RL-4

Abhilash Jain

November 2019

1 Task 1

Implement Q-learning using function approximation, at every timestep using the latest state transition to perform a TD(0) update. Test the implementation on the Cartpole environment. Test two different features for state representations:

```
def single_update(self, state, action, next_state, reward, done):
        # Calculate feature representations of the
        # Task 1: TODO: Set the feature state and feature next
    state
        featurized_state = self.featurize(state)
        featurized_next_state = self.featurize(next_state)


        # Task 1:  TODO Get Q(s', a) for the next state
        next_qs=[q.predict(featurized_next_state)[0] for q in self.
    q_functions]
        if done:
            next_qs = np.zeros(len(next_qs))
        # Calculate the updated target Q- values
        # Task 1: TODO: Calculate target based on rewards and
    next_qs
        target = (reward+(self.gamma*np.max(next_qs)),)

        # Update Q-value estimation
        self.q_functions[action].partial_fit(featurized_state,
    target)
```

Listing 1: Task 1

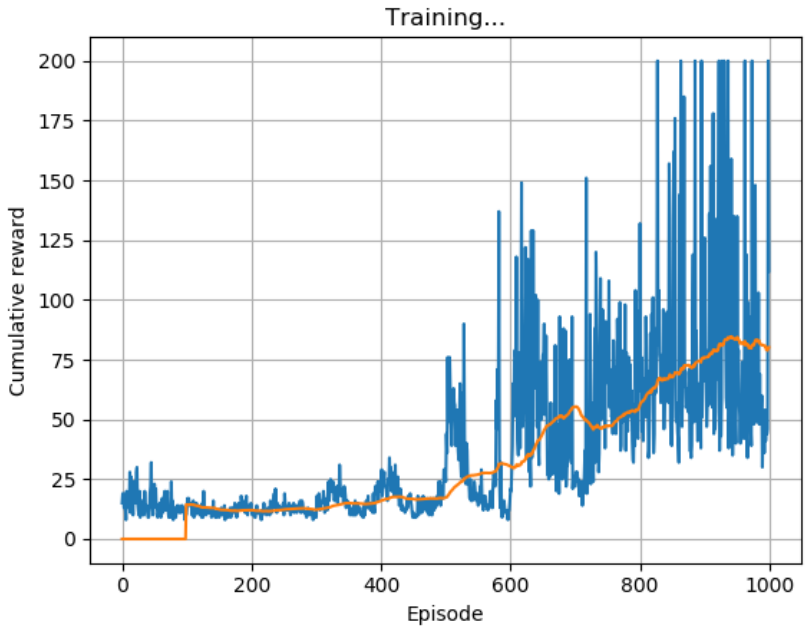
The code can be found attached for Task 1

```
def featurize(self, state):
        if len(state.shape) == 1:
            state = state.reshape(1, -1)
        # Task 1a: TODO: Use (s, abs(s)) as features

        abs_feat=np.absolute(state)
        new_feat=(np.append(state,abs_feat,axis=1))
        #return new_feat
        # Task 1b: RBF features
```

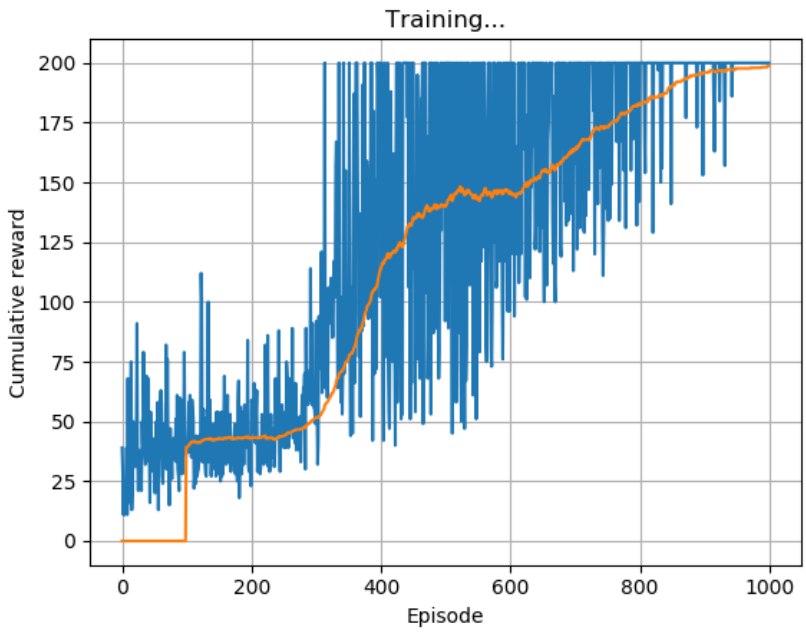
```
10          return self.featurizer.transform(self.scaler.transform(
    state))
11
```

Listing 2: Task 1a and 1

(a) handcrafted feature vector:



(b) radial basis function representations



1.1 Question 1

Would it be possible to learn Q-values for the Cartpole problem using linear features (by passing the state directly to a linear regressor)?

The Cartpole problem is a non-linear problem which implies the following: (http://underactuated.csail.mit.edu/underactuated.html?chapter=acrobot) quoted 'Cartpole requires a full nonlinear control treatment' It is trivial that the Q value should not gradually increase or decrease when you move the position X or other features as velocity, angle and angular velocity. We can then understand that Q(s,a) is not a linear function, and it cannot be presented with a linear function. In practice,We can see that from the handcrafted features,(which are linear) the performance was extremely bad. and nothing useful was really learnt. So in summary, it would not be possible to learn Q-values for the Cartpole problem using linear features

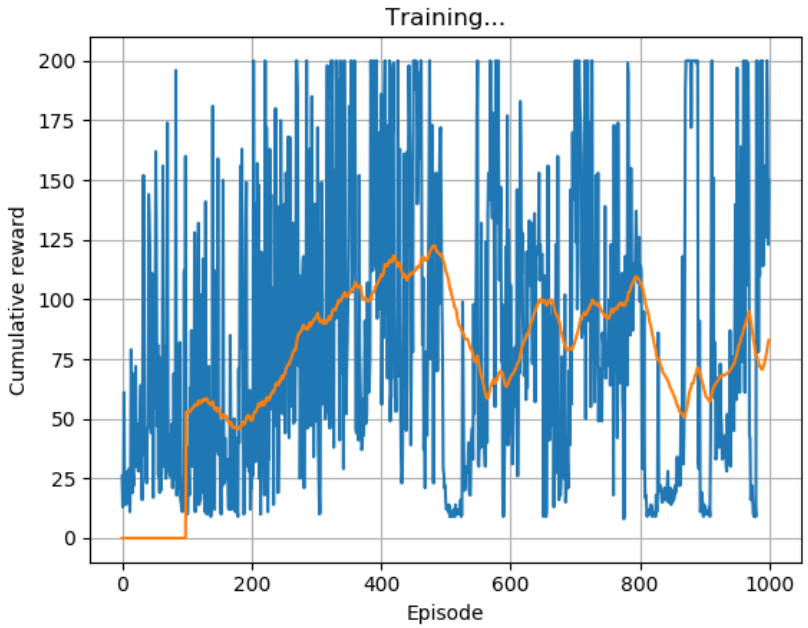
2 Task 2

Modify your Task 1 implementation to perform minibatch updates and use experience replay (while keeping the original code for Task 1 submission). Run the experiments with Cartpole with both feature representations.

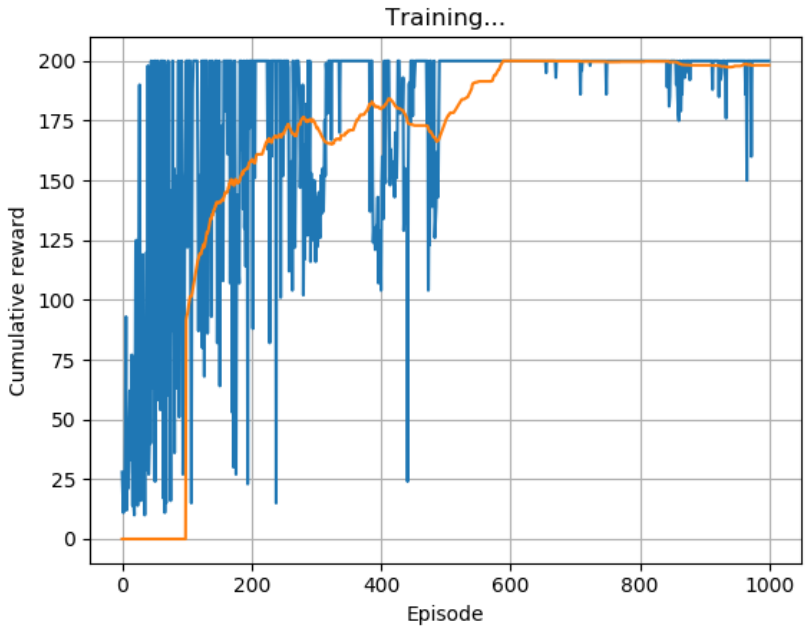
```
def update_estimator(self):
        if len(self.memory) < self.batch_size:
            # Use the whole memory
            samples = self.memory.memory
        else:
            # Sample some data
            samples = self.memory.sample(self.batch_size)

        # Task 2: TODO: Reformat data in the minibatch
        states, action, next_states, rewards, dones = map(list, zip
    (*samples))
        states, action, next_states, rewards, dones =  np.array(
    states),np.array(action),np.array(next_states),np.array(rewards
    ), np.array(dones)

         # Task 2: TODO: Calculate Q(s', a)
        featurized_next_states = self.featurize(next_states)
        next_qs = np.transpose(np.array([q.predict(
    featurized_next_states) for q in self.q_functions]))

        terminal_indices = np.where(np.array(dones) == True)[0]
        next_qs[terminal_indices] = 0

        # Calculate the updated target values
        # Task 2: TODO: Calculate target based on rewards and
    next_qs
        targets = rewards + self.gamma*np.max(next_qs, axis = 1)
```

Listing 3: Task 2

a)Hand-crafted features:



b)RBF:



2.1 Question 2.1

Which method is the most sample efficient, and why?

A method that is able to learn using only a few interactions with the environment would be more sample efficient, with this definition we can clearly see that the RBF-with experience replay converges faster when compared to any other method. This is due to the fact that Experience replay lets reinforcement learning agents remember and reuse experiences from the past and the reason for its uniqueness and why it converges faster is with experience stored in a replay memory, so it becomes possible to break the temporal correlations by mixing more and less recent experience for the updates, and rare experience will be used for more than just a single update. So even though it uses a memory buffer, it clearly uses it very efficiently, converging very fast, as we can see from the Task 2.

2.2 Question 2.2

How could the efficiency of handcrafted features be improved?

When we look at this problem we can say the the abs() function does not perform well, i.e. y=$|x|$ even though it is a non-linear function, we may require a smoother function, therefore handcrafted features could be may have been passed through a smoother function, the performance can be improved.

2.3 Question 2.3

Do grid based methods look sample-efficient compared to any of the function approximation methods? Why/why not?

Grid based methods are clearly not sample-efficient when compared to any of the function approximation methods.Function approximation methods are used when these grid-based(tabular) methods do not perform well because of large state space. Also grid-base methods update one state at a time and even though they would use less sample(1 sample) they clearly don't extract enough information for it to converge or head towards convergence.

2.4 Question 2.4

Which hyperparameters and design choices impact the sample efficiency of function approximation methods?

The hyperparameters which clearly impact are:

Learning Rate and Learning Rate Schedule: Two obvious hyper-parameters, maybe Using a annealing learning rate scheduler where the learning reduces as the training progresses.

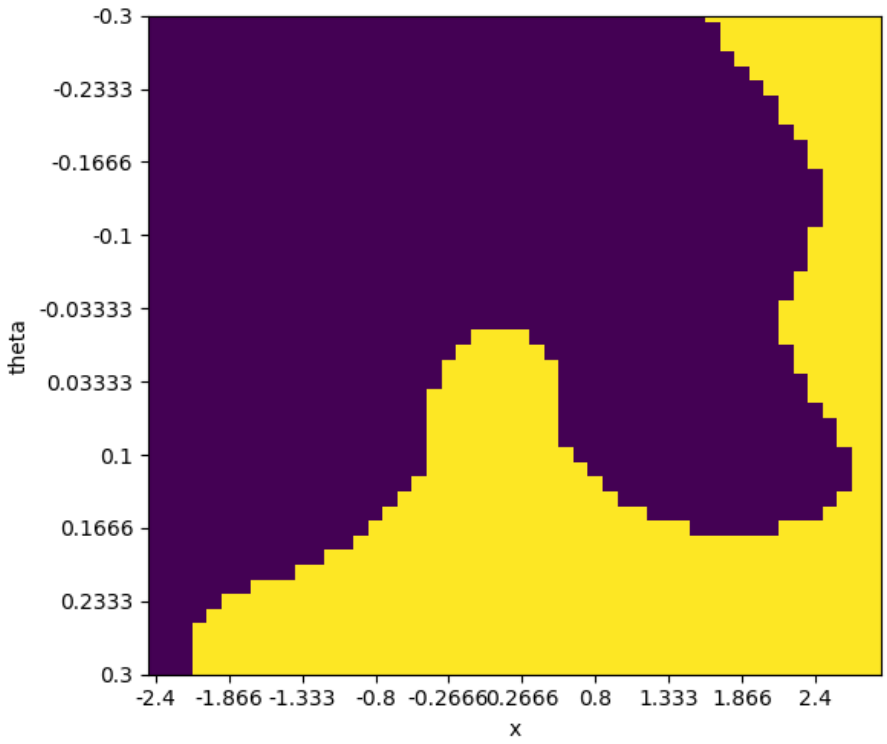
Gamma: Also a hyper parameter, Which affects the target.

The number of episodes as well is a hyperparameter

The Parameters to the RBF intializer, The obs_limits which define the observation range, the memory size, which controls the size of the relay Buffer.

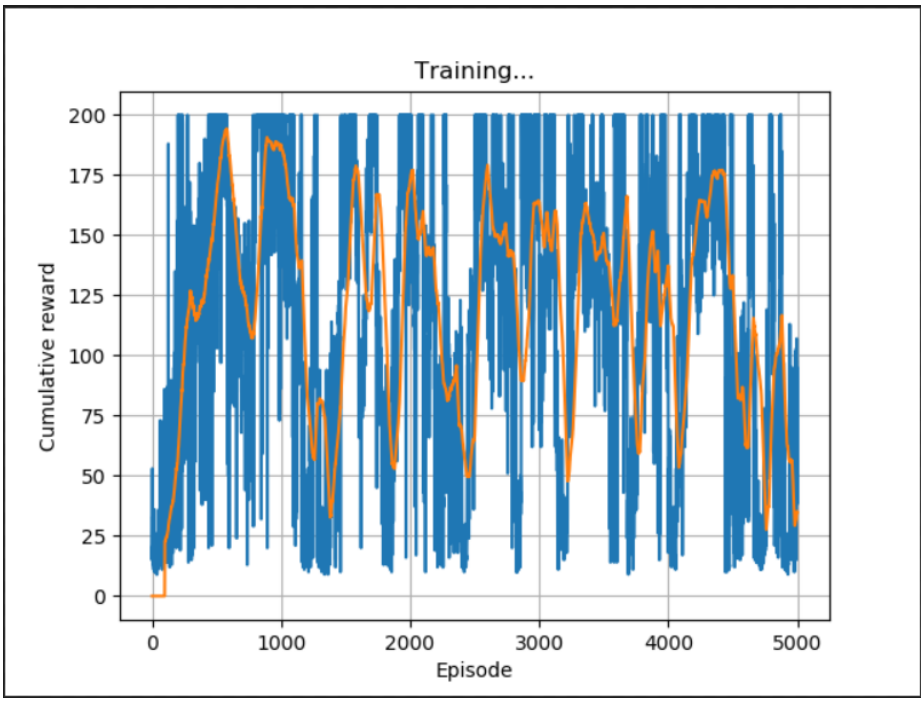
The experience replay drawing from a uniform distribution can be changed. https://arxiv.org/pdf/1511.05952.pdf - An interesting paper where they prioritize which transitions are replayed can make experience replay more efficient. This can be an interesting design choice. The feature map which we experimented with in Task 1a) and 1b) can also be an interesting design choice

3 Task 3

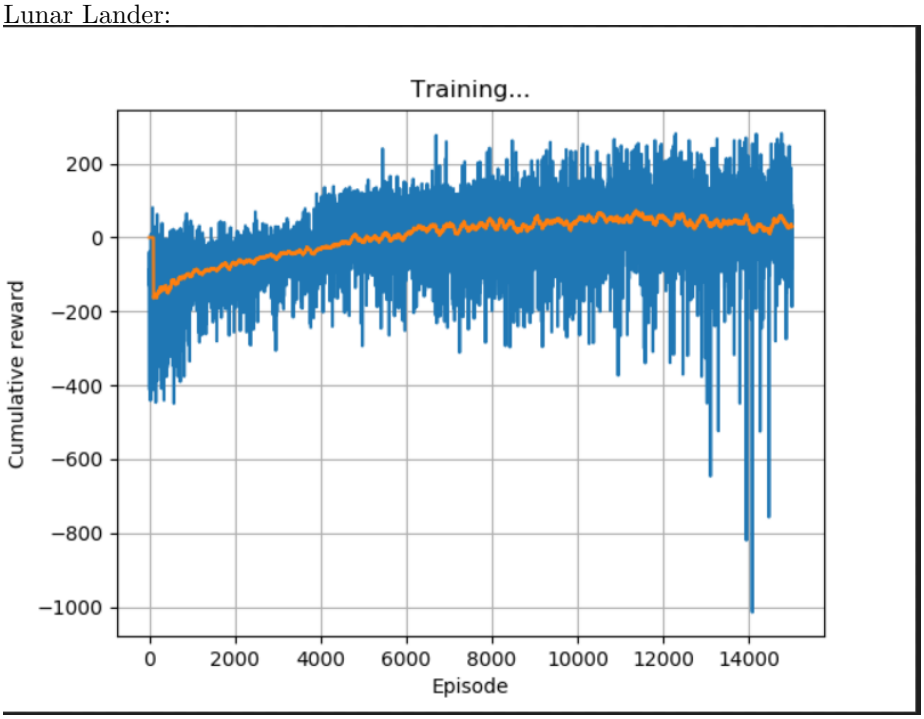


4 Task 4

Cartpole environment:



Lunar Lander:



4.1 Question 3.1

Can Q-learning be used directly in environments with continuous action spaces?

No, Q-learning cannot be used directly in environment with continuous action spaces, There are though different versions which can be used For example, https://arxiv.org/pdf/1702.08165.pdf This paper defines a soft-qlearning algorithm which can be applied with the continuos action space

4.2 Question 3.2

Which steps of the algorithm would be difficult to compute in case of a continuous action space? If any, what could be done to solve them?

The step of the algorithm where we choose an action, would be the most difficult to compute, finding the argmax (action) over a continuous function is not simple as it is for a discrete set. There-fore this is one of the area where the algorithm would fail. One way to fix this would be to some-how approximate this as well. The above mentioned paper takes an approach where, it views it as a stochastic optimization problem, we first express the soft value function in terms of an expectation via importance sampling, were they have an arbitrary distribution over the action space