Reinforcement Learning

Practical #04 - Q-Learning with LVFA

SAPIENZA
UNIVERSITÀ DI ROMA

Recap...

Clone or pull the repository with the following command

```
git clone https://github.com/KRLGroup/RL-2022.git
```

or

```
git pull
```

Info

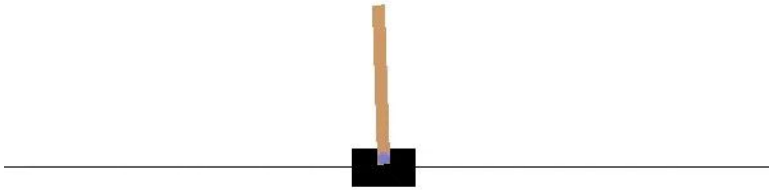
Today we will use python with jupyter-notebook. You can either use it locally or on https://colab.research.google.com/

The libraries needed for this practical are: numpy, gym and jupyter

You can install them all with pip

Exercise

Today we will try to train a cartpole agent using Q-Learning with LVFA.



Linear Value Function Approximator

We represent our state with a feature vector

$$\mathbf{x}(s,a) = (x_1(s,a), \ldots, x_n(s,a))^T$$

And consider a parametric Q function in the form

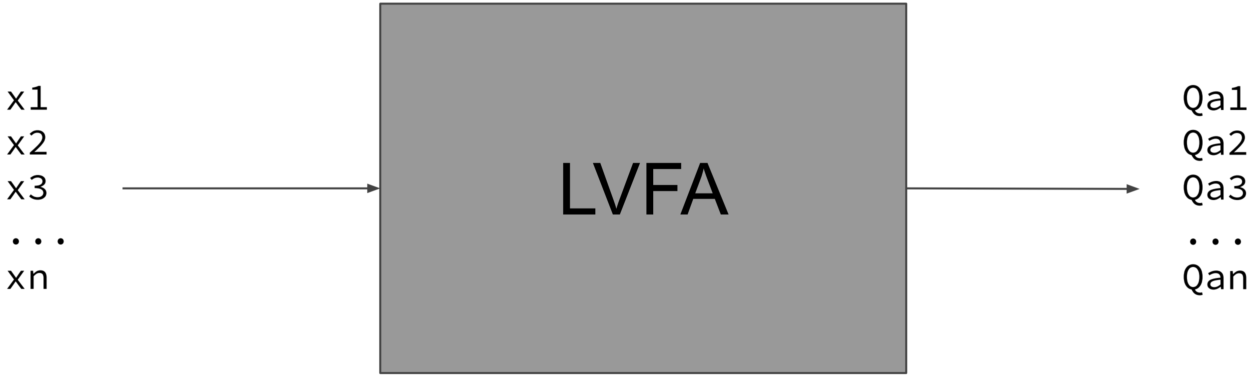
$$Q(s,a,\mathbf{w}) = \mathbf{w}^T \mathbf{x}(s,a)$$

Then

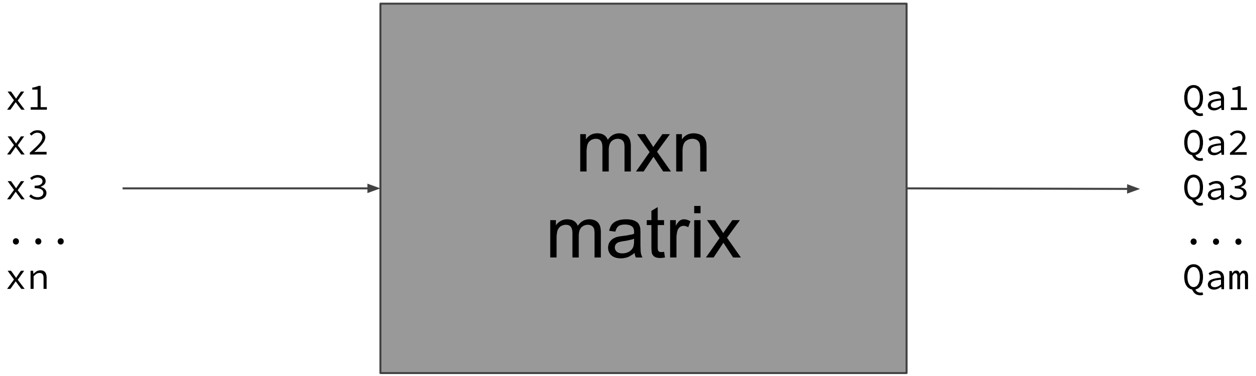
$$J(w_t) = \mathbb{E}(Q^\pi(s_t,a_t) - \mathbf{w}^T \mathbf{x}(s_t,a_t))^2$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(Q^\pi{}_t - \mathbf{w}^T{}_t \mathbf{x}(s_t,a_t))\ \mathbf{x}(s_t,a_t)$$

Linear Value Function Approximator



Linear Value Function Approximator



Linear Value Function Approximator

$$Q = WX$$

Feature Extraction

```
class VanillaFeatureEncoder:
    def __init__(self, env):
        self.env = env

    def encode(self, state):
        return state

    @property
    def size(self):
        return self.env.observation_space.shape[0]
```

Agent

```python
class QLearning_LVFA:
    def __init__(self, env, feature_encoder_cls, alpha=0.005,
                 alpha_decay=0.9999, gamma=0.9999, epsilon=1., epsilon_decay=0.9999):
        self.env = env
        self.feature_encoder = feature_encoder_cls(env)
        self.shape = (self.env.action_space.n, self.feature_encoder.size)
        # self.weights = np.zeros(self.shape)
        self.weights = np.random.random(self.shape)
        self.alpha = alpha
        self.alpha_decay = alpha_decay
        self.gamma = gamma
        self.epsilon = epsilon
        self.epsilon_decay = epsilon_decay
```

Forward Step

```
def Q(self, feats):
    feats = feats.reshape(-1,1)
    return 0
    #return ?


def policy(self, state):
    state_feats = self.feature_encoder.encode(state)
    action = 0
    # action = ?
    return action

def epsilon_greedy(self, state, epsilon=None):
    if epsilon is None: epsilon = self.epsilon

    # if epsilon_greedy return random
    # else return policy

    return 0
```

Training

```python
def update_transition(self, s, action, s_prime, reward, done):
    s_feats = self.feature_encoder.encode(s)
    s_prime_feats = self.feature_encoder.encode(s_prime)
    action_prime = self.epsilon_greedy(s_prime)
    td_error = reward


    delta_w = np.zeros(self.feature_encoder.size)
    # delta_w = ?

    self.weights[action] += self.alpha*delta_w


def update_alpha_epsilon(self):
    self.epsilon = max(0.2, self.epsilon*self.epsilon_decay)
    self.alpha = self.alpha*self.alpha_decay
```

```python
def train(self, n_episodes=200, max_steps_per_episode=200):
    for episode in range(n_episodes):
        done = False
        s, _ = env.reset()
        for i in range(max_steps_per_episode):

            action = self.epsilon_greedy(s)
            s_prime, reward, done, _, _ = self.env.step(action)
            self.update_transition(s, action, s_prime, reward, done)

            s = s_prime

            if done: break

        self.update_alpha_epsilon()

        if episode % 20 == 0:
            print(episode, self.evaluate(), self.epsilon, self.alpha)
```

Evaluation

```python
def evaluate(self, env=None, n_episodes=10, max_steps_per_episode=200):
    if env is None:
        env = self.env

    rewards = []
    for episode in range(n_episodes):
        total_reward = 0
        done = False
        s, _ = env.reset()
        for i in range(max_steps_per_episode):
            action = 0
            # action = ?

            s_prime, reward, done, _, _ = env.step(action)

            total_reward += reward
            s = s_prime
            if done: break

        rewards.append(total_reward)

    return np.mean(rewards)
```

Solutions on github

Sapienza
Università di Roma