Wet3 - 046203

Zohar Rimon

30.06.2020

1 Preliminary: the Mountain Car Problem and Feature Engineering

1.1

The action space is gym.spaces.Discrete(3), which means that the action is discrete and can be 0, 1 or 2. Further inspection of the step function reveals that the action is specifying the force from the cars wheels. For a=0 we get a constant force left, a=1 we get 0 force and for a=2 we get a constant force right (the force applied is a property of the environment).

The observation space (the states that the agent will see in order to decide which action to take) is gym.spaces.Box. This means that the observation space is the x location of the car and the speed of the car (as they called the gym.spaces.Box with the interval of the location and the velocity). Due to the fact that the observation space is injective to the state space (the pixels of the game), we can consider the observation space as the state space.

By inspecting the step function we can see that reward=float(done), so we a reward of 1 iff we have reached the flag and zero otherwise.

1.2

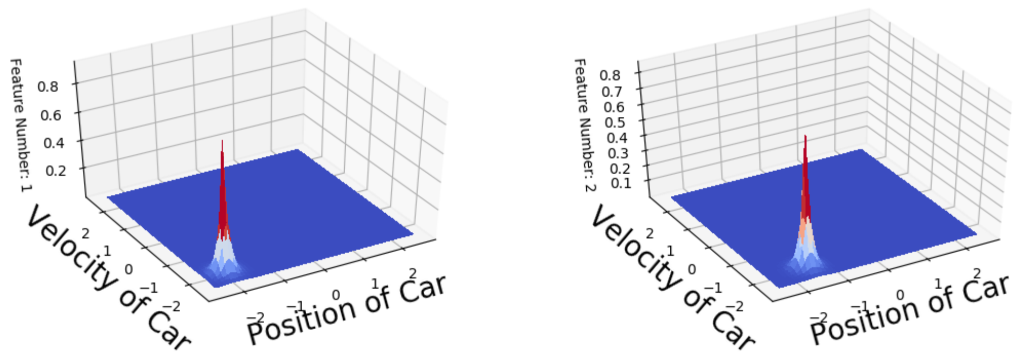

Figure 1: The first and second features of the different states

As we can see, the difference between the features is the location of the peak of the Gaussian.

1.3

Usually, features are used in order to reduce the dimensionality of the problem, Here we can see for example a case where there are more features than dimensions, so this is not the case here.Here the features help to insert non-linearity into the policy, if we would have used the states "as is", the policies that we could have learned would have been too simple to solve the problem. Further more, the features can also help us encode prior knowledge of the problem in order to achieve a simpler form of the value function. Here, obviously, RBF is suitable for the job, because we are interested in the distance of the cart from certain points in the state space. For example, the end position is in the point: (end position, 0), so encoding the distance from it as a feature can be very helpful.

2 LSPI

2.1

Let's look at the minimization problem for the PBE:

$$\theta^* = \underset{\theta \in \mathbb{R}^k}{\operatorname{argmin}} \left\| \Phi\theta - (R^\mu + \gamma P^\mu \Phi \theta^*) \right\|_\epsilon^2$$

From this equation we got the solution for the optimal theta:

$$C\theta^* = d$$
$$C = \Phi^T \Xi \left(I - \gamma P^\mu\right) \Phi, \quad d = \Phi^T \Xi R^\mu$$

And then, we used samples in order to estimate those values. Now, for an episodic environment, for terminal state we want the optimal θ to be:

$$\theta^* = \underset{\theta \in \mathbb{R}^k}{\operatorname{argmin}} \left\| \Phi\theta - R^\mu \right\|_\epsilon^2$$

So the LSPI algorithm's update step becomes:

$$\hat{d}_n = \frac{1}{N} \sum_{i=1}^{N} r\left(s_i, a_i\right) \phi\left(s_i, a_i\right)$$

And (notice that we do not look at the next step for terminal states):

$$\hat{C}_n = \frac{1}{N} \sum_{i=1}^{N} \phi\left(s_i, a_i\right) \left(\phi^\top\left(s_i, a_i\right) - \gamma \phi^\top\left(s_i^\prime, \pi_{\text{greedy}}\left(s_i^\prime; w_i\right)\right) \mathbf{1}\left(s_t \text{ is not terminal }\right)\right)$$

2.2

Mean position: -0.3
STD position: 0.52
Mean velocity: $7 \cdot 10^{-5}$
STD velocity: 0.04

2.3

The number of weights is the same as the length of $\phi(s, a)$, which is in our case $|A| \times |\phi(s)|$. Due to the fact that $|A| = 3$ and $|\phi(s)| = 120$, we get 360 weights. Further more, for each action we also include a bias term, so we get a total of 363 weights.

2.4

CODE

2.5

I have plotted the mean performances for 5 different random seeds in order to get a good estimation of the true mean.

2.6

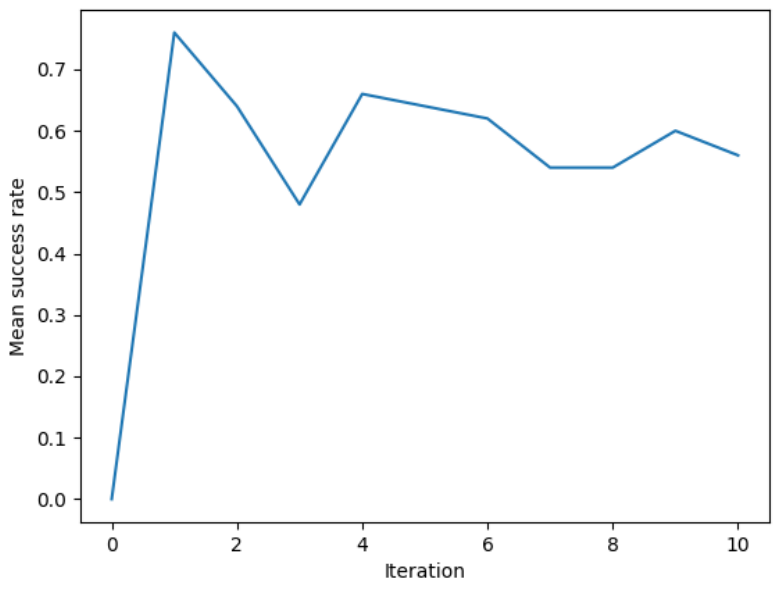

Figure 2: Mean success rate of the different random seeds at each iteration

We can see in Figure [2] that after some iterations most of the seeds converged but we did not achieve an optimal result. This is due to the fact that we only used 100K samples. We will soon see that for more samples we will get better results.

2.7

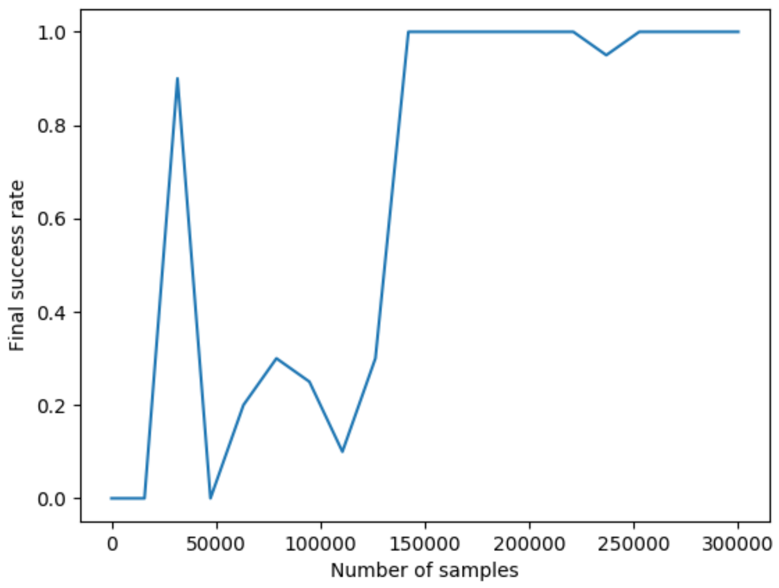

Figure 3: The final success rate of one of the seeds with different number of samples

We can see from Figure [3] that with sufficient number of samples we will converge to the optimal policy. The high variance we can see in the graph is due to the fact that we only took one random seed.

3 Q learning

3.1

We set the reward of ”goal not reached” to -1 and the ”goal reached” to 100. Thus, in order to get a cumulative reward of more than -75, the car should reach the goal in less than 175 steps.

We note that the positive value of the reward at the goal state does not effect the optimal policy. The optimal policy will always be the policy that reaches the goal state in the shortest path. Setting the reward at the goal state as relatively high positive value instead of a 0 helps the optimization process. Remember that the update step in online Q learning is:

$$\theta_{n+1} = \theta_n + \alpha_n \left(r\left(s_n, a_n\right) + \mathbf{1}\left(s_t \text{ is not terminal }\right) \max_a f\left(s_{n+1}, a; \theta_n\right) - f\left(s_n, a_n; \theta_n\right) \right) \nabla_\theta f(s_n, a_n, \theta_n)$$

So for larger rewards we will get a bigger update step. This will result in a larger attraction of the parameters in the direction of the large reward.

3.2

CODE

3.3

We will plot the results of the different seeds on separate graphs because each seed took different number of iterations to converge. Also, due to the short run time of each seed, I averaged the bellman error only on the last 10 iterations, and not 100.

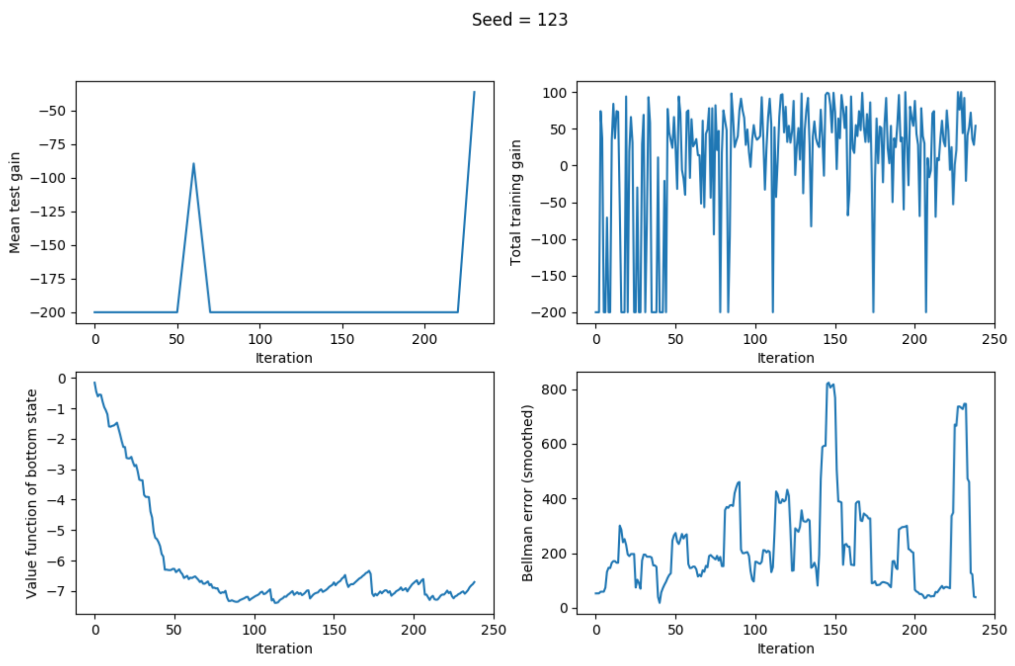

Figure 4: Four graphs for seed = 123: The mean test gain, total training gain, Value function of the bottom state and (smoothed) Bellman error, all vs iteration

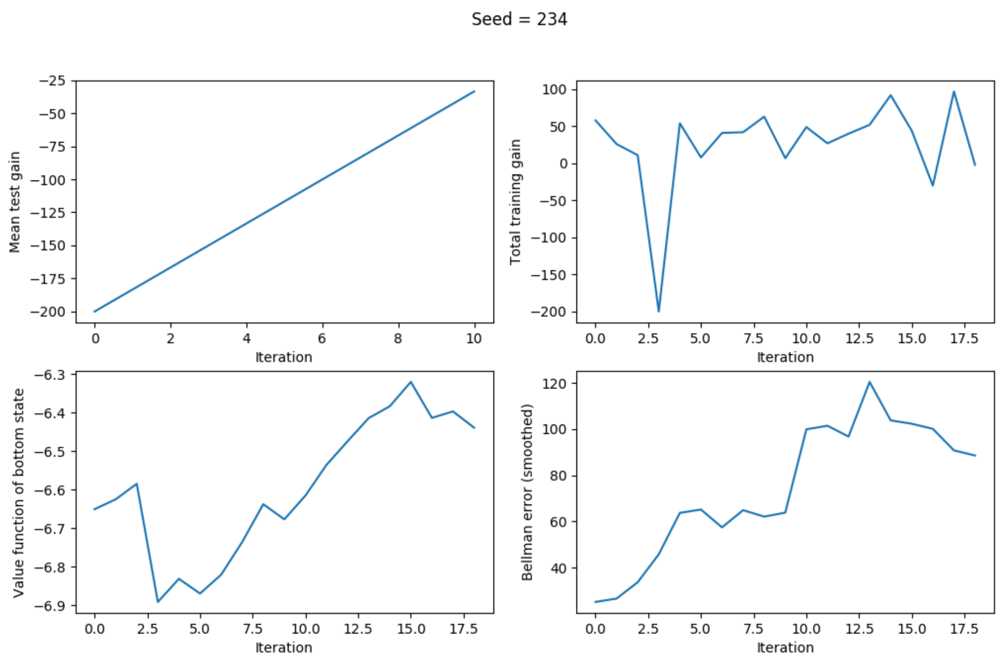

Figure 5: Four graphs for seed = 234: The mean test gain, total training gain, Value function of the bottom state and (smoothed) Bellman error, all vs iteration

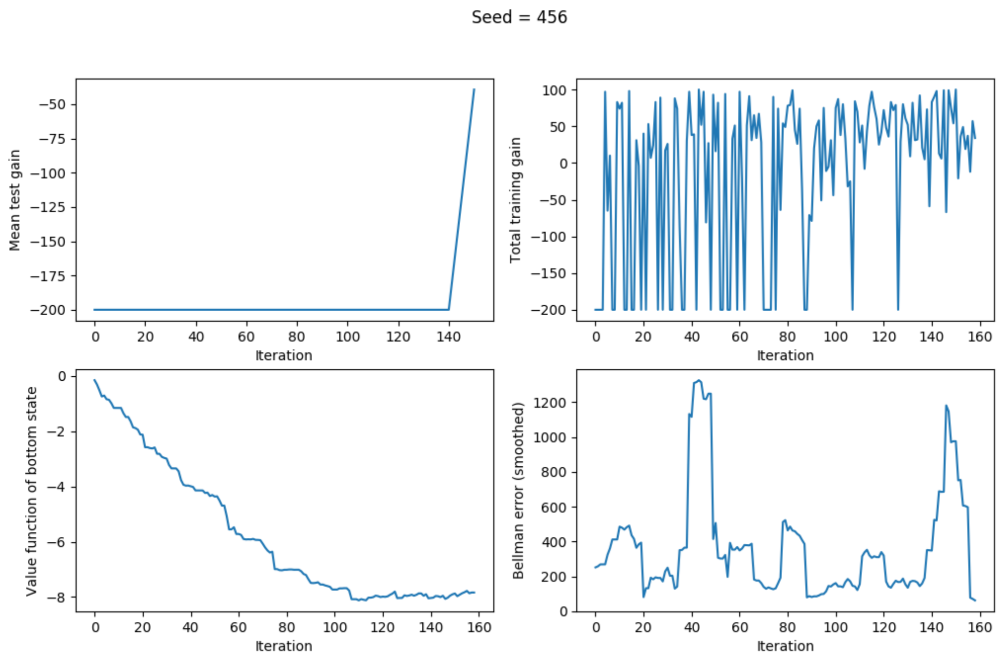

Figure 6: Four graphs for seed = 456: The mean test gain, total training gain, Value function of the bottom state and (smoothed) Bellman error, all vs iteration

Now, we will explain the results that can be seen in the four graphs:

- Mean test error: We can see that the mean test gain is at the lowest possible value (-200) at the beginning of the run. Once the agent learns how to get from the bottom of the hill to the goal, the gain jumps to a high value.
- Total training error: We can see two phenomenons, one is that the average is increasing, which is intuitive since the agent learns how to achieve higher rewards from more states as the training progress. The other phenomenon is that the variance of the rewards is very high, this is due to the fact that the initial state is very stochastic (we can observe that we do not get this high of a variance in the

testing phase for example). Due to the high variance in starting state, we get states that the agent already learned (and get high reward) and ones that it didn't see (and get low reward).

- Value function of the bottom state: Apart from seed 234, which was a very short run, we can see convergence of the value function at the bottom of the hill (with velocity 0). We expected convergence, since we proved that the maximum of the difference between the optimal approximated value function and the algorithm's value function, is bounded from above as we increase the number of iterations: $\limsup_k \max_s |V^{\mu_k}(s) - V^*(s)| \leq \frac{\epsilon + 2\gamma\delta}{(1-\gamma)^2}$
- Smoothed bellman error: We can see that the bellman error is very noisy and has big "peaks", this is due to the nature of the update step. When we reach a terminal state, the "done flag" parameter will cause a big bellman error. This big bellman error is very important because it is what teaches the agent about the high reward at the goal (what I talked about in section 3.1)

3.4

We can see in Figures [4, 6], that the value function at the bottom of the converges and the greedy policy is able to solve the problem after about 100-200 iterations. We can also see that the value that the value function converges to makes sense (small negative number). This is probably due to the fact that the features in this case are highly representative and the optimal approximated value function is close to the real optimal one.

On the other hand, we can see in Figure[5], that the value function did not converge even after the greedy policy was able to solve the task. This is due to early stop of the algorithm, where the algorithm succeeded to get to the goal state by chance and we stopped the algorithm before it got to a real convergence,.

3.5

We checked the algorithm behaviour for different exploration coefficients (for the epsilon greedy exploration):

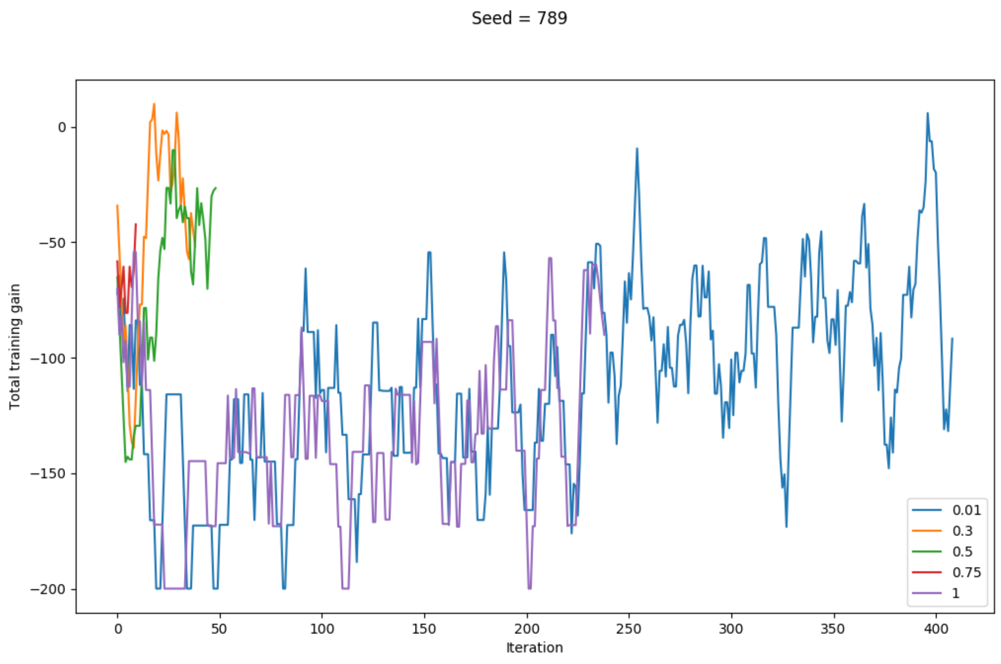

Figure 7: The training gain of the algorithm for different exploration coefficients

We can see that for too small epsilons, the algorithm can't converge, due to lack of exploration, the agent doesn't see the goal state during the training often enough. On the other hand, for too large epsilon, the algorithm doesn't work optimally as well, and converges after a while on a sub-optimal policy. This happens due to the fact that for high exploration we get low exploitation, the agent saw the states that lead to the goal but it continues to take random actions instead of being greedy.
The best results are for epsilon = 0.3 or epsilon = 0.5

4 Bonus

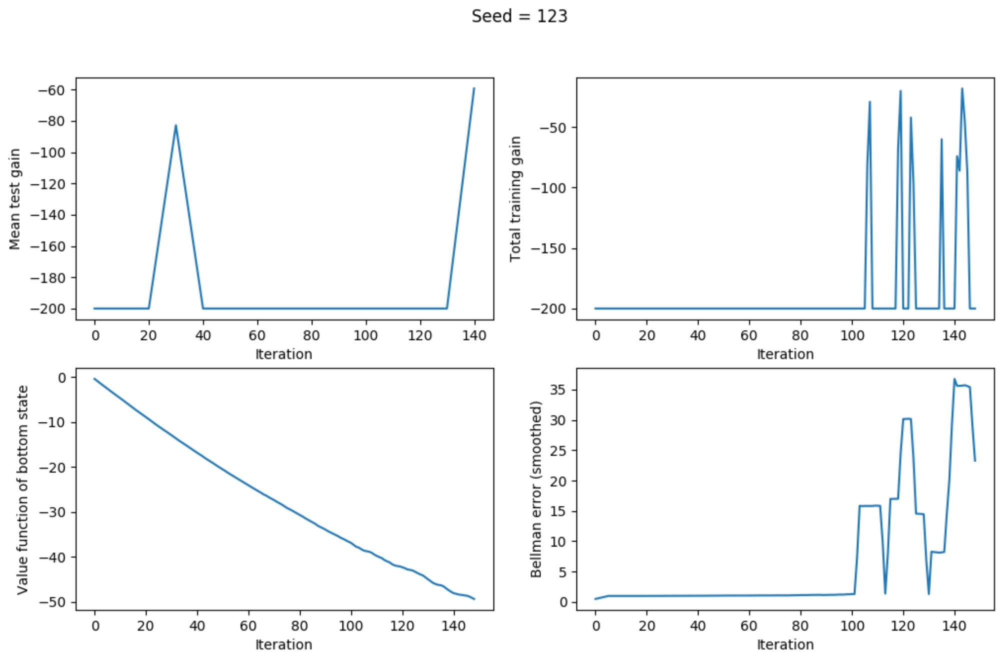

Figure 8: Four graphs for seed = 123 when always starting at the bottom of the hill and with the default exploration (epsilon = 0.1): The mean test gain, total training gain, Value function of the bottom state and (smoothed) Bellman error, all vs iteration

As we can see in figure [8], the algorithm still succeeds, even when trained only from episodes starting at the bottom of the hill and with velocity 0.

One (me) might think that this cannot happen due to the exploration problem. That is, if we would only choose random actions, the chance of reaching the goal state is decreasing exponentially with the distance between the goal and the bottom of the hill, thus will never learn about the big reward at the goal.

The reason that this is not the case is the initialization of the parameters of the policy. Due to the fact that we initialized the parameters to small values, we tough our agent the concept of ”curiosity”. That is, the initial values of the Q function are larger than the negative values of the reward the agent gets until he reaches the goal. Thus the agent will try to get to the states that he didn't see before (and understood that are bad).

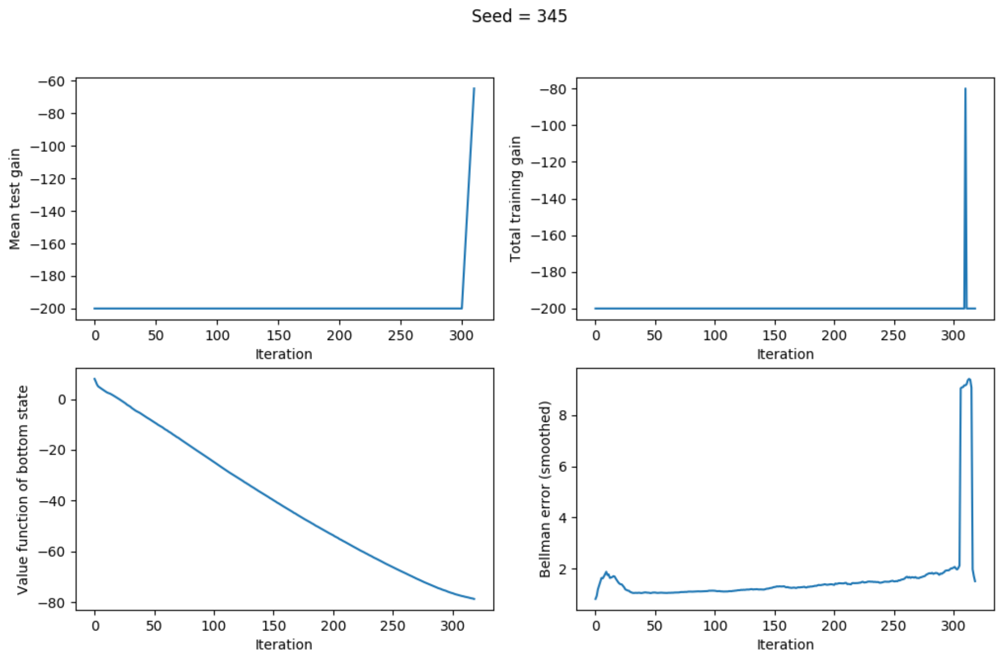

Figure 9: Four graphs for seed = 123 when always starting at the bottom of the hill and with the default exploration (epsilon = 0.1) and bad initialization of theta: The mean test gain, total training gain, Value function of the bottom state and (smoothed) Bellman error, all vs iteration

We can see in Figure [9] that after initializing the theta to larger value, the agent did not explore efficiently, and did not reach the goal until very late in the training process (thus took much longer to converge).
One way to fix this problem is to introduce other exploration mechanisms (such as explore first), while keeping in mind that things like larger epsilon in epsilon greedy will not work due the exponentially declining chance of reaching the goal state in this manner.