Assignment 3 ECSE 526

Zijie Jin 260617581

Disclaimer:

The algorithm included in this report greatly benefited from Github user *anassinator*'s previous investigation on deciphering the RAM meaning of pacman[1], specifically regarding which entries correspond to the position of pacman and ghosts.
Further, thanks to Otman for showing how to work with dictionary in python, without which the running time of this algorithm would have been orders of magnitude longer

Due to Windows subsystem linux's limitations, it was impossible to retrieve any screen information from ALE, including grayscale values and RGB values. Hence, the only information available to the agent was the 16X8 entries of RAM values.

Question 1:

In order to make evaluation of Q functions tractable, instead of feeding the whole 128 RAM entries into reinforcement learning model I chose and extracted several features from the RAM entries to be used as function approximations. Importantly, choice of these features should (at least somewhat) represent the true utility (or in this case Q function), and, in order for Q- learning to not have a high divergence chance, have to be linear in their parameters.

Considering the above constraints, the features I chose are listed as following:

1. Sum of geometric distance from all killing ghosts
2. Sum of number of pellets nearby
3. Sum of number of remaining powerups nearby
4. Remaining pellets on board
5. Sum of geometric distance from all edible ghosts
6. Number of killing ghosts heading into same direction as action a
7. Current score
8. Remaining lives

When evaluating these function approximations, the state used is the state immediately before application of action a. The rationale here is that since all actions make pacman move to a certain direction by a fixed amount, a good action should make pacman move towards a direction such that it moves away from ghosts that can kill it, towards edible pellets, power-ups or, if already powered-up, towards (edible) ghosts. Further, we want to avoid overall moving towards the same direction as non-edible ghosts. Lastly, we want to minimize the amount of remaining pellets on board. Note that I used "nearby" pellets and amount of remaining pellets "on board" as separate features; the idea is that even if the agent will not be able to find good state-action pairs such that it can finish a majority of pellets available, at least it can move towards getting as many points locally as possible.

As expected, the learned parameters for the above features are negative for 1,4 and 6, and positive for 2,3,5 and 7, suggesting that the features do describe the actual Q values fairly well. Choice of these function approximations also limited the computational time to a manageable length, which enabled longer training episodes. A good state-action pair will

give as high as possible values in feature 1,4,6, and as low as possible values for feature 2,3,5 and 7. Unfortunately, judging from the end results of my agent's performance, more optimization needs to be done. However, performance of a RL agent is much more than a mere function of how good its function approximations are, but also depends on other hyperparameters such as learning rate, discount factor, etc., so I would argue that it is fair to claim that this function approximation still somewhat accurately characterizes actual good state-action pairs, as shown by the correct signs of its weights function.

Performance Evaluation Methods:

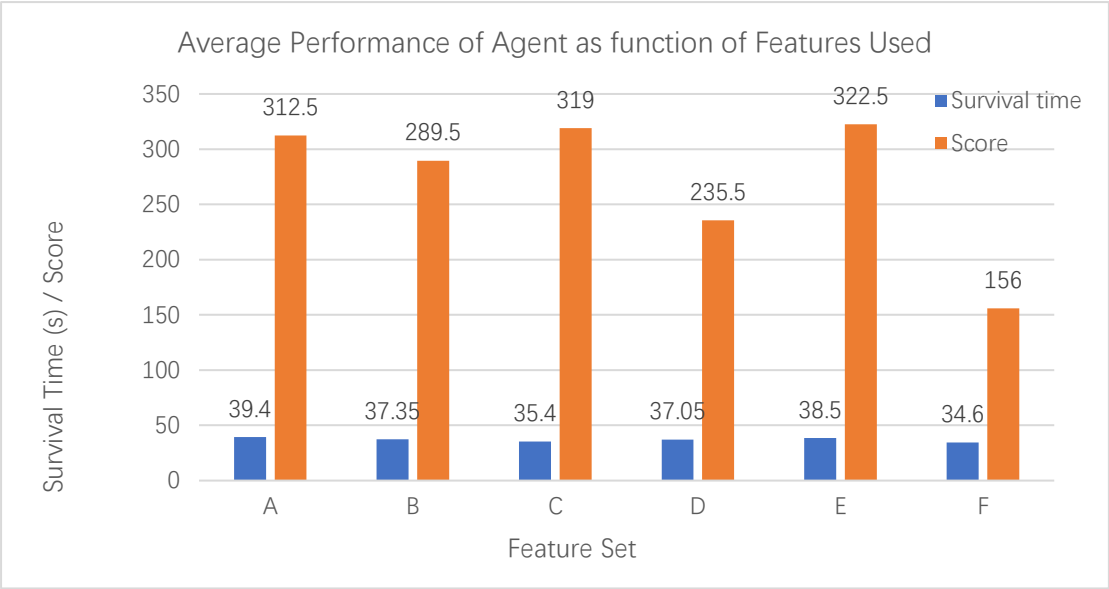
The performance of all agents evaluated below followed this protocol:

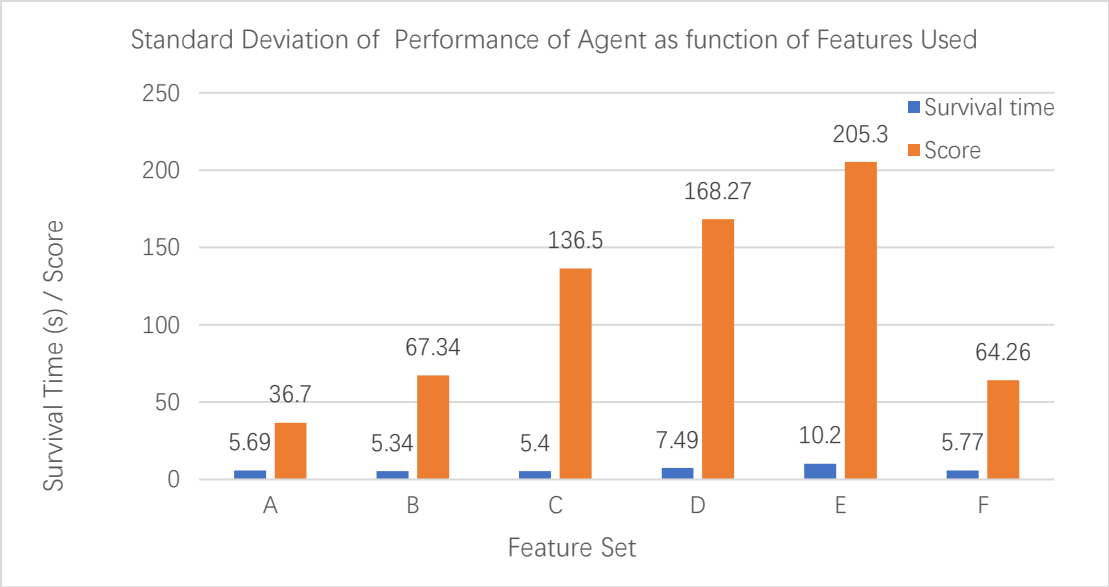
Agent is trained using specified model for a specified number of episodes, with fixed seed 123. During this training period the agent will update its function approximations as well as utilize its optimistic prior/exploration function, if any. Afterwards, the agent will follow its learnt function approximations and use a greedy policy on 20 test episodes, each with a randomized seed. The final performance will be reported as average score of the 20 episodes and average survival time of the agent within the 20 episodes.

Question 2:

In order to demonstrate effects of different function approximation approaches, I chose 5 feature sets from the features listed above and demonstrated their effects in the following figures. For all feature sets, a training episode of 500 is used.

Feature set	Features Used (numbering shown above)
A	1
B	1,2
C	1,2,4
D	1,2,4,6
E	1,2,3,4,5,6,7,8
F	2,4,6





As shown in the above figures, for these particular feature sets, the overall performance of the agent doesn't seem to differ by much. Notably, feature 0 (distance from active ghosts) alone is enough to give the agent a decent enough performance; this makes sense, since the single most important thing in pacman is probably to avoid the ghosts. Even if the agent acts otherwise randomly, so long as it avoids the ghosts it can eventually get a decent score. Looking at feature set F, on the other hand, we can clearly see that removal of distance from ghosts greatly degrades the performance of the agent, regardless of the other parameters. Again, this makes sense since the agent now blindly pursues pellets without noticing whether or not it will lose life due to ghosts chasing, so it will lose life quickly considering that the red ghost will actively chase pacman.

On the other hand, the addition of more features has a notable effect of making the agent's behavior more chaotic; this is reflected by the increasing standard deviation of agent's

performance as the number of features included increases. Of course, this increase can somewhat be migrated by adding more training episodes but fixing the training episodes makes it clear that the agent becomes less sure of what actions to take as number of features increase.

Question 3:

Q-learning itself is off-policy, hence we need to give the agent a policy to follow when it applies the model learnt; typically, a completely greedy policy will be used, such that the agent will try at all cost to maximize its Q. Such policy however, if used during training, will prevent the agent from trying anything that it hasn't tried before, therefore quickly force the agent to follow a constant path repeatedly. To combat this issue, we arbitrarily assign some exploration functions to encourage the agent to explored unvisited states. The two exploration functions are given as following:

The first one, perhaps also the more common one, is the epsilon-greedy policy. This policy is mostly the same as above described greedy policy, except that instead of always following a path towards max Q, the algorithm entails a small possibility, epsilon, that the agent will do a random action instead. The magnitude of epsilon then determines how often such random action is taken; e.g., a epsilon of 0.05 means that for every action there is 5% chance that the agent will, instead of find an action that gives maximum Q, do a random action instead. This way, we ensure that 1) the agent can keep pursuing a previously learnt good path, since a fair assumption is that a good path will likely lead to a good outcome, but also 2) that the agent attempt from time to time new paths, since perhaps there exist an even better path, just happen not to be stepped one during the first few trials.

Another approach is to assign an optimistic prior to state-action pairs. That is, for each state-action pair, depending on the amount of times it has been visited (N[s,a]), we assign different values to it on top of its Q value. The prior function used in this algorithm follows the same as in the book, such that:

$$Q(s,a) = f(Q(s,a), N[s,a])$$
$$= \{Maximum\ Q, if\ N[s,a] < cutoff \mid Q(s,a), if\ N[s,a] \geq cutoff\ \}$$

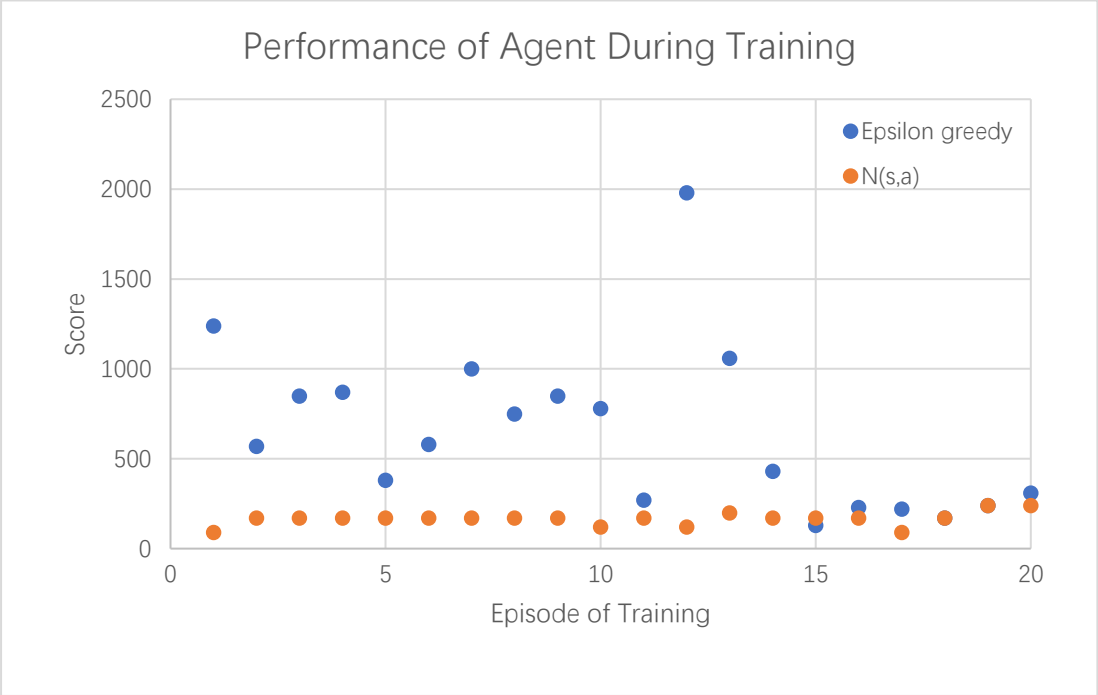
Essentially, the agent follows the same greedy-only policy described above; however, for any state visited less than cutoff amount of times, those states' Q values are treated as if they were at the highest, hence the agent will attempt to visit those states. The rationale behind this is that we would like to give all states enough trials before telling whether or not we should stick to a certain path. This approach applies a much more aggressive exploration function comparing to epsilon-greedy, hence imposing some demands on the length of training episode, as detailed in the next paragraph.

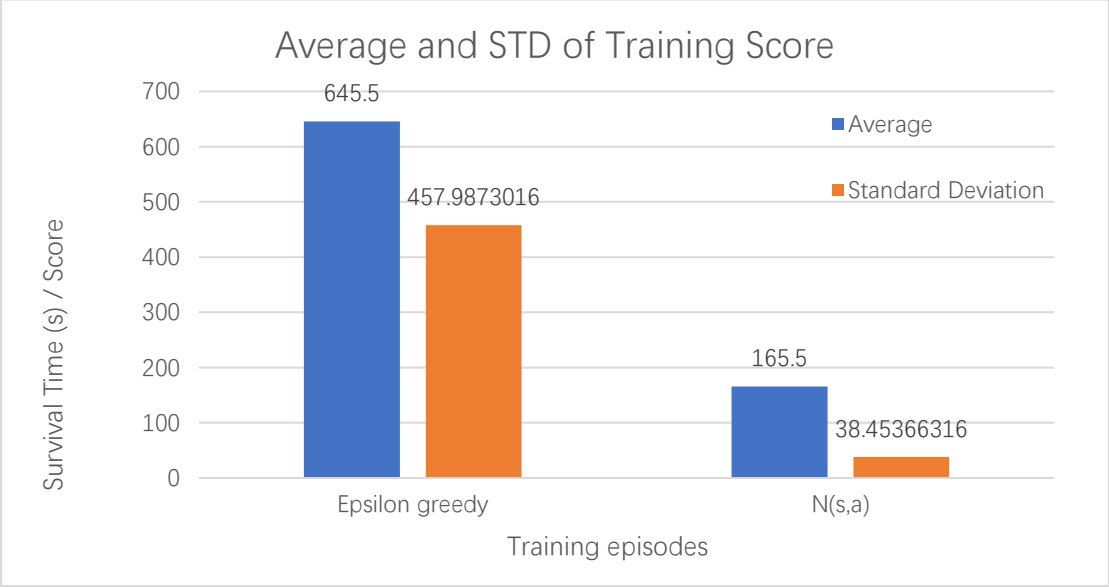
Here, cutoff is an adjustable hyperparameter, and is adjusted depending on the amount of training episodes. A higher cutoff will allow the agent to identify smaller differences between two states, therefore being more specific and potentially learn better, at the cost of increased computational times. Further, applying a high cutoff at smaller training episodes might completely disturb the learning: if the agent does not visit any given state enough amount of times, it might simply register all states as at maximum Q.

Overall, for this approach I generally use cutoff = 10 times for training episode length of less than 1000 and increase to 50 times for training episodes more than 1000. These numbers were generated by trial and error; the rate at which times a state is visited reaches cutoff has been monitored for a few cases and judging from the number of features currently in use (8), the above numbers were generated.

Question 4:

In order to demonstrate the difference in two exploration functions described above, I will show the performance of agents during the training phase, where the exploration functions impact the agent's choice of decisions; here, N(s,a) refers to the optimistic prior method described above.





Consequences to agent behavior:

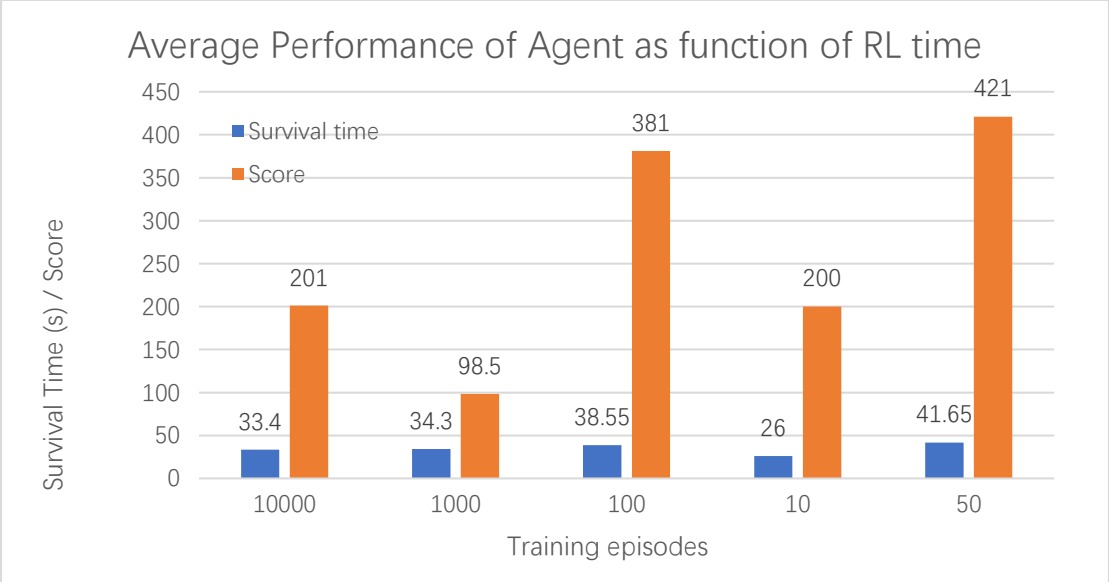
As seen in the graphs, when using epsilon-greedy policy, during training the agent performs significantly better than an agent using optimistic prior (both algorithms described above). Further, epsilon-greedy yields a much larger standard deviation comparing to optimistic prior, but even its worst instances are somewhat comparable to the average performance of the agent with optimistic prior. This is expected as using optimistic prior, the agent will attempt to explore as many states as possible such that all of them have been visited predetermined many times before moving on to follow a policy to maximize Q values. Meanwhile, agent with epsilon-greedy is, mostly, greedy, so it was trying to maximize Q values during the learning.

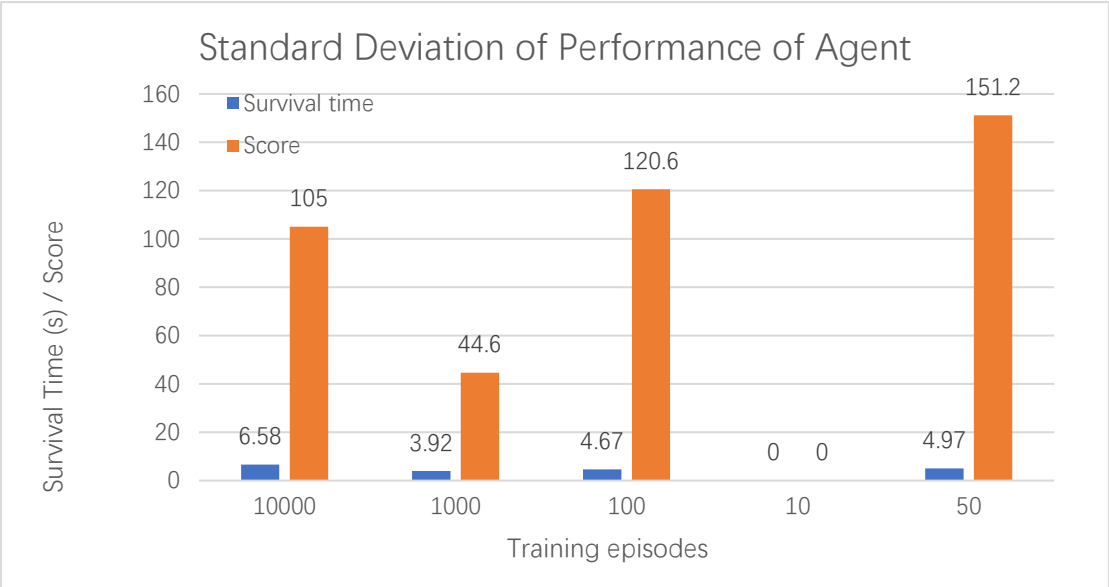
Another interesting consequence of using an optimistic prior with too large a cutoff comparing to its training episode length can be found in the figure for Q5, where for a training length of 10, the standard deviation for 20 test trials is completely 0 despite each game having different randomly generated seeds, accompanied by lowest score and survival time indicating that the agent likely spent too much time exploring during training phase (because it could not visit each state enough times to pass cutoff limit) and practically didn't learn any model of the world (due to all states having maximum Q value), hence was unable to move much away from initial environment.

Question 5:

As explained in the performance evaluation methods section, the evaluation is done through randomly generated seeds. **Hence, a performance from such evaluation which is comparable with that achieved through the learning episode (with fixed seed) will demonstrate that the agent indeed learnt good generalization of the game, instead of overfitted onto a particular gameplay pattern.** Further, I reported standard deviation of the performance results. As expected, a good generalization will feature low standard deviation in terms of both score achieved and survival time.

The performances are reported in the following figures. For these figures, an agent with optimistic prior with cutoff (as described above) is used.





As seen in the figures, interestingly, the agent's performance does not necessarily become better as training episodes become longer. Rather, it seems that they peaked at a training episode of 50 and started decreasing upwards until at some point between 1000 and 10000 that it started increasing again. The survival time followed the same pattern.

When it comes to consistency of performance, except the notable outlier of 10 trainings (which has been discussed in Q4), the standard deviation for both time and score decreased as training episodes increased, **showing that the agent is becoming more and more consistent with its model as more game events happen**: it's increasingly certain that closer distance with ghosts likely leaded to death and closer distance to pellets likely leaded to points. Of course, this itself was not enough to devise a good policy (which it didn't, indeed) but it nevertheless make the Q-values converge to a limit. Interestingly, after 10000 trainings the models start diverging again; this divergence however also corresponded to another increase in performance; my hypothesis for this behavior is that at some point between 1000 and 10000 trainings **the agent encountered some new states by chance, especially likely due to the randomness in the purple ghost's rather random movements, and found a yet better model to accommodate those states, hence "restarted" the learning.**

Reference:

[1] https://github.com/anassinator/ms-pacman/