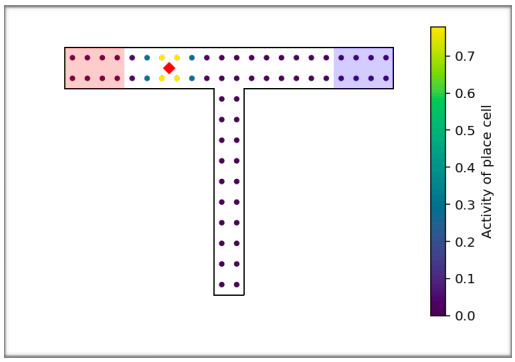


Fig 1: Example showing input layer activity for a given agent position (red diamond)

The input and output layers are connected by a series of synapses whose weights can be updated. The number of cells in the output layer reflects the number of actions available to the agent (initially just the four cardinal directions although this is modified during more complex simulations). The Q value of a given state action pair is reflected in the activity of the output-neuron associated with the action. As the weights connecting the two layers are modified the output-layer activation produced by a given state will also change; effectively allowing the agent to update the valuation of an action in a given state. The distance the agent moves in a given step is drawn randomly from a gaussian (mean 3 cm).

The epsilon parameter regulates Exploration vs Exploitation. The agent will exploit the action associated with the highest Q value with a probability of 1 - epsilon, otherwise it will explore a random action. A high value of epsilon results in the agent exploring more, this is useful when the environment is novel and the rewards associated with each state are unclear. On the other hand a lower value of epsilon encourages the agent to exploit the action associated with the highest Q value, this policy makes more sense once the environment is well known. For our initial simulations our value of epsilon decays from 0.5 to 0.05 over the course of a run. Later on we also simulate with different values of epsilon to analyse what effect they have on performance.

Learning:

We used the state action reward state action (SARSA) learning algorithm to update the weights in our neural network. SARSA requires a Markovian decision process where the next state the agent will enter is dependent on the current state (s) and action (a); however, given s and a the next state is conditionally independent of all other previous states and actions. The key algorithm is summarised below:

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

Here the Q value of a given state action pair is updated if the reward received at time t is unexpected this plus discounted future rewards (Q values from the next time step) are compared to the previous estimated Q value. The effect this update has on the new Q value is determined by a learning rate alpha. Another crucial parameter is the decay rate gamma. This determines the importance of future rewards, with low values of gamma forcing the agent to pursue immediate rewards. We will initially choose a decay rate of 0.5, however, later simulations will explore the effect of different values of gamma on performance.

Results:

We initially ran simulations with the default parameters as outlined above. We averaged performance over 10 naive agents and have plotted mean Q value and average escape latency (number of steps required to solve the task) against trial number in figure 2.

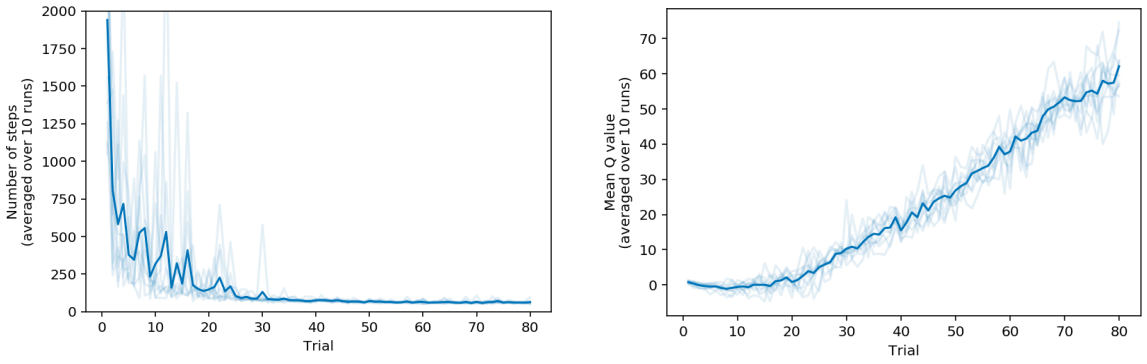


Fig 2: Escape latency and Mean Q value vs Trial number

It is unsurprising that the average escape latency decreases with number of trials, as the agent learns more about its environment you would expect it to perform the task quicker. This improved performance is undoubtably linked to the second plot, where the average Q value increases over trials. Initially all Q values are set to 0 and the agent has no clear path to follow, however, gradually as it explores the maze more and eventually encounters the positive reward Q values begin to increase thus guiding the agent to completing the task more quickly.

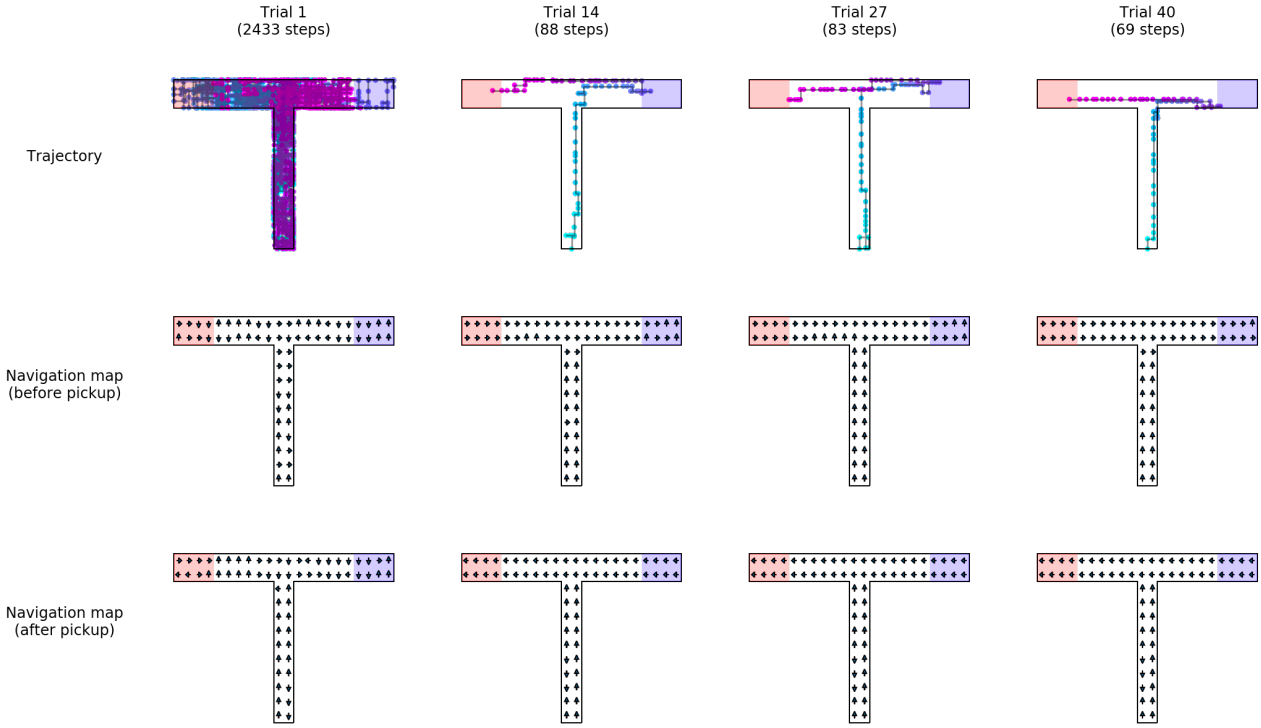
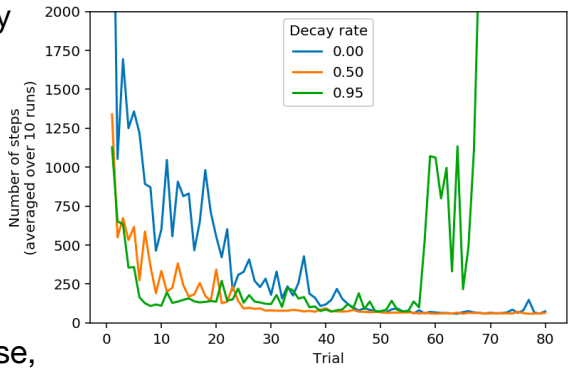


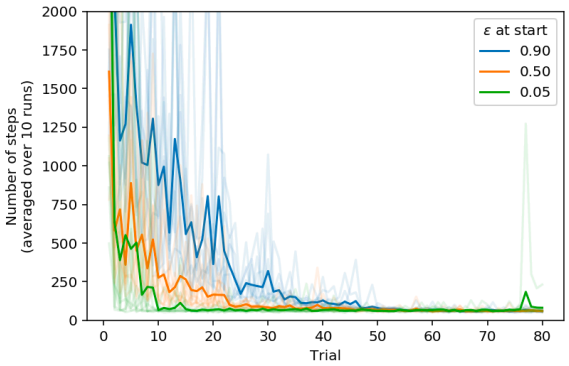
Fig 3: Trajectories and Navigation maps for agents after different number of trials

Further insight into how the agents behaviour changes over time as it learns more can be gained from Fig 3. The trajectory maps plot the agent's movement through the maze. As can be seen in Trial 1 the agent's movement is very erratic it regularly hits the walls and explores most of the maze. After 11 trials the agent is already significantly more direct (its escape latency is over 2000 steps shorter) as the most steep learning curve can be observed during the first few trials. The agent still hits the wall fairly frequently and therefore encounters negative rewards. Over the course of the next twenty trials (the last two plots) the path the agent takes improves significantly, it stops hitting walls and takes a much more direct route to first the pick up area and then the target area. It is important to note that even during trial 31 the agent still moves somewhat randomly as the epsilon parameter does not reach its final value (0.05) until trial 60. Navigation maps are plotted below the trajectories; they show the optimal direction of movement for a given location. They reinforce the concept that the over time the direction the agent should move in becomes more clear.

The first parameter that we varied was the decay rate, the results of this can be seen in the figure. An eligibility trace provides the system with a short term memory for many previous input signals so that their valuation can also be updated. For a decay rate=0 the system only considers immediate rewards as a result its takes a lot longer to converge to a reasonable escape latency. Information about reward can only backpropagate one state per trial in this case, therefore it is unsurprising that it takes a long time to converge. On the other hand for agents with a decay rate= 0.95 can take into account many more state action pairs that lead to a reward, as a result the average escape latency decreases much faster. However, for some trials with a high value of gamma the escape latency will diverge from the optimum after a number of trials (as can be seen in the figure after 55 trials).



We also varied initial epsilon values, the parameter that determines the balance of exploitation vs exploration. For all runs the value of epsilon decayed overtime. This makes sense as when the agent first enters the maze it should explore more to learn about its environment, later on when it has learnt about the valuation of state action pairs it makes more sense to exploit policies that are known to be rewarding. As can be seen in the figure, very high initial values of epsilon take longer to converge as the agent continues to explore the maze unnecessarily. On the other hand agents with low initial values of epsilon appear to converge faster, however, they can waste large amount of times if they get stuck in a part of the maze that was not previously explored. This can be seen in the figure when the green trace shows a peak around trial 75. On balance it makes most sense to chose an intermediate value of epsilon.



Finally we also varied the number of action the agent could take. The figure clearly shows that the greater the number of actions the slower the learning. Considering the agent learns state action pairs and the weight matrix grows when it there are more available actions this result is intuitive