Software Agents

INM 426 Coursework

Daniel Dixey Daniel.Dixey@city.ac.uk

Enrico Lopedoto Enrico.Lopedoto@city.ac.uk

April 12th, 2016

**Abstract**

The aim of this work is to give the reader a concrete example of a classical reinforcement learning application in terms of transition probabilities and reward functions. The domain of application will be the contest of a randomly generate environment according to specific function and the learning task will provide the agent a set of information enabling him to survive in the current environment the longer.

The environment will provide to the agent the possibility to move in three different directions taking five possible actions and the survival function will be defined as that function allowing the agent to skip obstacle generated in the environment.

Proposed algorithms for this task are Monte Carlo, Epsilon Decay and Neural networks. Measurements of this task are the time to completion in relation to final position along the track and percentage of completion of the track.

A comparison of different parameters will be performed and criticized.

Contents

**1. Introduction**

1.1 Domain Representation and task

1.2 Matrix representation and path Generation

1.3 Helicopter Task

**2. Problem Domain and Graph Representation**

2.1 Q Matrix

2.2 R Matrix

**3. Q Learning Functions**

3.1 – Greedy Monte Carlo

3.2 Epsilon Decay

3.3 Recurrent Neural Network

**4. Variations and analysis**

4.1 Altered alpha values

4.2 Altered learning rates

4.3 Altered state transitions

4.4 Altered reward functions

**5. Analysis**

**6. Conclusion**

References

1. **Introduction**

The nature of learning gives a clear shape to the idea of environment and how this influences someone having a wealth of information about cause and effect and how this one can learn and predict a possible consequence of an action given a current state. Learning can be also synonym of tools available in order to achieve a goal. Reinforcement learning refers to how someone can map situations and action when involved in some tasks in an unknown or unseen environment. In the classical example the learner is not aware at the beginning of which actions to undertake but has to learn in order to achieve a goal, a reward, and at a cost of some penalty.

In this contest, we will define our environment in which the agent will learn and will allow to our reference entity to survive the longer along the environment. A series of Q-Learning algorithm will be deployed in order to retain in his Long Short Term memory a basket of state, action tuples, each mapped with a reward or penalty. The application will automatically evolve in order to survive the most time or complete the track, as our utility function has been defined.

Technically, the software use to develop this application is Python and the visualization and interaction has been made possible with the usage of Matplot library.

* 1. **Domain representation and task**

The domain of application is state space environment in which our entity, the “helicopter” will try to self-drive in order to live as longer as possible in the randomly generated path. The environment is generate as a matrix of variable size in both its coordinates of length and width.The entity task is to complete the matrix starting from the left side and try to achieve the right side without interruption in between, represented by the randomly generate obstacles. We refer to this task as the example of a helicopter that has to maintain certain minimum and maximum quotes corresponding to the generate obstacles. In order to add complexity to the model a wind model has been generated.

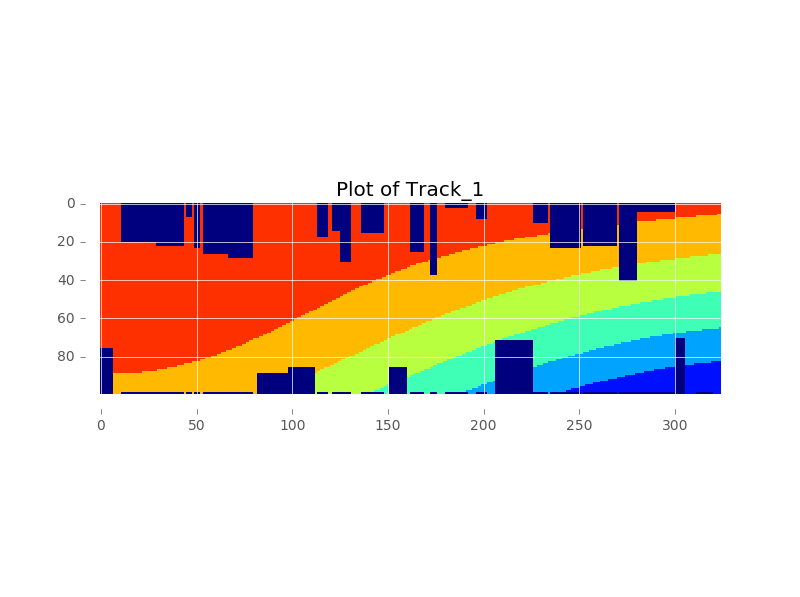
* 1. **Matrix Notation and Path generation**

A random number of matrix of zero has been created and in each a random number of parameters, subjects to certain maximum and minimum values constraints, has been drawn in order to create the obstacles as a sub matrix of ones. The final matrix can be viewed as a grid of function that are either the obstacle of the wind gradient, defined as follows

In order to generate the sub sample matrix of ones to overlap the 0-matrix of arbitrary dimensions, an index has been created and has been used to generate the side of the matrix between the upper and lower bounds. To do so, a random number has been generated and we kept the cutoff threshold of 0.5. Any number above the threshold would have had the effect to generate the sub matrix in the upper bound and vice versa if the generate number was below the threshold. The extra complexity has been gained with the wind function expressed as . The first generation is expressed as

Where and they represents respectively a random component generated only once and referring to the mixture of one of the two part of the model, either additive or subtractive and a marginal random component which varies at each time the model is generated and refers to specific coordinates of the model. The most recent version allowed the wind model to generate sparser gradient and can be expressed in the form of

Where this time again is expressed as random component relative to time and specific coordinates and as fixed factor which we chose being equal to 10. Below the representation of the two models respectively and the different gradient which provides different reward to the agent



* 1. **Helicopter Task**

The helicopter task can be described as a survival task in which the longer the track completed the better is the performance, however the maximum reward is gained only at the end of the path. the states is following are represented by the different gradient colours in the part of the track which is not an obstacle and the wind function guarantees a state which ranges from 0 to 6 according to which the helicopter is taking an action. Obstacles are initially set to be -10 and final reward to 100. The representation of the best route is the one by which the helicopters follows the most the wind and reaches the end of the track without interruptions.

Possible actions it can takes are either up or down but the new position will be changed of different index increment according to the different action. This index can range from 0 to 2 and can be applied to both ways, hence up or down. The new state is the one that forces the helicopter to have his position updated of one unitary increment to his right, forcing it to move toward the end of the track. According to the actions that the agent can take in the wind environment, these represents a slight change of the above descripted action list, in which the x-coordinate index can vary of either 1 or 2, according to the relative wind state.

1. **Problem Domain Graph Representation**

The classical definition of reinforcement learning allows us to think in a domain of cause and effect and relate this loop to the state action pair which is common to refer to while talking about software agent tasks. In this paper, the problem is represented by a numerical factor, easy to read by the agent. This coincides with the sum of different matrixes, representing each a layer of states: the first is the free space in which the agent can supervise the helicopter and the other is the range of states generated by the supplementary function of the wind.

The simple notation used will allows us to make a generalization of what happens in reality when flight policies applies to different aerospaces. It might be imposed to the pilot not to flight above or below a given quote because of risk of collision (generally applicable for the obstacles generate on the lower part of the matrix) or because other aero mobiles are in transit (this is more the case of the obstacles generated in the upper part of the matrix.

If the field of application is a concrete case, the domain for reinforcement learning is suitable given the tasks and limits the agent has in order to drive the helicopter toward the end of the track. The different states can be represented as a different wind regime an obstacle or one of either the two states of start and end. Visually can be represented with graph theory as follows:

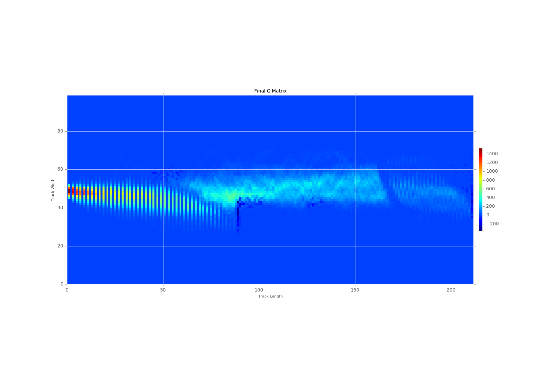
* 1. **Q Matrix**

The state in our model is consider based on the 'field of view' of the helicopter. The agent is able to create a state space of up to 3 squares in front and +/- 2 square below its current position so having 15 available future position to point. It can also take any value in the range of -1 to 8 where -1 indicates obstacles and the alternative values in the wind strength and direction. The final value is the 'height' of the Helicopter in the world and can take a value dependent on the height of the track.

The calculation shown below demonstrates how to calculate the number of possible states in our state space. Calculating all the possible permutations will lead to having (9\*\*15) \* 99 = 20,383,222,077,370,251 possibilities - where 99 is the height of the track. Given the flexible nature of the environment parameters and of the random nature of the wind model underneath the different states, it is more useful to parameterize the q value numbers, given by:

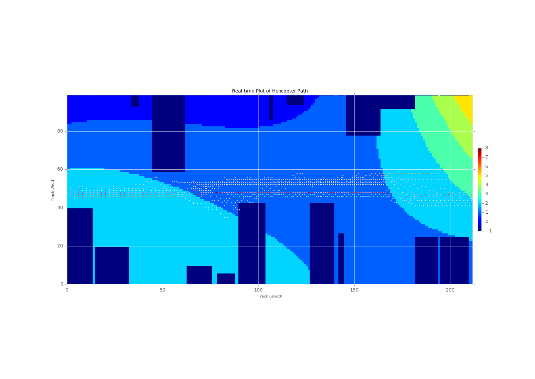
Where are the possible states, is the track height and and are respectively the depth of view of the agent and the width of its view, in our case these two parameters are 3 and 5 respectively.

Implementation in our model all possible states are not pre-determined prior to starting the learning process. The method used makes use of a key-value store so that when a new state is observed its corresponding value is stored. Computationally this saves a significant amount of time and memory and the primary reason for this is that the model may never experience a specific state and therefore there would have been no need to store all of this information. The stored values are represented in the following Q Matrix in which all the value observed are those stored in the dictionary and are those that actually are used to plot the gradient matrix of cumulative Q values in relation to the helicopter live state.



* 1. **R Matrix**

The reward function defines the reinforcement learning problem’s goal and it maps each perceived state of the environment to a reward which indicates the intrinsic appeal of each state. As single goal of the agent’s task is to maximize the total reward that it receives in the long run. Differently from evolutionary algorithms, reinforcement learning involves learning while interacting with environment. Our reward matrix maps these rewards as we described above and as we proposed here, a representation for the selected track:

****

1. **Q learning functions**

All the models discussed in the coursework will use an off policy strategy – an epsilon greedy which intends to ensure adequate exploration of all the state space. To exploit differences in result of this self-driving helicopter simulation we will compare three methods in which a different policy will be applied in order to select the best actions. These methods are the Monte Carlo, the Epsilon Decay and the Recurrent Neural Networks.

The Q Learning variables are represented below:

*Q(s, a) = Q value of a given state and action*

*a = Action.*

*s = State.*

*r = Reward.*

*= Maximum reward for action a.*

**Parameters**

*α = Alpha Value – Default value of 0.7*

We can refer to this as the learning rate which usually ranges between 0 and 1 and is responsible of the pace at which the learning task takes place or alternatively indicates how quickly the new values replaces the old.

The trade-off between this range of values is that when this parameter is set to be 0 then there is no real update as the agent only considers initial information whereas when is set to 1 the agent considers the most recent information. In this case the computational trade off arise because it may be quicker to retrieve but may not be accurate. In our task, finding shortest path does not necessarily provide the best solution and the completion of the track because obstacles in between might provide misleading information.

*γ = Gamma – Default value of 0.8*

This is the discount factor and is also ranging between 0 and 1 and determines the importance of future rewards. Values of 0 would only let the agent consider immediate rewards and not longer rewards and provides a myopic view of the agent. The contrary is true when is set to 1.

**Policy Variables**

*ε = Epsilon – Default value of 0.3*

All our algorithms do have a greedy policy component so this parameter easily applies to all and a cross comparison will be implemented. This value is the weighing probability in which an action is chosen. Respectively 0 and 1 are by definition of this value the boundaries. An higher or lower chances of choosing an action do depends on this parameter: a higher chance that the action chosen is random is the lower bound whereas the closer the value is to 1 the higher probability the action chosen is greedy and hence returns the highest reward.

**Motivation for different models**

Balancing the ratio between explore and exploit is a common topic of reflection in the reinforcement learning literature. Different approaches exists: Thrun highlight methods utilizing counters (Thrun, 1992)[3], Brafman and Tennenholtz uses model learning (Brafman et al, 2002)[4] or reward comparison in a biologically-inspired manner as pioneered by Ishii, Yoshida and Yoshimoto (Ishii et al, 2002)[5].

The ε-greedy method, as reported by Sutton and Barto (Sutton R. and Barto A., 1998)[1], is the first choice due to its computationally lightness as it does not require to memorize any exploration specific data. It also achieve near optimal results in many applications by the hand-tuning of only a single parameter, as shown in the Vermorel application (Vermorel J. and Mohri, M., 2005)[7].

This method however still lacks on methods of adapting the method’s exploration rate on basis of the learning progress and our proposal is to use two different variations using a learning decay factor, in section 3.2, and a biological inspired system that makes use of neural networks in order to produce a set of choices to utilize, in section 3.3.

The purpose of this analysis is to compare the efficiency of classical version of the model against modern variation of it with the scope to better understand the agent’s learning progress.

* 1. **– Greedy Q Learning**

In the fundamental formulation of Q-learning algorithm, it is not specified what action the agent should take but it learns and apply a function that allows him to choose an optimal action. Exploit and Explore are the result of a combination of history usage and random action. We refer to the first as the knowledge that it has found for the current state *s* by doing one of the actions a that maximizes *Q(s,a)* while the second indicates the random action that it takes in order to build a better estimate of the optimal Q-function. In the first alteration, we will launch the agent with different values of indicating different mix of explore-exploit policies adopted.

There have been a number of suggested ways to trade off exploration and exploitation:

The ε-greedy strategy consist in selecting a greedy action that maximizes Q[s,a] 1-ε-portion of the time and mix with a random action ε-portion of the time, where 0 ≤ ε ≤ 1. It is possible to change ε through time and this is what it will be tested in the second algorithm (ε-greedy Epsilon Decay). Initially the expected behaviour is that the agent should select a more random strategy to encourage initial exploration and translate its cumulated rewards in knowledge to use as time progresses, acting more greedily.

The ε-greedy strategy treats all of the actions equivalently excluding the best one: in this way, the presence of two seemingly good actions among actions that look less promising, it may be more sensible to select among the good actions. More effort is put toward determining which of these promising actions is best, rather than putting in effort to explore the actions that look bad, which could be done by selecting those actions a with a probability depending on the value of *Q(s,a)*, a method to which we usually refers to as a soft-max action selection.

This algorithm provide the agent a random choice and the parameters are set to be fixed.

**Pseudocode**

Initialize arbitrarily

For each episode:

Measure initial state

For each step of the episode:

Choose at from using policy ε-greedy derived from

Take action a, measure next state and

(, ) (, ) + [ + , (, ) - (, ) ]

While s is **not** terminal

End

The agent updates the approximation of action-value function Q by its latest observation from the environment. Immediate reward and the new state is the information needed for Q learning to update. One thing to note, the agent is updating the Q value of the last state, i.e. when it transits from s to s' with action a, it uses the immediate reward r of this transition to updates Q(s,a), which is the value of performing action a in state s. When the agent receive a reward r from state s doing action a, with the maximum expected value in state s', it refines the Q(s,a) value. After many trials, the action-value function Q keeps improving and will converge to the optimal Q\*.

The representation below provides information regarding the pattern completed and the agent Q-matrix in relation to his states along the path. Representation has been made with 10 and 100 trial:

As it is possible to appreciate, the initial number of trials makes the agent to drive the helicopter toward a more erroneous and volatile pattern. The state of the wind are almost not considered for a training of 10 trials. Different is instead the pattern provided with 100 trials in which is visible how the helicopter is driven and how follows the states of the wind: firstly chase the region at the border with a worse wind state between the green and the blue. Then the 0-state wind, coloured in blue, allows the agent not to make extra effort until the region of increasing values of wind state. From here on the helicopter is driven toward increasing gradients until the reach of the highest one.

* 1. **– Greedy Epsilon Decay**

In this algorithm the epsilon which drive the greedy policy is weighted with exponential decay.

**Pseudocode**

Initialize arbitrarily

For each episode:

Measure initial state

For each step of the episode:

Choose at from using policy ε-greedy derived from

Take action a, measure next state and

(, ) (, ) + [ + , (, ) - (, ) ]

While s is **not** terminal

End

Again, the representation below provides information regarding the pattern completed and the agent Q-matrix in relation to his states along the path. Representation has been made with 10 and 100 trial

As is shown, the two graphs represents how drastically the pattern is improved while using a bigger number of trials. In particular, differently from the previous algorithm, the decay of the epsilon allow the agent to be less volatile and stale around its last state of imperturbation. The realized path is much smoother and the influence of the wind is much neater. In particular, let the reader consider the light blue state, in which the wind is in unspecified conditions (state = 0) so the agent maintain a stable pattern without too much volatility, like was happening in the previous algorithm.

**3.3 Recurrent Neural Networks and Deep Q Networks**

Deep Q Networks (DQN) last year where brought to the attention of many researchers and the world when Deepmind released a paper demonstrating the networks capability at playing Atari games. The research featured in the Nature publication and demonstrated that their implementation had overcome the issues that had typically challenged using a Neural Network as a function approximation for the Q values. Summarised in the table below from the paper Playing Atari with Deep Reinforcement Learning (Deepmind, 2015) the issue and the techniques that have been used to overcome these issues:

|  |  |
| --- | --- |
| **Issues** | **Technique** |
| Stability Issues | Reward Clipping (Normalisation) |
| Distribution of the data can change quickly | Error Clipping (Truncation) |

The issues outlined in the table above were implemented and were shown to have a good impact on the capability of the approximation. The analysis indicated that normalising the range of the reward to a finite range helped to support the issues of dealing with large Q-values and their respective gradients – one negative of this approach was that the model may find it harder to differentiate the difference between small and large rewards due to the normalisation. The second technique that was to introduced was error clipping into the model – this is a commonly used method to deal with the potential of exploding gradients.

The pseudocode below is a high-level description the methods that were implemented in the Deepmind paper and our respective model

**Pseudocode**

Initialize arbitrarily

For each episode:

Measure initial state

for each step of the episode:

Choose at from using policy ε-greedy derived from

Take action a, measure next state and

(, ) (, ) + [ + , (, ) - (, ) ]

Store the transition (,, ,) in memory\*\*

Sample random mini-batches of transitions from the memory, (st,at, rt, s\*)

Optimise the MSE between the Q-Network values and the Q-Learning Targets

While s is **not** terminal

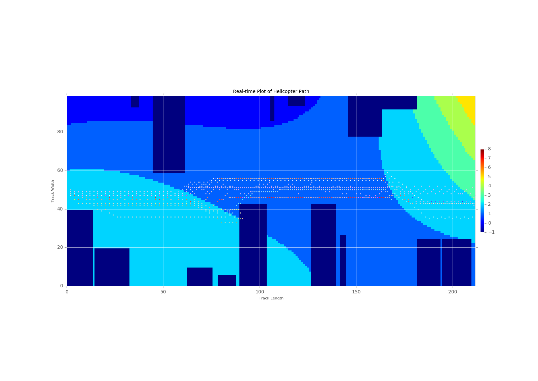
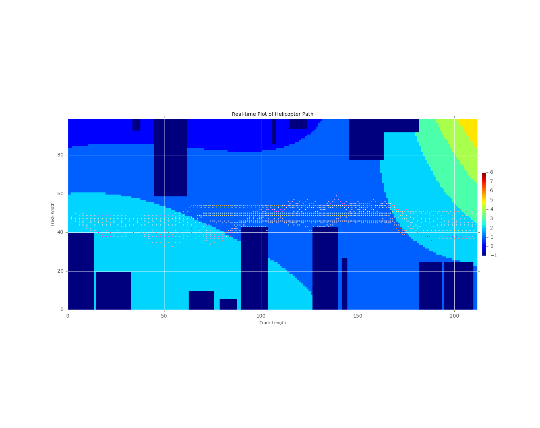
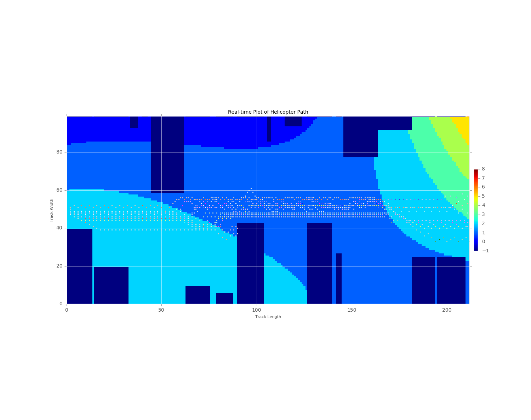
End

The original paper described that a Convolutional Neural Network (CNN) was used to “watch” the replays of the game. In our implementation of the Deep Q-Network, we use a CNN that outputs to a Long Short Term Memory (LSTM) layer and then finally into a linear output layer providing the Q-Values from the model. The key distinction was that an LSTM layer was used - it has been demonstrated in many papers that an RNN is very capable of capturing temporal patterns in sequential data. Since the goal of Q-Learning is too learn good policies for sequential decision making it, therefore, seemed appropriate to include this layer type. One feature that was implemented in the paper was that the model recorded and stored instances of the transitions – experience replay. This idea was used in one the most successful use cases of Neural Networks in Reinforcement Learning, this model was called TD-Gammon - this method was also used in our implementation.

1. **Parameters Alterations**

Fundamental parameters for each of the models are alpha, epsilon and gamma, which we extensively explained in their application. In the default case these have values of 0.65, 0.75 and 0.7 respectively. Changing these we are expecting the agent to The main parameter we recorded in order to keep track of the progress of the task were the time to complete the track and the percentage of completion of the same.

With 200 trials, below are showed charts represents respectively the time of completion of the track and the percentage of completion. In their default parameters the respective models look as follows:



As it is possible to appreciate the best realized trajectories are varying in relation to the change of the parameters and in each environment can be seen how more volatile the trajectory is given a more stochastic discovery process while become more stable with high values of the parameters.

The metrics used to compare the algorithms functionalities will be the plot of the Q matric, Time to Complete and Realized Reward.

*Q Matrix* – is the simple lookup function of the q values of the agent in relation to its position for each trial. Being configured in an iterative loop, it is possible to overlap different realized tracked Q trajectories, meant as plot of Q values in relation to the agent position and consequently to its state. The global cumulative sum produces what we have as Q matrices. Moreover a comparison among parameter changes and models will be performed and in each section will be specified the approach undertaken.

*Time to Complete* – is the logarithm of time spent to complete the track for each trial.

*Realized Reward* – is the logarithm of sum of the final Q matrix realized so that is possible to visualize the cumulative values of that, targeting to achieve a stationary stage in which

* 1. **Alpha**

Alpha parameters governs what is the learning rate in the Q-learning algorithm and is responsible of the overriding process of the old information with respect to the new one. It ranges between 0 and 1 and the agent is not learn anything when in is set to 0 while will consider only the most recent information when set to 1, which is optimal in a fully [deterministic](https://en.wikipedia.org/wiki/Deterministic_system) environment. When the problem is [stochastic](https://en.wikipedia.org/wiki/Stochastic_systems), the algorithm still converges under some technical conditions on the learning rate, that require it to decrease to zero.

In this section we are comparing the three methods and appreciate the contribution that each method will add to the agent scope of completing the track while changing the alpha parameter.

|  |  |  |
| --- | --- | --- |
| **Best Q matrices** | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
| \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results\case_three_alpha\Plot\figure_4.png | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_three_alpha\Plot\figure_4.png |  |

While the first model is more volatile, the second provides a more concentrated Q values around the best pattern of the agent. THE THIRD

Because of the epsilon decaying factor the stochastic approach of the agent is such that less error is realized (blue region in the first chart) and average values are summed around the best realized pattern, testifying the memory component effect.

|  |  |  |
| --- | --- | --- |
| Time to Complete | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
| \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results\case_three_alpha\Plot\figure_5.png | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_three_alpha\Plot\figure_5.png |  |

The time to complete, measured as Log of the actual time to complete per trial, provides an indication of how the first method is more concentrated around the zero level (on y axis) while the second spend more time for the first iteration while decays in time because the learning rate, increasing, stores more memory, avoiding random behaviour.

|  |  |  |
| --- | --- | --- |
| Realized Reward | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
| \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results\case_three_alpha\Plot\figure_6.png | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_three_alpha\Plot\figure_6.png |  |

The realized reward, as has been defined before, shows the irregularities of how each parameters reaches the maximum log reward. Concentration of high final rewards after 200 trial are provided by median region of learning rate parameter [0.5, 0.7] in the first model while the high rewards are achieved by a sifted range of values in the second model [0.3, 0.6]. The THIRD

* 1. **Epsilon**

In this

|  |  |  |
| --- | --- | --- |
| **Best Q matrices** | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
| \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results\case_four_epsilon\Plot\figure_4.png | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_four_epsilon\Plot\figure_4.png |  |

|  |  |  |
| --- | --- | --- |
| Time to Complete | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
| \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results\case_four_epsilon\Plot\figure_5.png | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_four_epsilon\Plot\figure_5.png |  |

|  |  |  |
| --- | --- | --- |
| Realized Reward | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
| \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results\case_four_epsilon\Plot\figure_6.png | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_four_epsilon\Plot\figure_6.png |  |

* 1. **Gamma**

In t

|  |  |  |
| --- | --- | --- |
| **Best Q matrices** | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
| \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results\case_two_gamma\Plot\figure_4.png | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_two_gamma\Plot\figure_4.png |  |

|  |  |  |
| --- | --- | --- |
| Time to Complete | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
|  | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_two_gamma\Plot\figure_5.png |  |

|  |  |  |
| --- | --- | --- |
| Realized Reward | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
|  | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_two_gamma\Plot\figure_6.png |  |

* 1. **Rewards Alterations**

Changing the rewards can produce a process know as "optimism in the face of uncertainty". This method consist in initializing the Q-function to values that encourage exploration. If the Q-values are initialized to high values, the unexplored areas will look good, so that a greedy search will tend to explore. While this will encourage exploration, however, the agent can hallucinate that some state-action pairs are good for a long time, even though there is no real evidence for it. A state only gets to look bad when all its actions look bad. It takes a long time to get a realistic view of the actual values when all of these actions lead to states that look good. This is a case where old estimates of the Q-values can be quite bad estimates of the actual Q-value, and these can remain bad estimates for a long time.

To get fast convergence, the initial values should be as close as possible to the final values; trying to make them an overestimate will make convergence slower.

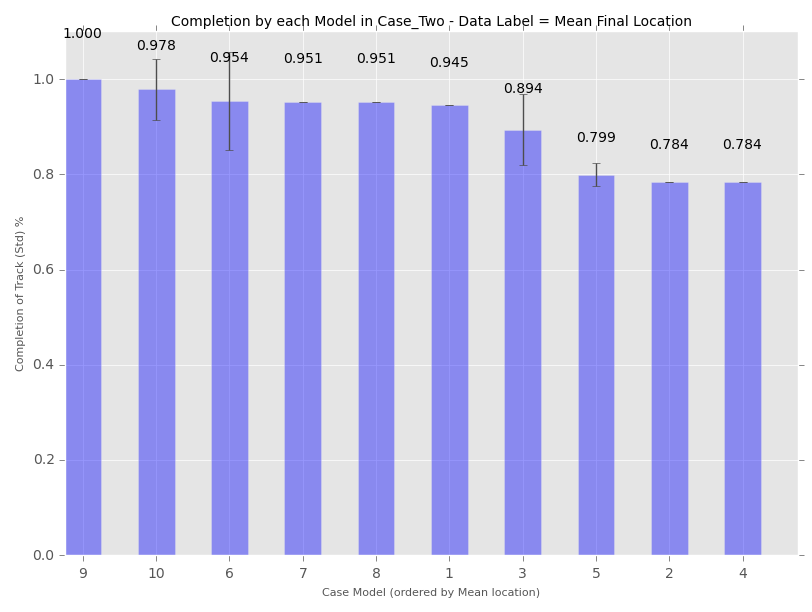
|  |  |  |
| --- | --- | --- |
| **Best Q matrices** | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
| \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_1\case_five_rewards\Plot\figure_4.png | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_four_epsilon\Plot\figure_4.png |  |

|  |  |  |
| --- | --- | --- |
| Time to Complete | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
| \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_1\case_five_rewards\Plot\figure_5.png | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_four_epsilon\Plot\figure_5.png |  |

|  |  |  |
| --- | --- | --- |
| Realized Reward | | |
| **– Greedy** Q Learning | **– Greedy** Epsilon Decay | Deep Q Networks |
| \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_1\case_five_rewards\Plot\figure_6.png | \\U00PFL01.HD00.UNICREDITGROUP.EU\J000110$\J000110.HOME\Redirected_Profile\Desktop\Master\Master\SA\cw\rf_helicopter-master\Results_2\case_four_epsilon\Plot\figure_6.png |  |

1. **Analysis**

By analysing the result we can see that



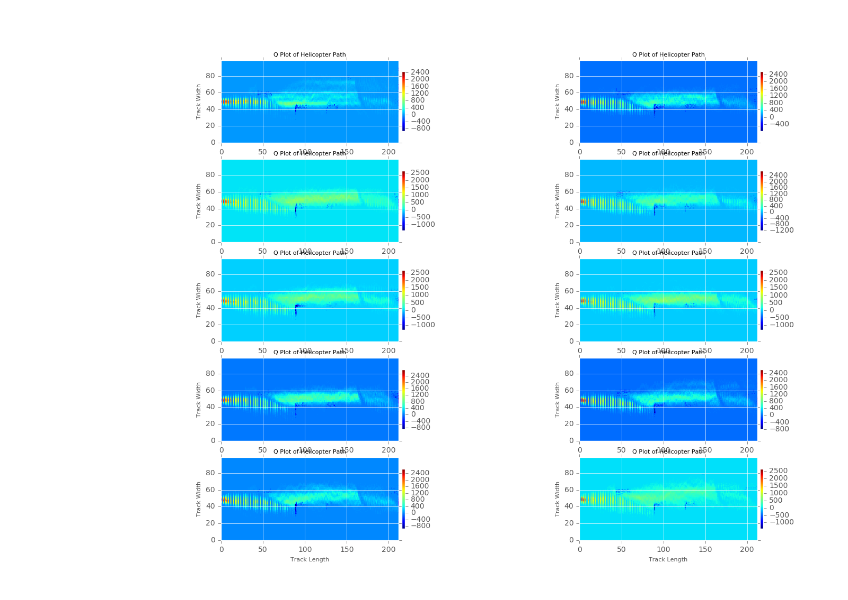
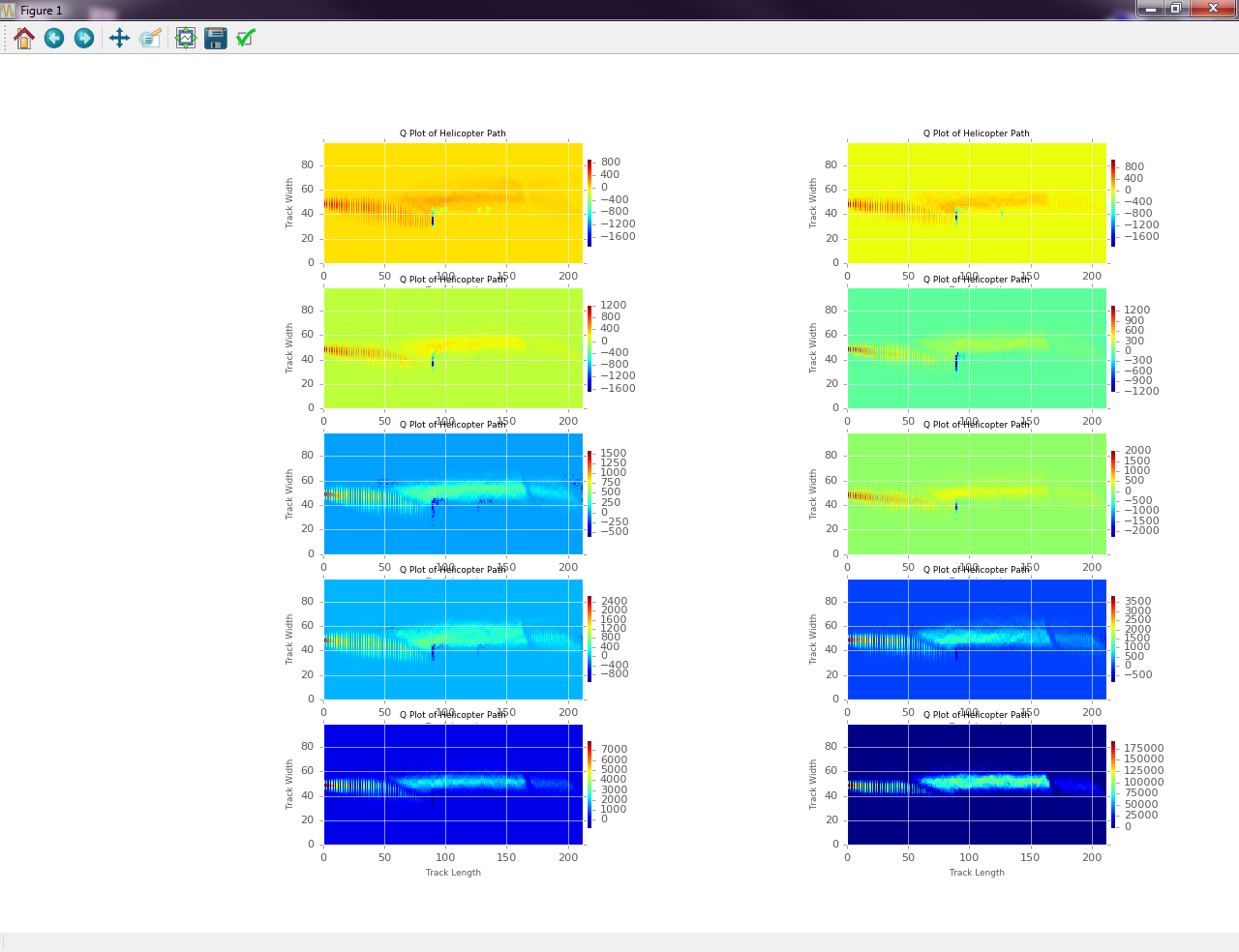
1. **Conclusion**
2. **Appendix**

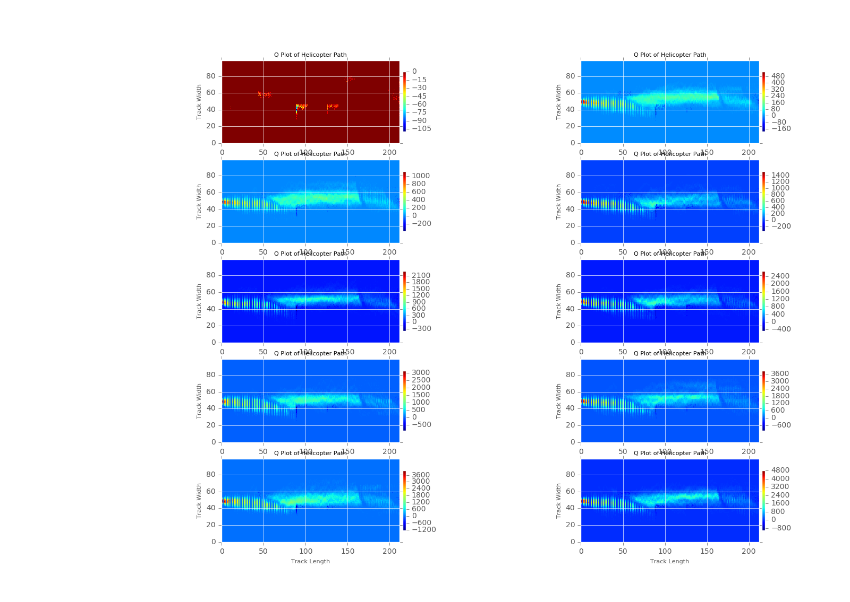
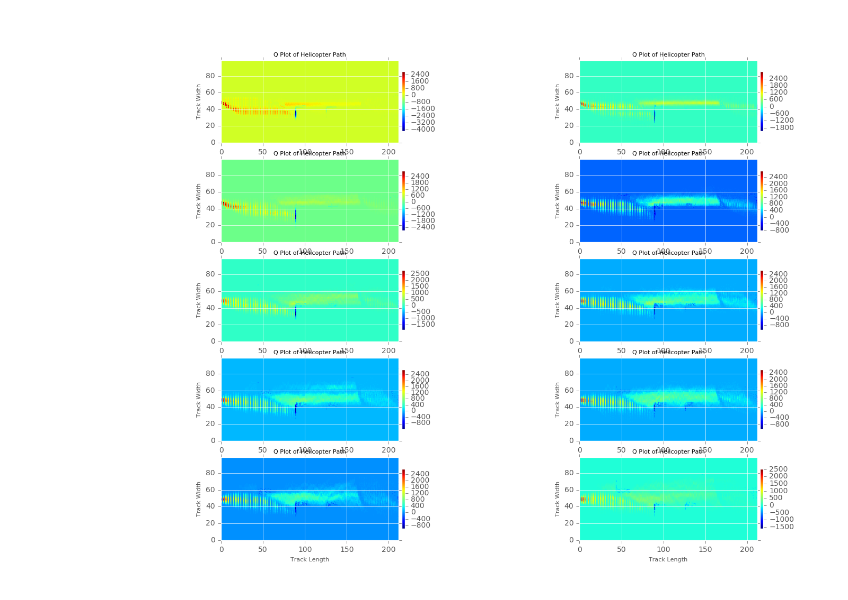
**A.1**

Variations of Gamma, Alpha, Epsilon and Rewards for first model with **– Greedy** Q Learning. Following, values and representation of Q matrixes in intermediates levels.

Steps taken for the variables Gamma, Alpha, Epsilon are ranging from 0.1 to 1 while Rewards of completion of track varies from 10 to 500 with increments of 50, the crashed case ranges from -1 to -110 with decrements of -10 and open space reward changes from 0 to 10.

Variables are represented from left to right from top to bottom respectively for Gamma, Alpha, Epsilon and Rewards.



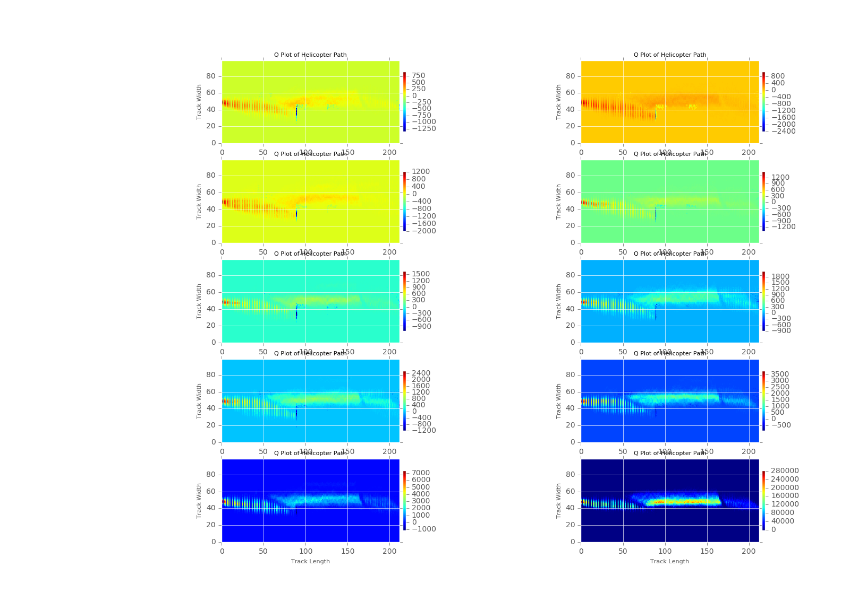
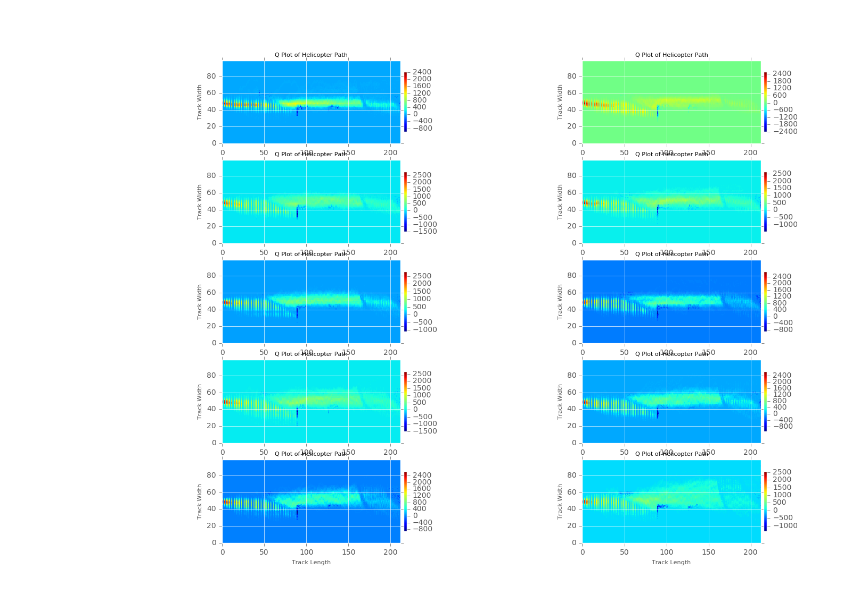
****

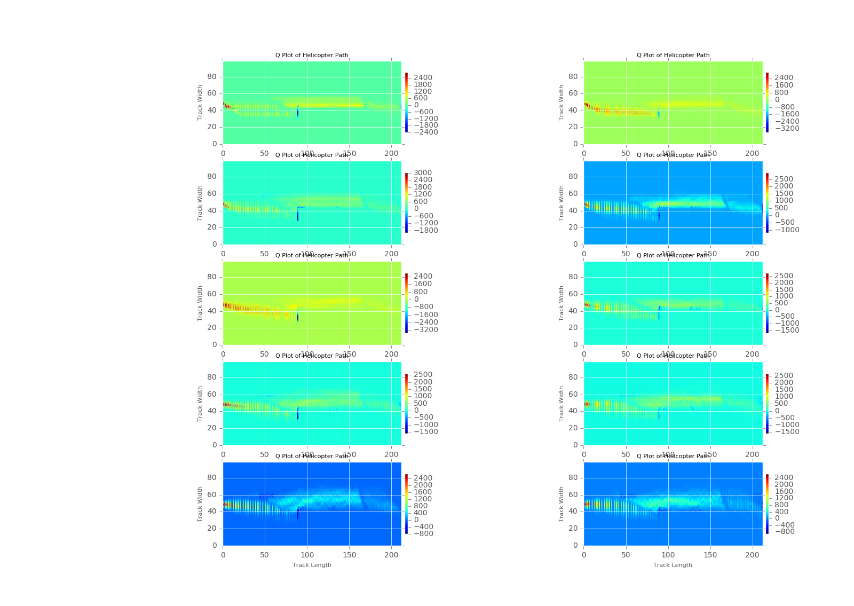
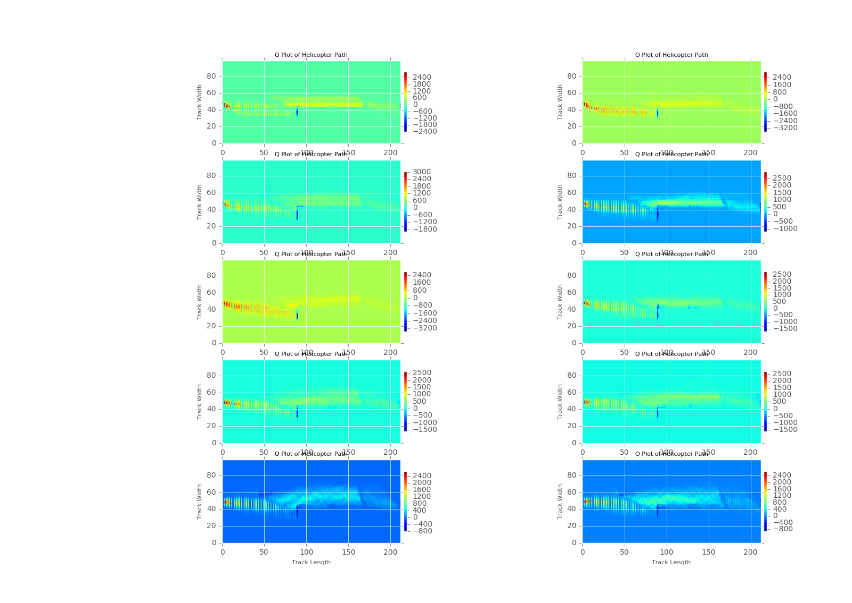
**A.2**

Variations of Gamma, Alpha, Epsilon and Rewards for second model with **– Greedy** Epsilon Decay. Following, values and representation of Q matrixes in intermediates levels.

Steps taken for the variables Gamma, Alpha, Epsilon are ranging from 0.1 to 1 while Rewards of completion of track varies from 10 to 500 with increments of 50, the crashed case ranges from -1 to -110 with decrements of -10 and open space reward changes from 0 to 10.

Variables are represented from left to right from top to bottom respectively for Gamma, Alpha, Epsilon and Rewards.

****

** **

**A.3**

Variations of Gamma, Alpha, Epsilon and Rewards for third model with Deep Q Network. Following, values and representation of Q matrixes in intermediates levels.

Steps taken for the variables Gamma, Alpha, Epsilon are ranging from 0.1 to 1 while Rewards of completion of track varies from 10 to 500 with increments of 50, the crashed case ranges from -1 to -110 with decrements of -10 and open space reward changes from 0 to 10.

Variables are represented from left to right from top to bottom respectively for Gamma, Alpha, Epsilon and Rewards.

**References**

[1] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)

[2] Watkins, C.: Learning from Delayed Rewards. PhD thesis, University of Cambridge, Cambridge, England (1989)

[3] Thrun, S.B.: Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, PA, USA (1992)

[4] Brafman, R.I., Tennenholtz, M.: R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research 3 (2002) 213–231

[5] Ishii, S., Yoshida, W., Yoshimoto, J.: Control of exploitation-exploration metaparameter in reinforcement learning. Neural Networks 15(4-6) (2002) 665–687

[6] Heidrich-Meisner, V.: Interview with Richard S. Sutton. K¨unstliche Intelligenz 3 (2009) 41–43

[7] Vermorel, J., Mohri, M.: Multi-armed bandit algorithms and empirical evaluation. In: Proceedings of the 16th European Conference on Machine Learning (ECML’05), Porto, Portugal (2005) 437–448

[8] Caelen, O., Bontempi, G.: Improving the exploration strategy in bandit algorithms. In: Learning and Intelligent Optimization. Number 5313 in LNCS. Springer (2008) 56–68

[9] Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. Technical Report CUED/F-

INFENG/TR 166, Cambridge University (1994)

[10] Bertsekas, D.P.: Dynamic Programming: Deterministic and Stochastic Models. Prentice Hall (1987)

[11] George, A.P., Powell, W.B.: Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. Machine Learning 65(1) (2006) 167–198

[12] Robbins, H.: Some aspects of the sequential design of experiments. Bulletin of the American Mathematical Society 58 (1952) 527–535

[13] Awerbuch, B., Kleinberg, R.D.: Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, ACM (2004) 45–53

[14] Azoulay-Schwartz, R., Kraus, S., Wilkenfeld, J.: Exploitation vs. exploration: Choosing a supplier in an environment of incomplete information. Decision Support Systems 38(1) (2004) 1–18