ECSE 526-Artificial Intelligence

Assignment 3 - Ms. Pacman with Reinforcement Learning

Alexandre Coulombe - 260801407

March 11th, 2020

1 Description approach to generalization

The Arcade Learning Environment (ALE) interface uses the *Stella* Atari 2600 game emulator to run the game *Ms. Pacman*. The game displays frames of 210 pixels high and 160 pixels wide, where each pixel is a 7-bit (0-127) color value [1]. The game has four possible actions : up, down, left and right. These actions allow Ms. Pacman to move around in the game. The goal of the game is to eat all the dots, or food, on the game screen while avoiding the ghosts. Failing to avoid the ghosts will result in a life lost. Losing three lives leads to a game over.

If we use the approach of using Q-learning with a look-up table for all the state-action pairs, we would require a table with $4 \times 128^{210 \times 160}$ entries (which overflowed my calculator). Generalization must be used to be able to interpret all the state-action pairs and approximate the utility of a given state. This is done by doing **function approximation**, which is any form of representation other than a look-up table. This is done by representing the game screen by a set of features and using a weighted linear function of the features to approximate the Q value of the state-action pair, (s, a). The weighted linear function has the following form:

$$\hat{Q}(s,a) = \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + ... + \theta_n f_n(s,a)$$

The parameters of the function approximation, the theta values, give a weight on how important to value each of the features in the feature set. The weights on more important features are higher and lower on less important features. The parameters can be learned by using the temporal difference (TD) and Q-learning functions to update the parameters to get closer to the true Q-values of the true function. The TD and Q-learning equation becomes :

$$\theta_i = \theta_i + \alpha \Big[R(s) + \max_{a'} [\hat{Q}_\theta(s', a')] - \hat{Q}_\theta(s,a) \Big] \frac{\partial \hat{Q}_\theta(s,a)}{\partial \theta_i}$$

By using a feature set that fully represents the game state of the game Ms. Pacman, an AI using reinforcement learning and a function approximation can approximate the true function of the game after going through training and converging to parameters that estimate the true function. This feature set would allow for the selection of good state-action pairs. For example, avoiding ghosts, eating food and eating scared ghosts would be characterized as good state-action pairs. This approximation would also allow for the agent to approximate states it has never visited using information from states it has visited, making it able to make good state-action pair decisions in these not visited states. Lacking game features would make it more difficult to approximate the true function and even not be able to.

2 Results of Generalization

The game agent developed for the Ms. Pacman game uses the following game features in its approximation function : the distance to the nearest food, the number of food dots remaining, the distance to the nearest scared ghost, the distance to the nearest active ghost and the number of oscillations Ms. Pacman has performed. The features were selected for various reasons. The distance to the nearest food feature was implemented to give the agent insight on where to go for getting a higher game score. The number of food dots remaining feature is to allow the agent to know how much food it needs to eat to win. The distance to the nearest scared ghost feature is so that it is able to hunt that ghost after eating a power pellet for a large amount of points. The distance to the nearest active ghost feature is to allow the agent to know to get away from it. Finally, the oscillation feature is due to the agent getting stuck in corners and not leaving them, so the feature is meant to discourage the agent from staying in the same area for too long and be trapped by the ghosts.

At first, the agent only had the food features. The agent performed poorly with only these features. A second approach used only the ghost features, which did much better. Finally, both approaches were combined and the oscillation feature was added. This combination has a much higher peak performance than both previous approaches. The game score over 20 episodes for the three approaches can be seen on Figure 1.

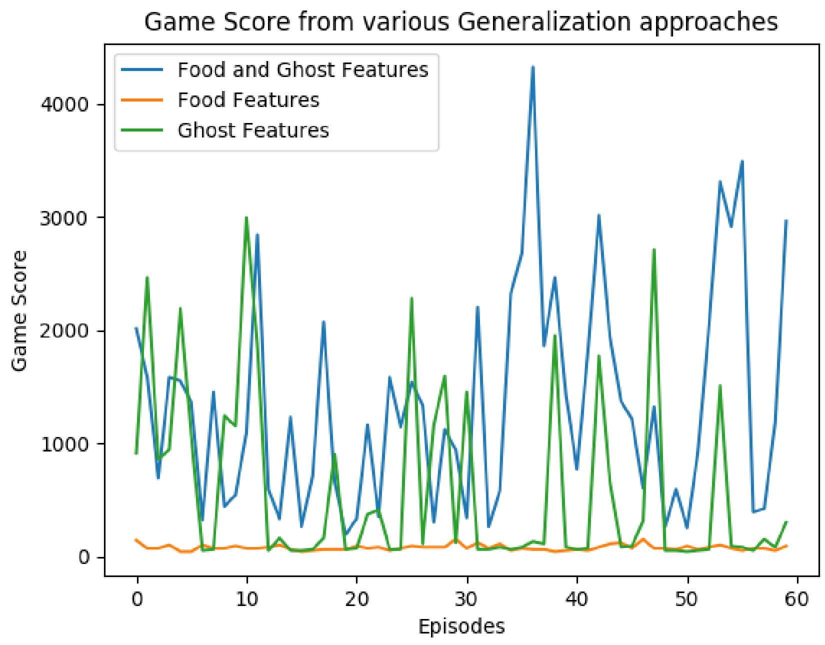

Figure 1: Game Score of different generalization approaches

When the agent only had the food features, it had no perception or knowledge of the ghosts in the game and would be eliminated by the ghosts very quickly. When only using the ghost features, the agent avoids the active ghosts when they are nearby, hides in corners when they are not and hunts the ghosts when it accidently eats a power pellet after being chased to it. The agent has no insight on the food and has no intention of winning, except if the ghosts chase it into eating all the food. This is unlikely, because the agent gets trapped by the ghosts way before it is even near eating all the food accidently. By combining both approaches and adding the oscillation feature to check if the agent is oscillating in a corner, the agent has the active ghost avoidance and scared ghost busting behaviour and the food eating behaviour from the previous approaches and leaves the corners more often without the ghosts chasing it out.

3 Description approach to exploration

The optimistic prior used for the Q-learning algorithm is the generalization equation described previously with the features mentioned. The equation that approximates the Q value of a state-action pair is as follows :

$$\hat{Q}(s,a) = \frac{\theta_1}{distance(nearest\_food)} + \frac{\theta_2}{distance(nearest\_scared\_ghost)} + \frac{\theta_3}{distance(nearest\_active\_ghost)} + \frac{\theta_4}{number\_of\_oscillations} + \frac{\theta_5}{number\_of\_food\_remaining}$$

Each of the features is inversely proportional to its value. This is because all the features selected are more important when they are small, or a short distance, and have less impact on the value when it is far away, longer distance. Each of the parameters would have a value that would scale the feature value. Their sum would give the approximate value of the state-action pair. To find the Q value of the state-action pair, the state would be the current state of the game, the action would be one of the possible actions and the features would be evaluated on the new state that the action would lead to from the current state.

The exploration functions used were the ϵ-greedy policy and a counter-greedy policy, using a counter for the amount of time the agent stays in an area. The ϵ-greedy policy makes the agent perform a random action with probability ϵ and the action found for the maximum Q value otherwise. The counter-greedy is like the ϵ-greedy policy, however the probability of performing a random move gets higher when the agent remains in an area too long. This makes it a weighted version of the ϵ-greedy policy. Both these approaches use the random move approach due to simplicity in implementation and less requirements for keeping track of state-action pairs that have not been explored yet, while allowing the agent to explore the game.

4 Results of Exploration

The performance of the exploration functions used, the ϵ-greedy policy and the counter-greedy policy, have been evaluated on the metric of game score as seen in Figure 2. For the ϵ-greedy policy with $\epsilon = 5\%$, the agent would act based on the optimistic prior for the state-action pairs most of the time. The agent would perform a random action that would give it insight on other states and be able to undo bad consequences that occur from the move. This would allow the agent to try actions and be able to get itself out of trouble it may have thrown itself in. In the episodes using the ϵ-greedy policy, the agent performed very well exploring through random action to find new areas of food and learning about the ghost's game mechanics when active and scared. Bad consequences were easily undone, but consecutive random action would eliminate the agent in certain situations. The counter-greedy policy did much worse. This is understandable because the agent would only be doing random actions until it has left the area it was staying in. This means no rationality is going into the move and is behaving randomly until it leaves the area or the ghosts get it, which is what happened most of the time. This policy makes the agent unable to get out of trouble when in danger, or, even worse, gets itself into danger.

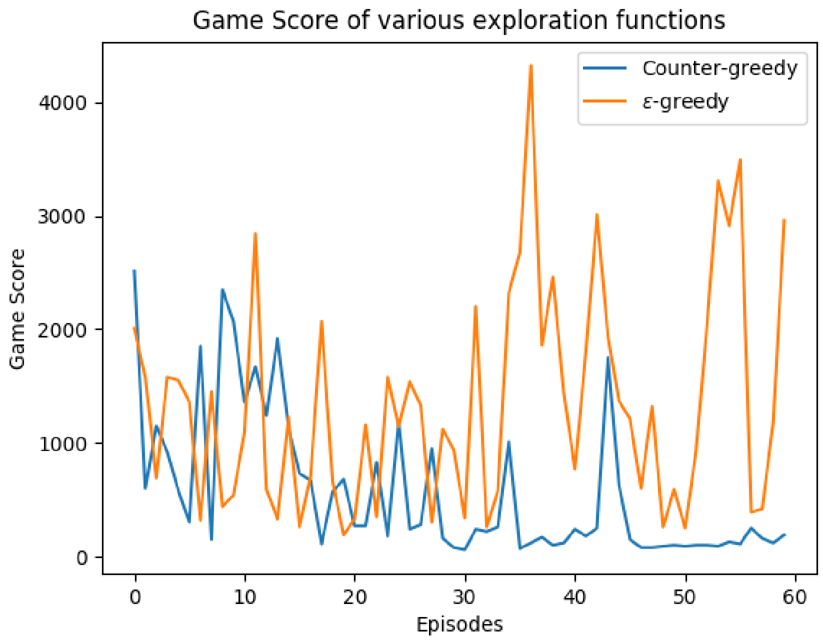

Figure 2: Game Score of different exploration functions

5 Agent Performance

When the agent was training, it was understanding the game functionality at a reasonable rate. Initially, the agent starts to move in one direction from the starting zone and starts to realize that it is a good thing to eat food. When the agent first encouters the ghosts, the agent doesn't realize the danger and loses a life to the ghost. It takes a few lives lost for the agent to understand that it should avoid colliding with the ghosts. It starts developing the strategy of avoiding the ghosts by running away from the nearest ghost in the direction that makes it get away from the ghost. When the agent accidently eats a power pellet, it does not understand that the ghosts give lots of points when eaten until a ghost accidently goes into the agent. The agent then realizes that the scared ghost are great and, after this game event happens a few more times, starts to hunt down the scared ghosts and chase them. The weighting of the features change quite often due to the game events and makes the agent gain and lose the listed behaviours during its training.

These behaviour are very good for the agent to maximize its performance. Avoiding ghosts kept the agent alive for longer, eating food makes it closer to winning the game and eating scared ghosts is the most rewarding way to make points within the features provided.

The final agent has a somewhat good performance. Altough it cannot beat the game in its current state, it has been able to get good game scores. The agent was in exploitation policy for 20 episodes on various seeds as seen in Figure 3. The training the agent received allows it to perform well on different seeds. On seed 1234, the agent was cornered a lot more often by the ghosts by getting boxed in, making it have an average game score that is lower than the two other tested seeds. In the other two seeds, the agent was able to avoid active ghosts and eat scared ghosts when powered up, while eating food in the meantime. However, it must also be noted that the agent oscillates a lot while moving around and would also place itself equidistant from numerous clusters of food. Also, the agent would put itself in corners at times and get boxed in which would make the agent lose a life. This may be due to the features forming a local optimum in the corners. The higher scores attained are when the agent is able to eat multiple scared ghosts in one power up interval or multiple power up intervals.

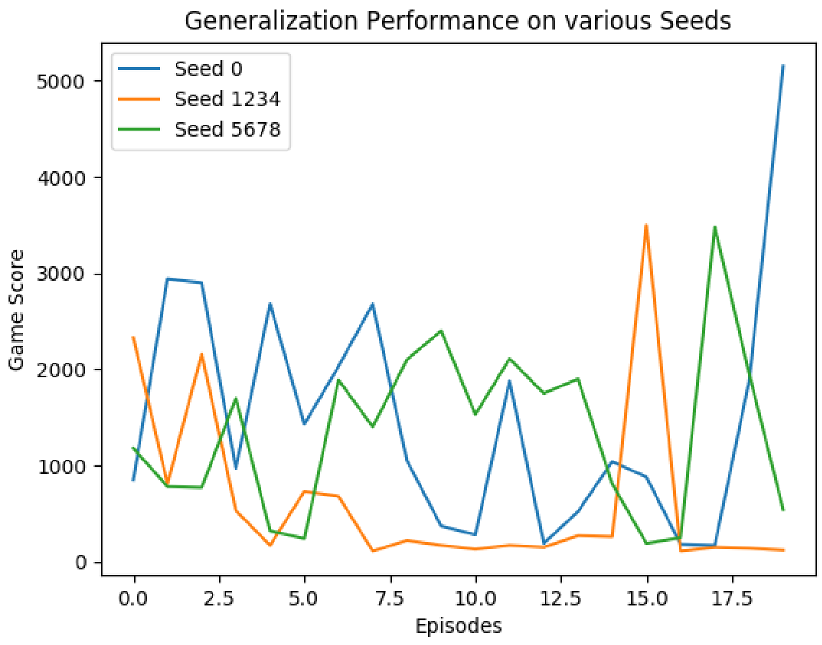

Figure 3: Game Score of different game seeds

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*, 47:253–279, jun 2013.