Practical Exercise 3 - RL with Function Approximation

Matan Weksler – 302955372 Ido Glanz – 302568936

2 - Preliminary: The Mountain Car Problem and feature engineering

1. The systems' state and action space and reward function as we understood them are as follows:
   a. **State space** - a tuple of [position, velocity], indicating both the position of the cart (in the range of -1.2 to 0.6) and the velocity (ranging from -0.07 to 0.07). position and velocity are both continuous in the above range. The goal state for example is [position = 0.5, any action]
   b. **Action space** - a discrete integer in the range [0, 1, 2] indicating: [0=left, 1=neutral, 2=right]. The actions are then added to the current speed (subtracting the acceleration/deceleration caused by gravity).
   c. **Reward** - a function of state (in practice only dependent on position) equaling 1 if in terminal state (position >= 0.5) and 0 otherwise.

2. **Value Function Approximation -** The advantages in encoding the state vector as an RBF (as we understood it), is that we are allowing for more complex and diverse function approximation not necessarily linearly dependent on the state vector. Specifically, RBFs map the (standardized) deviation of the state (or absolute distance) from centers we initialize and in this manner, we can generate many RBFs with different "centers" (or kernels) whereas each could later be weighted as to its "importance" to the value of the state. In control problems such as ours, these types of functions could in a way describe our distance or error to a specific goal (in either of the state variables) and hence prove to be useful.

3. **Data Standardization** - generally, standardization allows us to "treat all variables equally" in the sense they are scaled and un-biased. This is useful when using RBFs as their kernels could be chosen with $\mu = 0$ and $\sigma = 1$ (or around those values). Not all RBFs will necessarily use this mean and variance but they will nevertheless be proportional to each other and thus equally weighted.

3 - LSPI

1. **Episodic update rule –** To our best understanding, in the LSPI method samples are selected randomly at across the distribution of possible states and actions (not necessarily uniformly). In we wanted to adapt it to an episodically samples method we could sample a random state and from there randomly selected action up until hitting a terminal state (which would generate a reward of 1) and then randomly initialize a new starting state. To be honest, we used a completely IID method, sampling in random both states and actions acquiring data. As we sampled uniformly, terminal states were also visited and as the action space was finite and the state continues and smooth (hence neighboring states were alike and we didn't need a very high resolution

of sampling) we obtained enough terminal states to allow the model to converge and learn a proper model.

2. **Data collection process** - we sampled the environment randomly using the sample methods of the space class (sampling randomly both action and state samples uniformly across the action and state space). Basically, each sample was taken i.i.d and stored in a data-set class (containing state, action, reward and next state). After testing we found 10K samples were enough to converge and in which around 500 sampled states had positive reward.

3. **Deciding on Q-function features** - To be honest we tried many different representations, first without using RBFs, then incorporating them but only centralized around 0, only to find out none were good enough. We then tried a different approach and generated a grid of position by velocity grid, taking each junction as an RBF kernel (generating, depending on grid resolution, between 4 to even 36 different kernels. This proved better than before but still no better than 30-40% success rate. At this point we decided to think of the features again, in a more physical approach and tried out features that seemed to us would affect the policy. We tried looking at the distance from the end position as well the sign of the speed (which makes some physical sense if you think of a policy of accelerating always in the direction of the speed, hence generating maximal oscillations). This proved to work well and our feature vector finalized to: [Gaussian RBF of the position around the end position, 0.6 and variance of the data, speed^2, position^2, sign(speed), 1 (bias)]

   As we used a set of weights for each action (3 in total), and each feature vector is of size 5, we have a total of 15 weights in the model.

4. **Testing the model** – See below plot of the average success rate as function of learning iteration (testing of 10 initial states and averaging over 5 complete runs)
5. **Training** – Our stopping criterion is based on the change in the weights vector, as so once the absolute change (second norm) of the weights is less than e-5 we terminate the process.

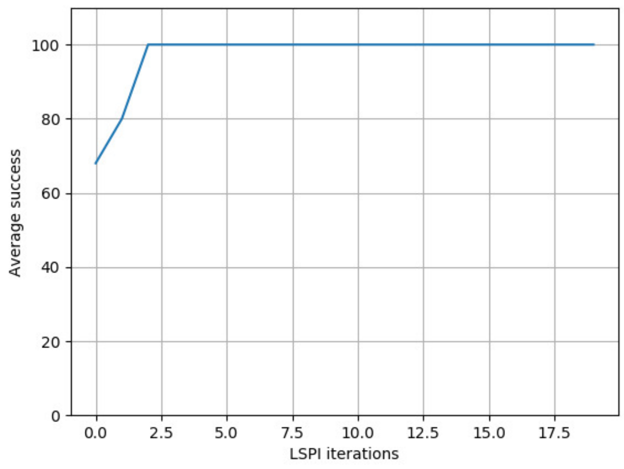

Figure 1 - Average success over number of iteration. As can be seen, after 2 iterations on average we reached 100% as defined in the question (9/10 games)

6. **Sample size of LSPI –** Assessing the needed sample set needed to assure convergence. Oddly 1000 samples did the job while 5K didn't, this is to our understanding due to the stochastics in the data sampling (potentially the 1000 samples might have held diverse enough samples to build a proper model while 5000 didn't). That said, for 10K we observed good results almost always.

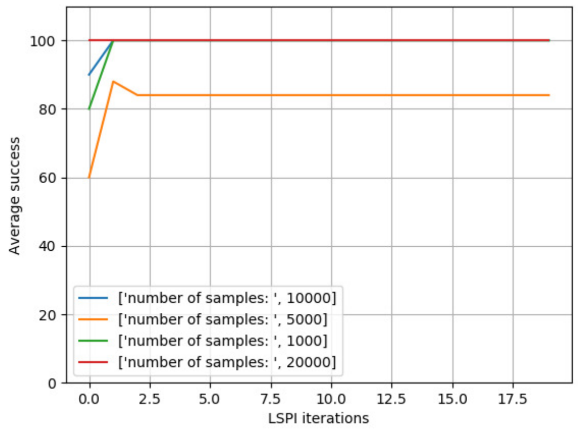

Figure 2 - Average success rate vs the number of LSPI iterations

4 – Q-learning

1. **Online data collection with ϵ-$greedy$ policies –** we defined a probability vector of going greedy or selecting in random a different state, thus if we have 3 actions in our game the vector would be $[P(1), P(2), P(3), P(greedy)]$, and P(greedy) would decay by the square root after each iteration (normalizing the vector after each expansion). In this manner, at the first iteration, we would go greedy with the probability of 0.5 and w.p. 0.25 we would choose the two other options (actions). In each training cycle, we ran up to 50 games.

2. Training:
   a. We used a batch size of 500-1000 samples and only 1 update step in each training cycle.
   b. In each training cycle, the model uses approx. 1K samples which are about 1% of the total experience accumulated along 50-100 training cycles.
   c. We need to run multiple training cycles as in each one our data-set (or experience) is limited to the number of games we played and the policy which we currently follow whilst obtaining it. In short, data sampling is w.r.t to the current value function (to the extent of epsilon-greedy) which isn't necessarily correct nor allowing too much exploration.
   d. We used a learning rate which correlates to 0.5/batch_size. The batch size is to normalize the update with the number of samples if used, and 0.5 is an empirical scalar which we found converged better (strictly empirical).

3. Testing: See Plot below

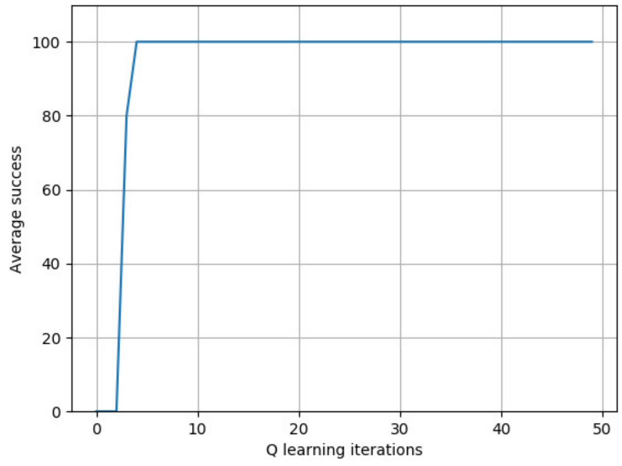

Figure 3 – Average success of number of iterations. Though it reached 100% in this run, in many runs it was very unstable and reached only 80%.

Generally, we observed very unstable results in the algorithm and though in the above graph we reached 100%, in other cases we only converged to 80%. To our understanding the algorithm is very sensitive to the states it randomly choses and the stochasticity of epsilon allowing it to explore.

4. Different ϵ:

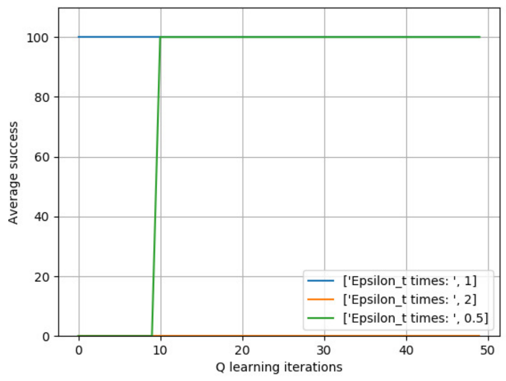

Figure 4 – testing multiple epsilon schedules (twice and half what we originally used)

When using half the epsilon we originally configured, after 10 iteration we also converged to a 100% success but when doubling the value, apparently the sampling was too stochastic and the algorithm couldn't converge.

5. Sample efficiency:
The average number of samples we observed were needed to converge were:
    Q-learning – 50*10*10 = 5K samples
    LSPI – 10K samples
Whilst the LSPI algorithm assumes it can sample enough points to represent the environment and that each can reach the terminal state, the Q-learning method uses batches of data created from played games and following policies (to the extent of epsilon greedy) hence.