ECSE 526: Assignment #3

Presented to: Jeremy Cooperstock

Prepared by

Rahul Behal [260773885]

[ECSE 526 – Artificial Intelligence]

Department of Electrical and Computer Engineering

McGill University

2020.11.14

Table of Contents

1 Generalization and Function Approximation

1.1 Approach

The task for this report was to develop an active reinforcement learning algorithm to play the game Q*bert on an Atari 2600 emulator through the Arcade Learning Environment [1]. The algorithm used was Q-learning with function approximation. This is a model-free method that was chosen due to the intractable state size of the problem.

This report will assume that the reader is generally familiar with the rules and objectives of the game, but a deeper explanation can be found referenced from an Atari magazine [2].

The game's display looks as follows:

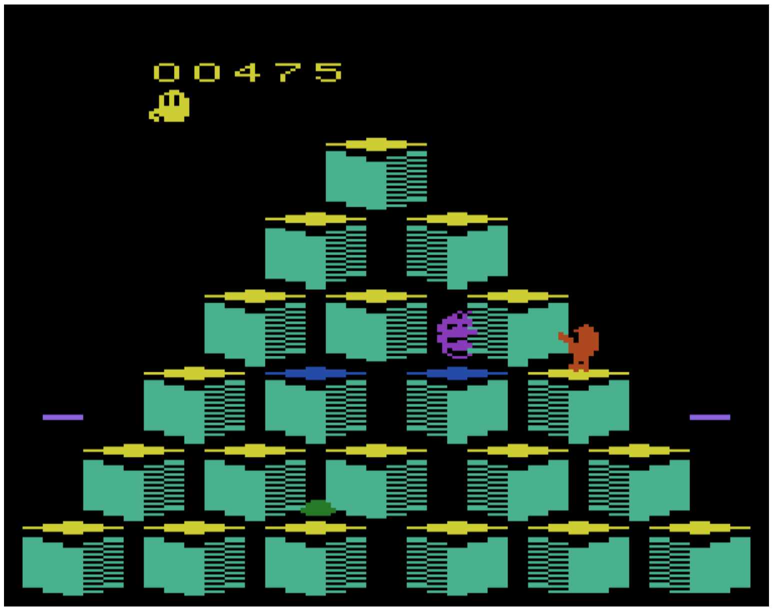

*Figure 1 Screencap of Q*bert on Atari 2600 emulator Stella*

The state was represented through considering the player's position, number of lives, status of the blocks, position of enemies, position of friendly entities, and the status of the discs.

The player's position was represented as a tuple of the x and y pixel coordinates while the number of lives was a simple integer.

The status of the blocks, position of the enemies, and position of the friendly entities were represented through a 2D array with the rows corresponding to the rows of blocks in Q*bert and the column number referring to which block it is from the left. If the block is the target colour, the array holds a 1 value, otherwise it is 0. With respect to the position of the enemies and friendly entities, if they are on a specific block, it holds a 1 value in the array, otherwise it is 0.

Finally, the state of the discs was represented by an array in which the value would be a tuple of the x and y coordinates of the disc if it exists, or None if it does not exist.

Overall, this leads to an approximate state size of $3.1*10^{21}$. As a result of this intractable state size, function approximation will be used as a representation for the Q-function.

Linear function approximation was used with six different features. There were 3 approaches considered representation of a Q-function: a complex approach, a simple approach, and an approach that was between the two, i.e. mixed.

The overall Q-function for the complex case can be represented as follows, where θ_n are the weights and $f_n(s, a)$ are the feature components:

$$\widehat{Q}_{\theta(s,a)} = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \theta_3 f_3(s,a) + \theta_4 f_4(s,a) + \theta_5 f_5(s,a) + \theta_6 f_6(s,a)$$

The weights were initialized randomly between the range [-1, 1].

Table 1 Various features considered in function approximation

Function	Feature Component	Choice Rational
$f_1(s,a)$	Distance between player and blocks that are not the target colour.	These are the blocks Q*bert needs hop on to gain points and progress the game. Relative position is valuable information for the algorithm.
$f_2(s,a)$	Distance between player and enemies.	The player dies if they are in the same position as the enemy. Therefore, relative position is valuable information for the algorithm.
$f_3(s,a)$	Distance between player and friendly entities.	The player gains points if they are in the same position as friendly entities. Therefore, relative position is valuable information for the algorithm.
$f_4(s,a)$ $f_5(s,a)$	Distance between the player and the left and right discs.	The player gains points if it kills the coily by hopping on the disc, so the relative position here is valuable again.
$f_6(s,a)$	The number of lives the player has left.	A negative reward is introduced for the player upon death. Therefore, when considering which state to choose given an action, the number of lives is valuable information.

It is worth noting that features were normalized by dividing by the highest possible values.

The mixed and the simple case sample features from the most complex case. The Q-functions for the simple and mixed case are represented as follows:

Simple:

$$\widehat{Q}_{\theta(s,a)} = \theta_0 + \theta_1 f_1(s,a)$$

In the simple case, the algorithm is just given the distance to blocks that are not in end the goal state yet. This was considered to be the bare minimum information pointing towards an objective.

Mixed:

$$\widehat{Q}_{\theta(s,a)} = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \theta_3 f_3(s,a)$$

In this case, the algorithm is just given the distance to blocks that are not in end the goal state yet as well as the distances to friendly entities and enemies. Now multiple factors are included that contribute to the overall reward received.

The algorithm was also implemented for 3 different distance metrics: Euclidean, Manhattan, and Hamming. Only the Euclidean and Manhattan distance metrics will be discussed in this report. In all cases, the measure of distance used was the x and y pixel coordinate distances. The x and y coordinate was always taken to be the center of the block. In the case of the position of an entity, the center of the block it is on is used. The expressions for distance metrics are as follows, where **p** and **q** are coordinate vectors.

Euclidean:

$$d(\mathbf{q},\mathbf{p}) = \sqrt{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{q_i} - \boldsymbol{p_i})^{\mathbf{2}}}$$

Manhattan:

$$d(\mathbf{q},\mathbf{p}) = \sum_{i=1}^{n} |q_i - p_i|$$

1.2 Results

In order to keep consistency, the 3 different generalization approaches were compared by holding the other variables constant, namely the distance metric and the exploration function. The results are represented through graphs of the 50 episode moving averages of the score. Total training time was 1500 episodes across the three approaches.

The results are shown as follows:

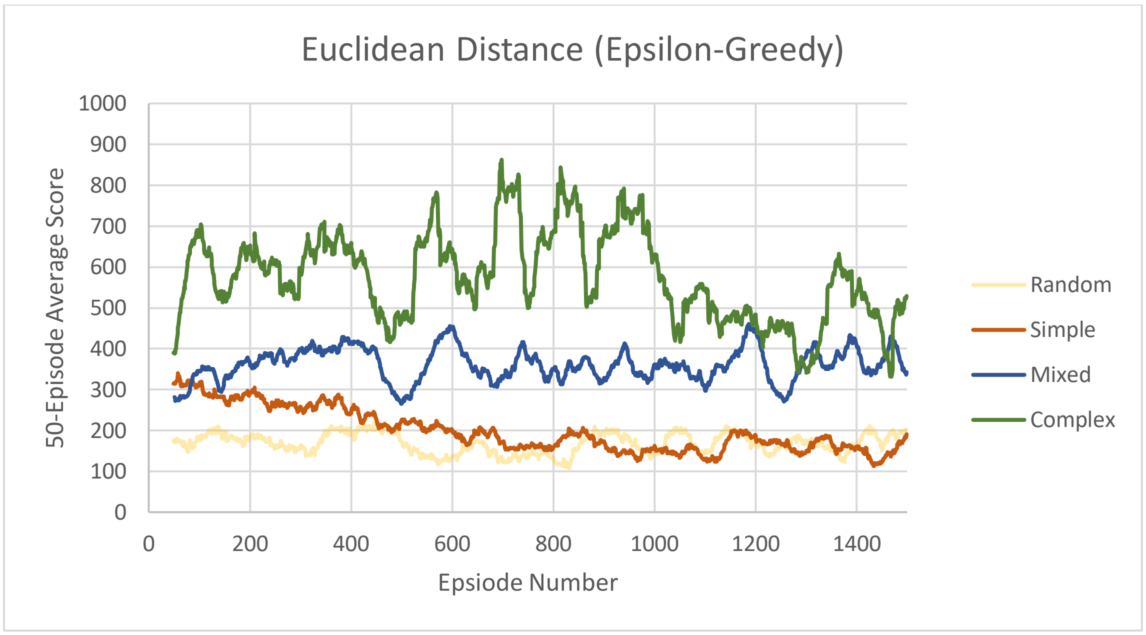

Figure 2 Generalization approach comparison using Euclidean distance with eps-greedy exploration

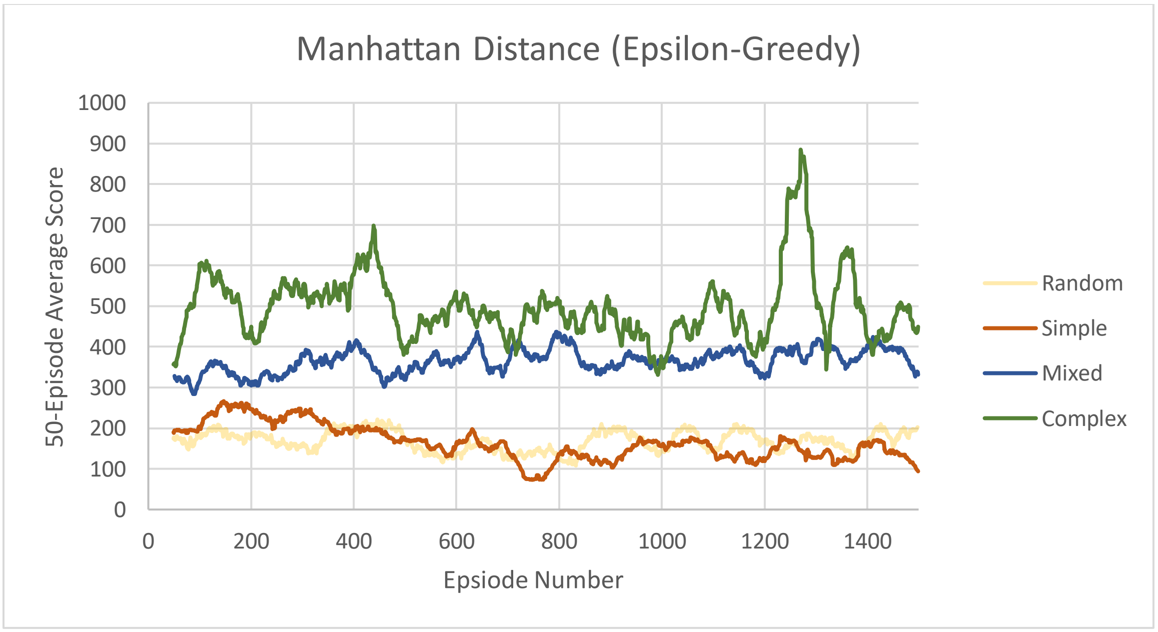

Figure 3 Generalization approach comparison using Manhattan distance with eps-greedy exploration

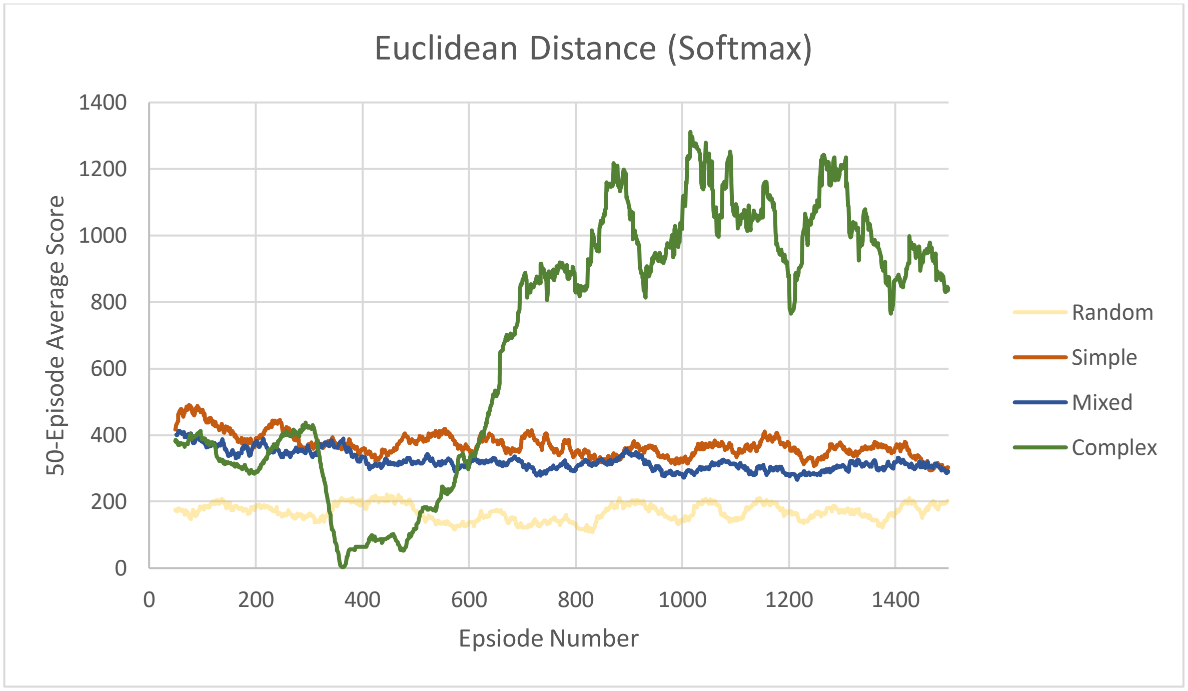

Figure 4 Generalization approach comparison using Euclidean distance with softmax exploration

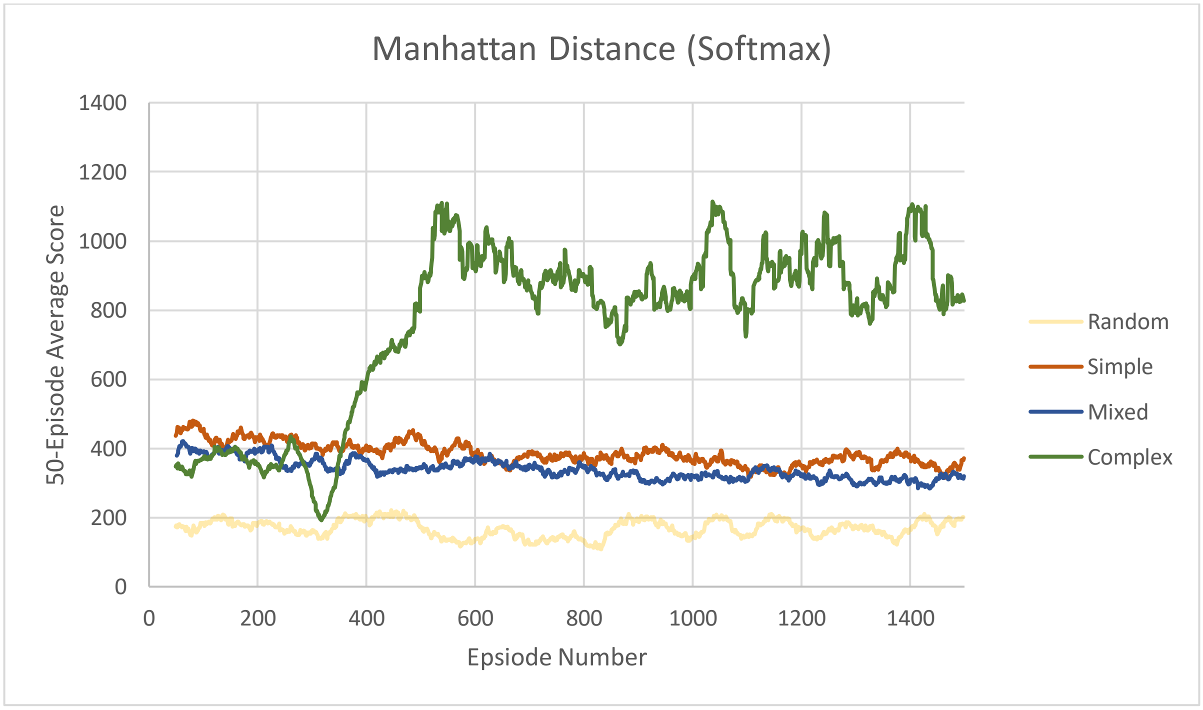

Figure 5 Generalization approach comparison using Manhattan distance with softmax exploration

The results for an agent making completely random moves was added as a baseline to all graphs. It can be seen that the function approximations beat out the random agent significantly in all cases except for the simple approximation using epsilon-greedy exploration. The nuances in the exploration algorithms will be discussed in the subsequent section of the report.

In all cases, the complex algorithm including six different features outperformed all other algorithms by a statistically significant margin. This suggests that the additional functions, namely disc distance and the number of lives contributed heavily to the learning. After further exclusion, it was found that a significant contributor to the performance was the lives feature. The algorithm quickly learns to value a higher number of lives over a lower one when considering next actions; this can be seen through a consistently high $\boldsymbol{\theta_6}$ value.

In the epsilon-greedy cases, the mixed function approximation outperformed the simple function approximation by a statistically significant margin. This points to the fact that the additional distance information from the friendly and enemy entities contributes to the performance of the agent. The algorithm learns to avoid the purple enemies and draw closer to the green friendly entities. This can again be seen through the relative weight values. A lower negative weight for the enemy distance weight, $\boldsymbol{\theta_2}$, and a relatively higher weight for the friendly entity distance weight, $\boldsymbol{\theta_3}$.

In the softmax cases, both the simple and the mixed function approximation outperform the random agent by a significant margin. However, it can be seen that the simple approximation actually slightly outperforms the mixed function approximation in this case. It should be noted that all approximations for the softmax algorithm outperformed the epsilon-greedy algorithm regardless which will be further discussed in the next section. Due to the nature of the softmax algorithm, it is possible that the function does not have enough feedback to start reliably choosing more favourable actions. As such, it may get stuck in a nonoptimal pattern that the simple algorithm avoids through only having one feedback and goal value.

Overall, through an analysis of the weights and the score results, it is clear that the features added in the complex case significantly impact the learning of the agent and the performance. It can be shown that the function in the complex case is learning the value of higher lives. There is also evidence to suggest that the function in the mixed case distinguishes between enemies and friendly entities and learns to avoid and gravitate towards them respectively. Finally, the data suggests that the agent is able to learn and perform better than random just given the simple case.

2 Exploration

2.1 Approach

The use of two different exploration functions was considered in this report: epsilon-greedy exploration and softmax exploration.

Epsilon-Greedy:

The epsilon-greedy exploration function aims to choose an action at random with probability ϵ, otherwise take the action that results in the highest Q-value. The probability, ϵ, is a hyperparameter that can be changed. After trial and error to tune, 0.05 was found to be an appropriate value. This is also in-line with previous research done on the topic [3]. ,

Though this exploration method does result in successful learning of most features to some degree, it is not extremely effective for Q*bert overall. This is likely due to the fact that each action alone does not result in a very significantly different state. As a result, a random one-off action here and there is not very effective at exploring possible better states. The specific results of this approach will be further discussed in the next subsection.

Softmax:

The softmax exploration function forms a Gibbs distribution using the Q-values over all possible actions at each decision making step. The action that the agent ends up choosing is a sampling from this distribution. This results in the agent being more likely to choose actions that have a relatively higher Q-value. The probabilities are calculated according to the following equation:

$$P(a) = \frac{e^{Q(s,a)/\tau}}{\sum_{i=1}^{n} e^{Q(s,a)/\tau}}$$

In the above equation, τ, is known as the temperature parameter. It is used to scale the distribution as a whole. Higher temperatures cause the actions to be more equiprobable, while lower temperatures have the opposite effect. Tuning of the temperature parameter was done through trial and error. An appropriate value for temperature was found to be 15. Below 10 and the agent would not explore enough to learn a good policy, and would result in poorer performance than random. Above 20 and the random variation was extremely high, resulting in difficulties consistently learning and performing.

2.2 Results

For sake of simplicity, the Euclidean distance function was used in all the below results. However, all three function approximation approaches were compared. Like in the previous section, the results are displayed through a graph of the episode number versus a moving 50-episode score average.

The graphs are as follows:

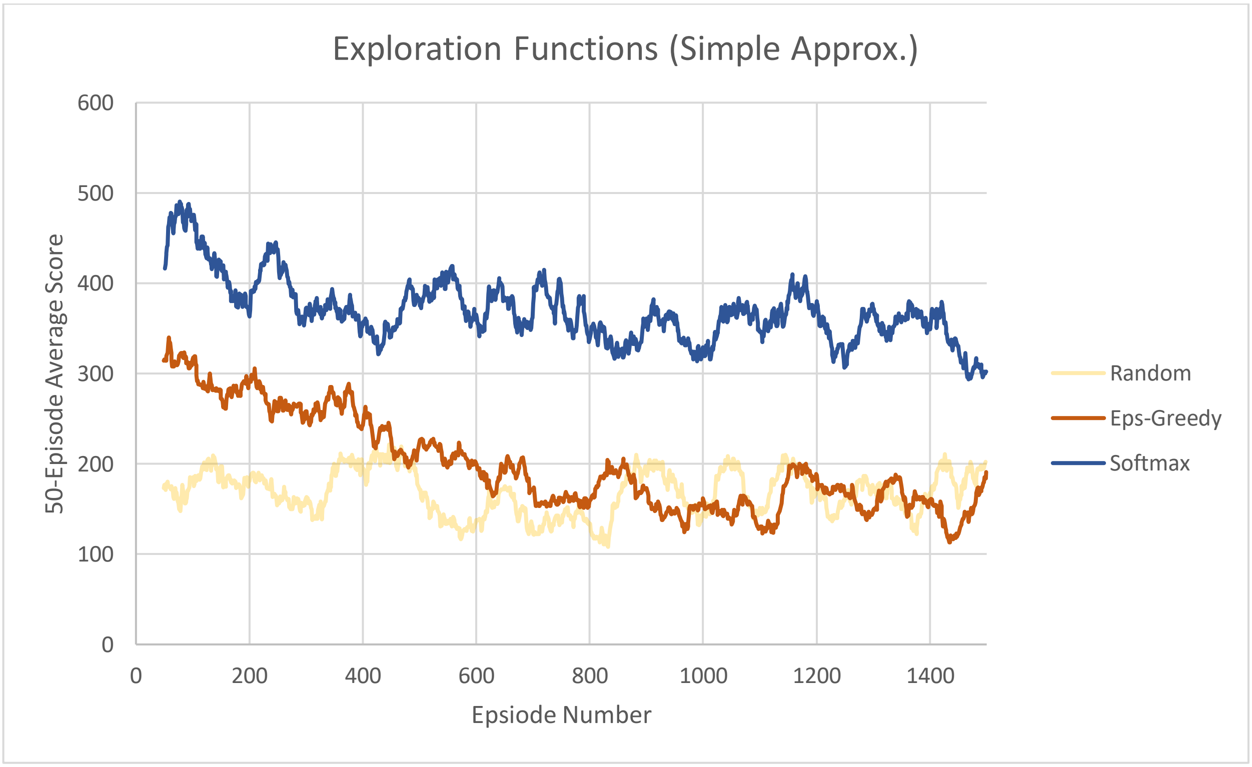

Figure 6 Exploration function comparison using simple function approximation

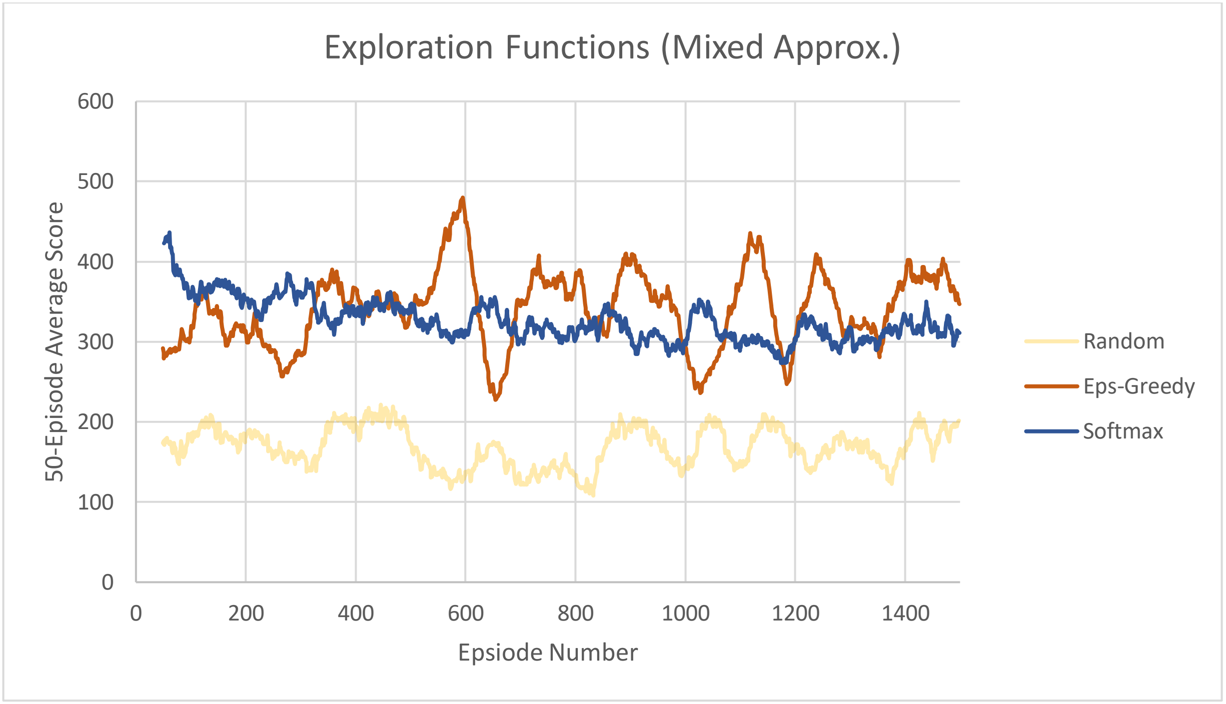

Figure 7 Exploration function comparison using mixed function approximation

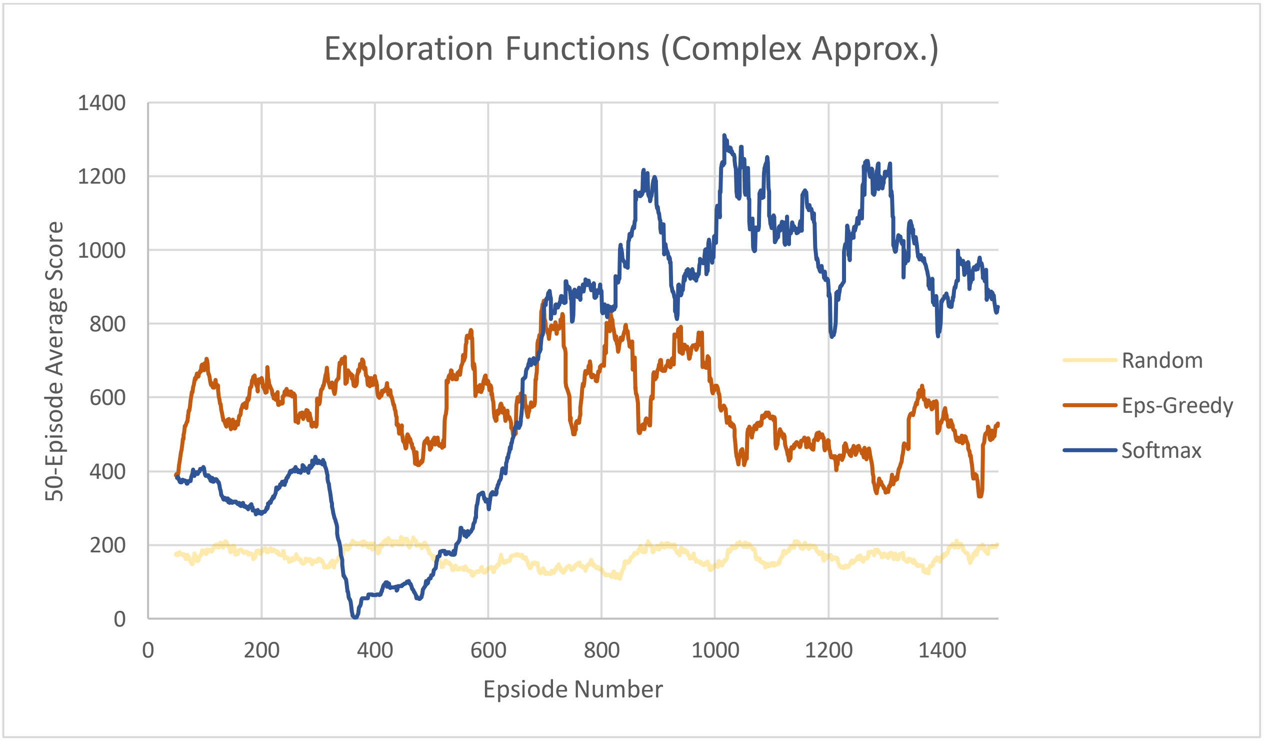

Figure 8 Exploration function comparison using complex function approximation

It is clear from the above graphs that both exploration algorithms are viable for getting at least above random performance. In the case of the simple and the complex function approximation, softmax exploration far outperformed epsilon-greedy. However, in the simple case it appears as though most of the learning is done in the first 50 episodes. After that, the performance of the algorithms trends downwards. As both exploration algorithms exhibit this behaviour, it is likely a limitation of the function approximation.

In the case of the mixed function approximation, it appears as though softmax does slightly worse on average than epsilon-greedy. Again, the actual learning appears to be done in the first 50 episodes. From there, in both algorithms the values are just ranging, though still much above random. The less random nature of the softmax algorithm is evident in the lower variance relative to the epsilon-greedy algorithm.

Most of the learning done by algorithms using epsilon-greedy exploration is done in the first few hundred episodes. After that, the random action-taking appears to hinder the algorithm, causing a plateau in growth and subsequent ranging performance. The softmax algorithms also learns very quickly once momentum is gained going down a path that results in high rewards. Unlike the epsilon-greedy algorithm, it appears that this can happen even after a few hundred episodes as is shown in the complex case. Here most of the learning occurs in the 400-1000 episode range, and then the algorithm reaches a plateau and begins ranging.

Overall, for the game Q*bert, softmax exploration seems to perform much better than epsilon-greedy. The complex softmax algorithm consistently was able to attain double the average score of the epsilon-greedy algorithm. Most of the learning for both algorithms occurred within short time periods. Epsilon-greedy learned quite quickly and early on and then seemed to plateau. Softmax learned early on in the simpler cases. With the more complex case, it took softmax a while before learning took off, but when that happened it learned a lot and quickly before plateauing.

3 Agent Performance

All results displayed in the report thus far have been using the seed of 123. In order to ensure that the function is indeed generalizing and not overfitting for the specific level, multiple seeds can be tested. The best options found above will be used for the following tests, namely a Euclidean distance function with softmax exploration and complex function approximation.

The results can be seen as follows:

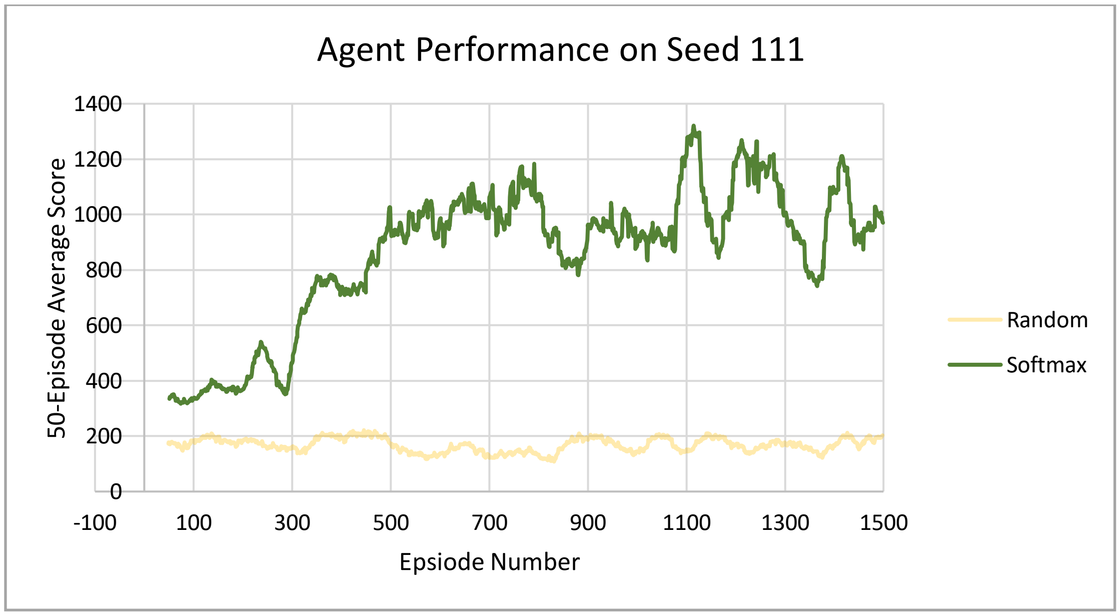

Figure 9 Agent performance of softmax algorithm with complex approximation on seed 111

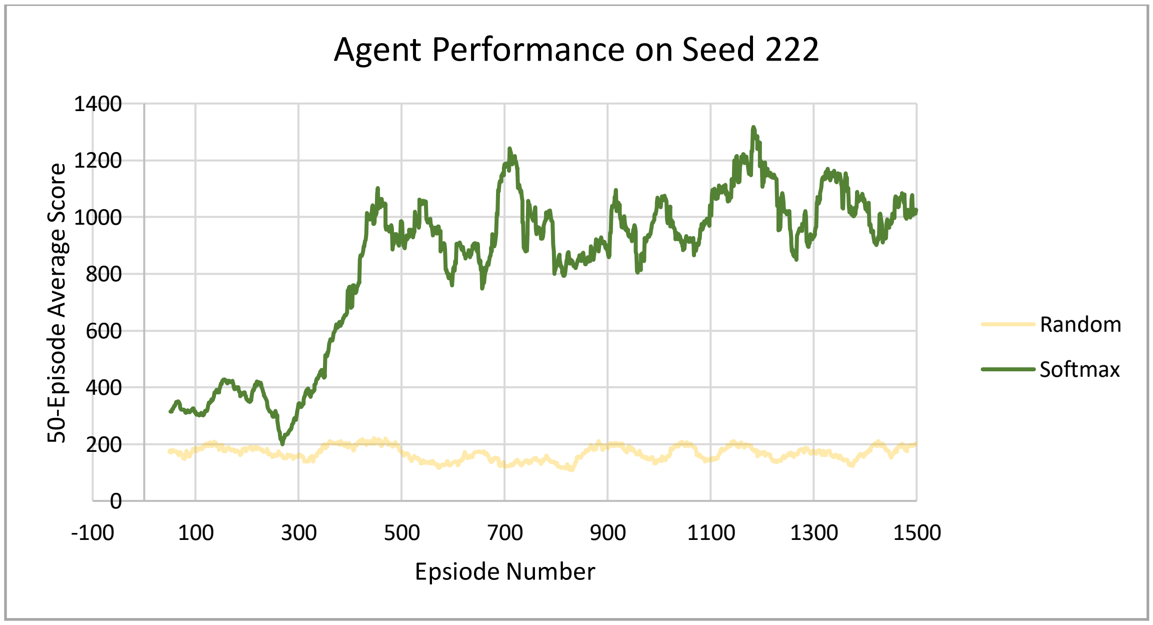

Figure 10 Agent performance of softmax algorithm with complex approximation on seed 222

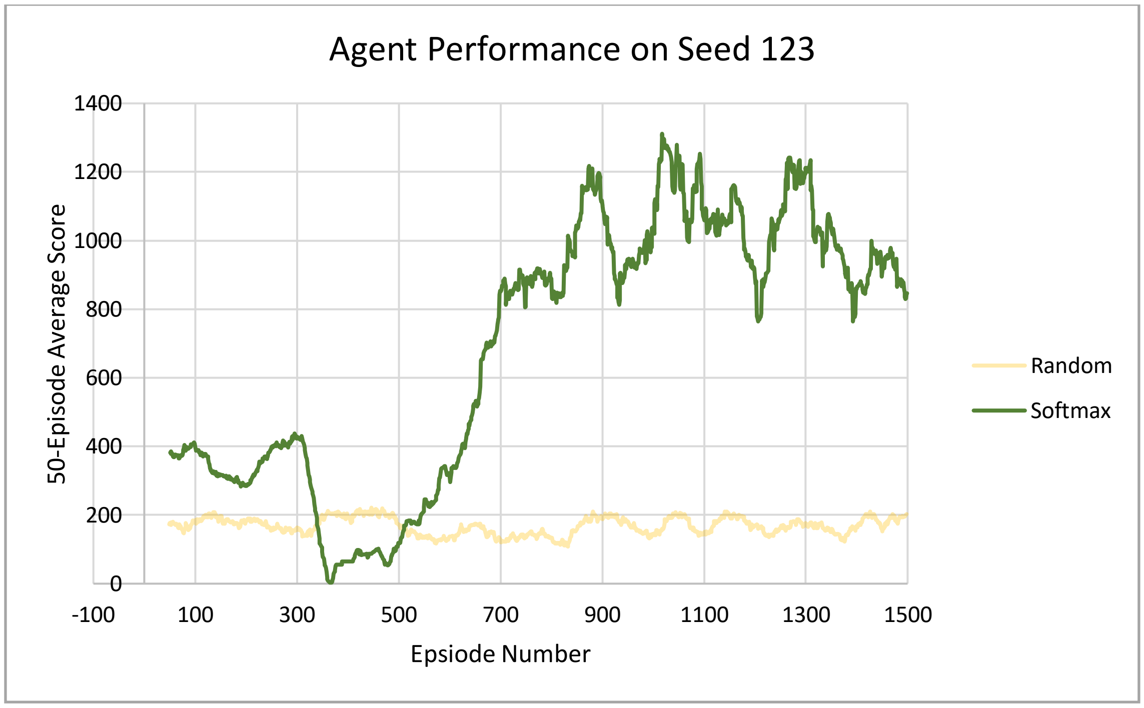

Figure 11 Agent performance of softmax algorithm with complex approximation on seed 123

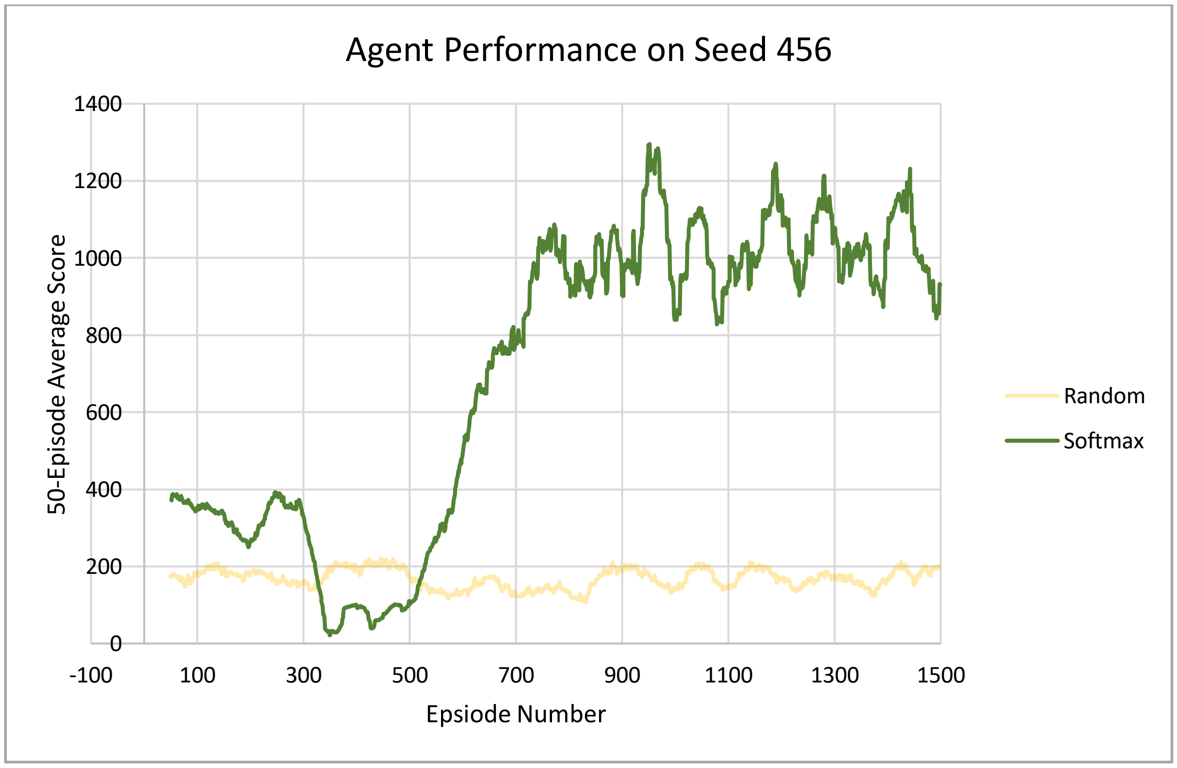

Figure 12 Agent performance of softmax algorithm with complex approximation on seed 123

The weights of the three trials can be analyzed to gain a better understanding of the agent's behaviour and various strategies that may have been employed.

Table 2 Weights for complex function approximation across various seeds

Seed	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
111	70.07	-43.89	15.83	-49.16	-49.27	1945.28
222	162.22	-44.22	1.64	-86.67	-78.95	2121.22
123	48.43	-53.62	46.24	3.03	-0.34	1830.86
456	46.90	-60.86	65.09	-15.38	-22.36	1815.79

It is clear that the generalization is legitimately effective and the function approximation was not just appropriating to that one seed. The algorithm learned in a very similar way across all three seeds, and capped out at about the same range as well. Additionally, across the range of all seed values, the weights maintained a similar range. The relative values and signs of these weights can be analyzed to extract what strategies the agent was able to learn.

It can be seen that the θ_6 values are consistently much higher than the rest. This suggest that the agent heavily valued actions that resulted in the agent having a higher amount of lives rather than a lower amount. The θ_1 values are consistently the second highest weight value. This is also quite logical as it is the weight representing the state of the blocks. This shows that the algorithm is actually learning the objective of the game.

The distance between the player and the enemies weight, θ_2, is consistently assigned a very low negative value. This is indicative of the fact that the player is associating a negative reward with the enemy, and is thus learning to avoid it. A similar phenomena occurs with the rewards from the green ball, θ_3. The weights of the disc distances seemed to range from positive to negative in a wide range. This can point to the fact that the algorithm wasn't able to accurately determine the strategic purpose of the discs.

Overall, the agent was able to showcase consistent learning of various principles in the game. This includes the concept of valuing lives and the overall objective of the game. Additionally, the algorithm was able learn and adapt strategies based on mechanisms like the distance of the enemy as well as the distance of the friendly entities. Thus it can be concluded that Q-Learning with function approximation is able to tackle the problem of playing the game Q*bert.

4 References

[1] M. G. Bellemare, Y. Naddaf, J. Veness and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents, Journal of Artificial Intelligence Research, Volume 47, pages 253-279, 2013.

[2] Brohaugh, W. (1983, January 1). Q*bert: A Player's Guide. Video and Arcade Games, 1(2), 28-28. Retrieved October 28, 2020, from https://www.atarimagazines.com/cva/v1n2/qbert.php

[3] Defazio, A., & Graepel, T. (2014). A Comparison of Learning Algorithms on the Arcade Learning Environment. Retrieved October 29, 2020, from https://www.aarondefazio.com/adefazio-rl2014.pdf.