Change in Behavior and Transfer with Change in Properties of the Environment

CB

Aditya Anantwar (19CS10006)

Abhinandan De (19CS10069)

Instructor: Prof. Aritra Hazra

September 18, 2022

CS60077

1 Introduction

This project is aimed towards analyzing the change in behavior of the reinforcement learning based system with change in properties of the environment. We initially explore some basic RL algorithms for analysis and plan on extending the analysis to study transfer between two scenarios having variable properties of the environment.

We begin our analysis on the **CartPole-v1** environment which is a version of the cartpole problem [Brockman et al., 2016]. It consists of a pole attached by an unactuated joint to a cart which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces to the left and right directions of the cart.

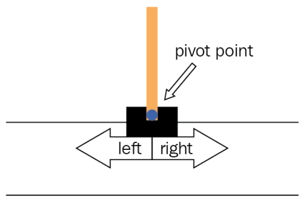

Figure 1: Open AI's CartPole environment

We simulate changes in properties of the environment by changing the physical properties of the cartpole such as mass, length and also try altering the magnitude of force and gravity. We utilize two popular RL algorithms: **DP** and **Q-learning** for analysis. In the initial phase, we train our model after altering the physical properties and observe the change in rewards generated using the same training algorithms.

2 Methods

2.1 Dynamic Programming

Dynamic Programming algorithms help solve the category of problems known as planning problems [Sutton and Barto, 2018]. Given a complete model and environment specification (complete information about the state space) dynamic programming gives an optimal policy for the agent to follow. The key idea in dynamic programming is to use value functions to organize and structure the search for good policies. From Bellman optimality equations, the optimal value function V^* and optimal policy Q^* are given as:

$$V_* = \max_a \mathbb{E}[R_{t+1} + \gamma V_*(S_{t+1})|S_t = s, A_t = a]$$

$$Q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a')|S_t = s, A_t = a]$$

2.1.1 Policy Evaluation

Policy evaluation predicts a state-value function v_π for an arbitrary policy π. The existence and uniqueness of V_π is guaranteed as long as $\gamma < 1$ or eventual termination is guaranteed from all states under the policy π. The evaluation of one iteration is linear in $|S|$. Consider a sequence of approximations V_0, V_1, V_2, ... each mapping S^+ to R. V_0 is chosen arbitrarily and each successive approximation is obtained by using the Bellman equation. This sequence is guaranteed to converge over infinite iterations with the fixed point being the optimal value function $V*$. The algorithm for policy evaluation is given in the appendix (Algorithm 2).

2.1.2 Policy Improvement

The reason we compute value function for a policy is to find better policies. Suppose we have determined a value function V_π for some policy π. To improve our policy, we would like to know whether for some state s we should choose an action a $\neq \pi[s]$. That is whether changing the policy would give a better value function than V_π[s]. One way to determine this is to choose action a in state s and then continue following the policy π. If we find that the value function improves, we need a new policy. Therefore, if choosing this action a is better than choosing π[s], we can greedily update π[s] to a to form a new policy. This is known as policy improvement.

2.1.3 Policy Iteration

Once a policy π has been improved using v_π to yield a better policy π', we can then compute $\mathrm{v}_{\pi'}$ and use it to find an even better policy π". Thus we obtain a sequence of monotonically improving policies and value functions:
$\pi_0 \xrightarrow{E} v_{\pi 0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi 1} \xrightarrow{I} \pi_2 \xrightarrow{E} ... \xrightarrow{I} \pi_* \xrightarrow{E} v_*,$
Each policy is guaranteed to be strict improvement over the previous by our policy improvement heuristic (unless it is already optimal, in which case it wont change). Because a finite MDP has a finite number of deterministic policies, this sequence must converge and the fixed

point for this convergence will be the optimal policy. This way of finding the optimal policy is called policy iteration. The algorithm for the same is given in the appendix (Algorithm 3).

2.1.4 Value Iteration

A drawback of policy iteration is that each iteration involves policy evaluation, which can be expensive computationally. If policy evaluation is done iteratively, then convergence exactly to v_π occurs only in the limit. The policy evaluation step of policy iteration can be truncated in several ways without losing the convergence guarantees of policy iteration. One important special case is when policy evaluation is stopped after just one sweep. This algorithm is called value iteration. The formula for value function update is as follows:

$$v_{k+1}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]$$

The algorithm for value iteration is given in the appendix (Algorithm 4).

2.2 Q-Learning

Q-Learning is one of the famous reinforcement learning algorithms that approximates the optimal action value function independent of the policy being followed [Sutton and Barto, 2018]. It is also referred to as off-policy TD control where the central idea is to find out the optimal behaviour for a given state. To encounter the tradeoff between exploration and exploitation, we utilize an ϵ-greedy policy. A formal description of the algorithm is given in the appendix (Algorithm 5).

2.3 Our general approach

There are 5 physical properties which may be altered for the environment

- Mass of cart
- Mass of pole
- Force applied
- Gravity
- Length of Pole

We currently alter the properties in a mutually exclusive manner and make observations for 5 distinct values for each property. The formal description is given here:

Algorithm 1 The general setup

```
Require: Discretized CartPole environment
  Approximate dynamics                          ▷ This is necessary for DP
  loop
      Alter environment configuration
      Train the model
      Test the model
      Restore configuration
  end loop
```

2.4 Discretization

An additional challenge for the formulation of both DP and Q-Learning algorithms was the state space of the CartPole environment. This is because both the algorithms work on discrete state spaces. Hence, we required a conversion to a grid world in order to facilitate the functioning of our algorithms. This was done using the technique of discretization.

The state space in the CartPole environment is represented by this 4-tuple

$$(position, velocity, angle, angular\_velocity)$$

Essentially, we try to partition the continuous state space by dividing them into intervals. This would result in a map from a 4-tuple in the continuous state space to another 4-tuple corresponding to the respective intervals. However this poses an additional challenge on the implementation for the DP approach. This arises because DP is a model based approach in which assumes the availability of model dynamics $P^a_{ss'}$ and reward function R^a_s which isn't available at our disposal.

To encounter this we currently utilize a simple approach based on Monte Carlo approximation. We simply run the model for a large number of episodes. After that, we approximate dynamics and rewards as follows:

$$P(S_{t+1} = s'|S_t = s, A_t = a) = \frac{N(S_{t+1} = s'|S_t = s, A_t = a)}{\sum_{s \in S} N(S_{t+1} = s'|S_t = s, A_t = a)}$$

$$R^a_s = \frac{\sum_{s' \in S} R(S_{t+1} = s'|S_t = s, A_t = a)}{\sum_{s \in S} N(S_{t+1} = s'|S_t = s, A_t = a)}$$

Here P refers to the transition probability model, R is the reward function and N denotes the frequency function. As an extension to this, we are currently exploring techniques of value function approximation which is better suited for continuous/large state spaces.

2.5 Observations

2.5.1 Test Results for DP

The time complexity for running each iteration of policy evaluation is $\mathcal{O}(n^2)$ where n is the size of state space($|S|$). Each run of policy improvement is also $\mathcal{O}(n)$. Policy iteration in a general case would depend on how fast the algorithm converges. As the run of policy iteration and value iteration, both can be limited to a fixed number of iterations for a good approximation of the policy and value functions, we have limited the run to a maximum of 10 iterations. As one may observe from the graph (Figure 2), the reward exceeds 100 for most cases which supports the fact that we converged to a valid policy.

Table 1: Table showing the rewards for different scenarios for DP with Policy Iteration

Length	Reward	Mass of Cart	Reward	Mass of Pole	Reward	Gravity	Reward	Force Magnitude	Reward
0	113.9	0	105.8055	0	114.6723	0	113.3926	0	123.9067
0.25	113.4907	0.5	114.5764	0.05	114.4168	4.9	79.5733	5	114.9168
0.5	124.4041	1	114.785	0.1	114.7848	9.8	105.3647	10	114.7911
0.75	114.9563	1.5	124.3584	0.15	114.4597	14.7	113.7681	15	112.1435
1	114.9463	2	104.1571	0.2	113.6004	19.6	113.6683	20	114.9378

Table 2: Table showing the rewards for different scenarios for DP with Value Iteration

Length	Reward	Mass of Cart	Reward	Mass of Pole	Reward	Gravity	Reward	Force Magnitude	Reward
0	163.428	0	133.6556	0	158.1512	0	159.4197	0	160.9021
0.25	112.8146	0.5	159.8159	0.05	166.9628	4.9	164.8667	5	161.5242
0.5	166.8262	1	145.1423	0.1	159.7848	9.8	163.9536	10	125.0289
0.75	161.605	1.5	159.5127	0.15	161.0587	14.7	153.3153	15	164.1749
1	159.4285	2	147.3229	0.2	147.182	19.6	152.4109	20	155.1366

2.5.2 Training for Q-Learning

It should be noted that the complexity of running each episode is $\mathcal{O}(cL)$ where L is the length of the episode and c is the constant factor accounting for taking a step in the environment and updating values. So, if we run a total of N episodes, the runtime comes to $\mathcal{O}(NcL)$. Taking this into account, we chose N to be 50000. We now try and observe the convergence of our algorithm by constructing a plot of rewards against the episode index. In order to reduce the number of points in the graph, we report the mean reward for continuous non-overlapping windows of size 2000, somewhat akin to a convolution. One might observe that

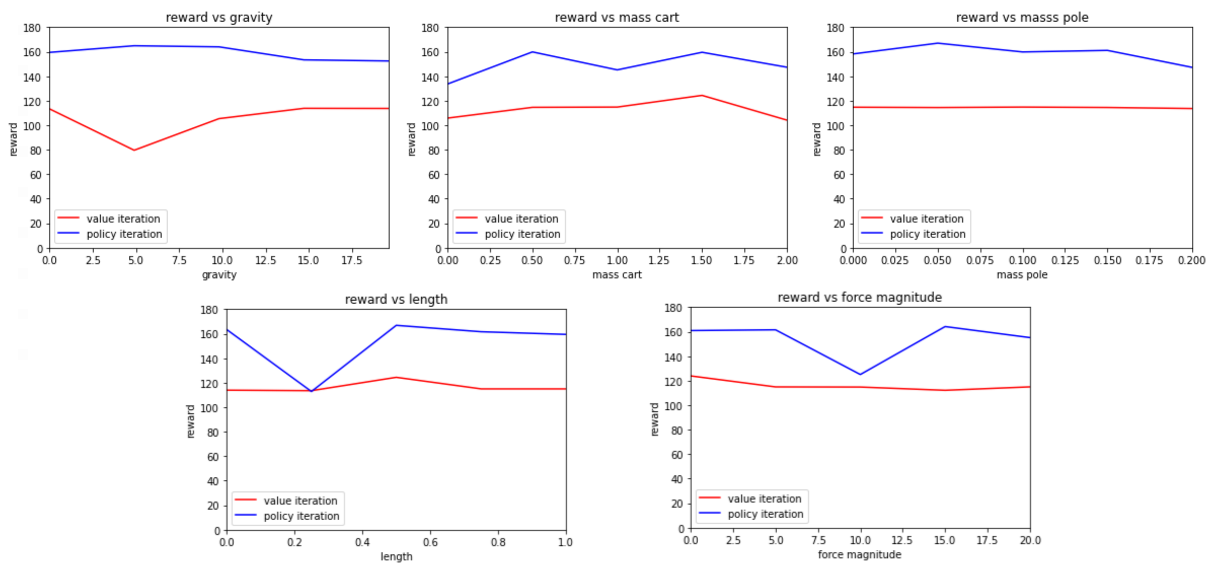

Figure 2: Average rewards per 1000 episodes for testing under the 25 different scenarios

some configurations converge while some don't (Figure 3). This may be attributed to the aspect of randomization that is introduced via Q-Learning.

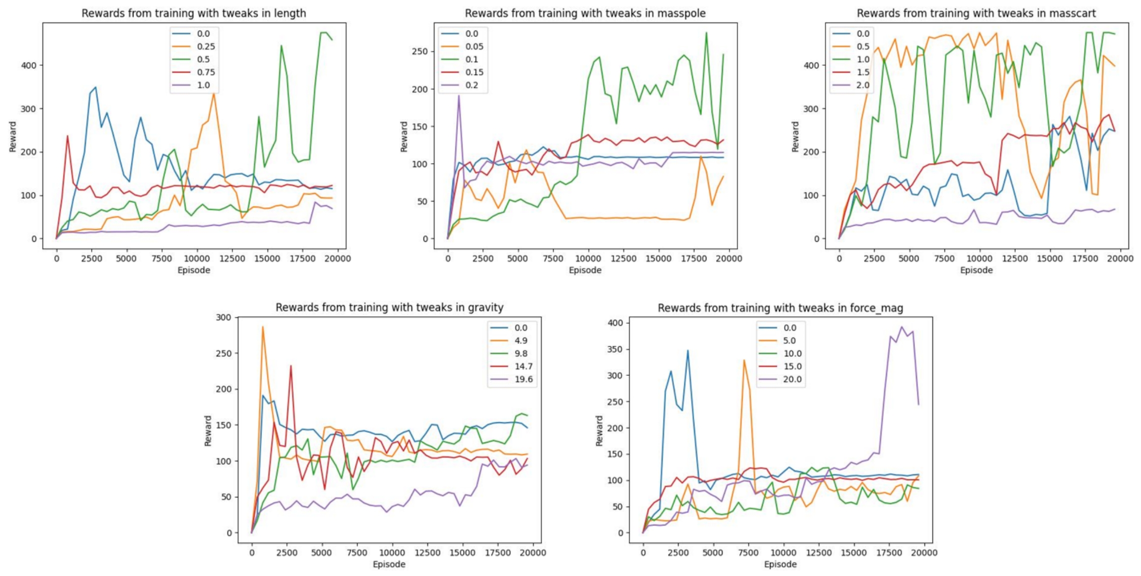

Figure 3: Window-wise average convoluted rewards per 2000 episodes for training under the 25 different scenarios

2.5.3 Test rewards for Q-Learning

A look at the average test rewards for different cases reveals some interesting information. A look at the data for length shows the rewards to range from approximately 106 to 475. One might argue that the randomization bit in the ϵ greedy policy might have contributed to this anomaly. This might be true to some extent, but it certainly isn't the sole reason for such a spread. It is the change in environment which has caused the same algorithm to perform differently under the different situations. Also, another anomaly might arise due to the discretization strategy that was used which might be responsible for the highly variable rewards. An analysis at a finer grain would probably improve the performance.

Table 3: Table showing the rewards for different scenarios

Length	Reward	Mass of cart	Reward	Mass of pole	Reward	Force	Reward	Gravity	Reward
0	179.599	0	213.219	0	119.656	0	107.165	0	130.017
0.25	129.013	0.5	120.823	0.05	100.287	5	109.868	4.9	107.619
0.5	475	1	212.643	0.1	434.675	10	113.528	9.8	147.795
0.75	105.714	1.5	437.89	0.15	142.363	15	109.549	14.7	105.954
1	124.059	2	67.121	0.2	114.892	20	179.568	19.6	122.432

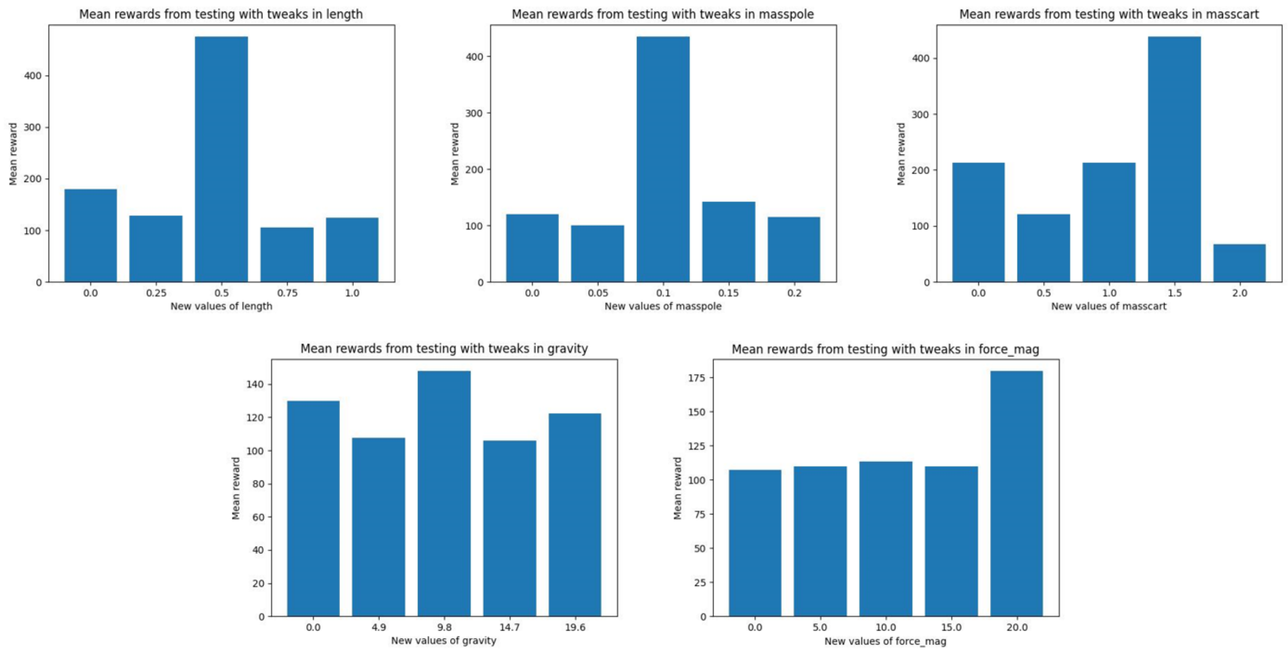

Figure 4: Average test rewards for different scenarios

2.5.4 Comparing DP with Q-Learning

There exists a classical bias-variance tradeoff between dynamic programming and Q-Learning algorithms. The DP algorithm introduces bias owing to the bootstrapping while Q-Learning leads to a lot of variance as we learn from individual episodes which might be extremely specific. This is clearly visible from our observations(Figure 5). The values of the different configurations were chosen in the form of an arithmetic progression with the first term being 0 and the third term being the original value. So, the third run for each attribute would correspond to the original configuration. One might argue that running the model with the same configuration makes little sense, but it revealed the high variance of the Q-Learning algorithm. The following table compares the data of Q-Learning with DP. Also, since the

	Mean	Standard deviation
Policy Iteration	114.826	6.021
Value Iteration	152.147	15.477
Q-Learning	276.728	149.415

Figure 5: Table for comparing mean and variance for the unaltered environment case

runtime of DP is roughly cubic w.r.t. the number of states, we had an additional restriction for choosing our bins during discretization. Although one might have easily chosen more number of bins for a more fine grained analysis in the case of Q-Learning, we chose to keep the discretization same for better comparison. This is one of the advantages that an off-policy method like Q-Learning provides. We have also compared the difference between rewards for DP and Q-Learning. Q-Learning outperforms policy iteration in 17 out of 25 cases and Value Iteration outperforms Q-Learning in 17 out of 25 cases. However, there is another interesting observation is the magnitude of difference between the rewards. The maximum deviation where DP does better is around 60 while for Q-Learning it is more than 300. This exhibits the high variance problem of Q-Learning.

2.5.5 Future Work

- Using value function approximation for learning the value functions and policy. We plan on using tile coding to formulate these functions and hope for better results in the general case.
- Transfer from one configuration to another: This actually is the central goal of our project where we will try and explore methods like transfer learning and find out if our model generalizes well to a different setting.

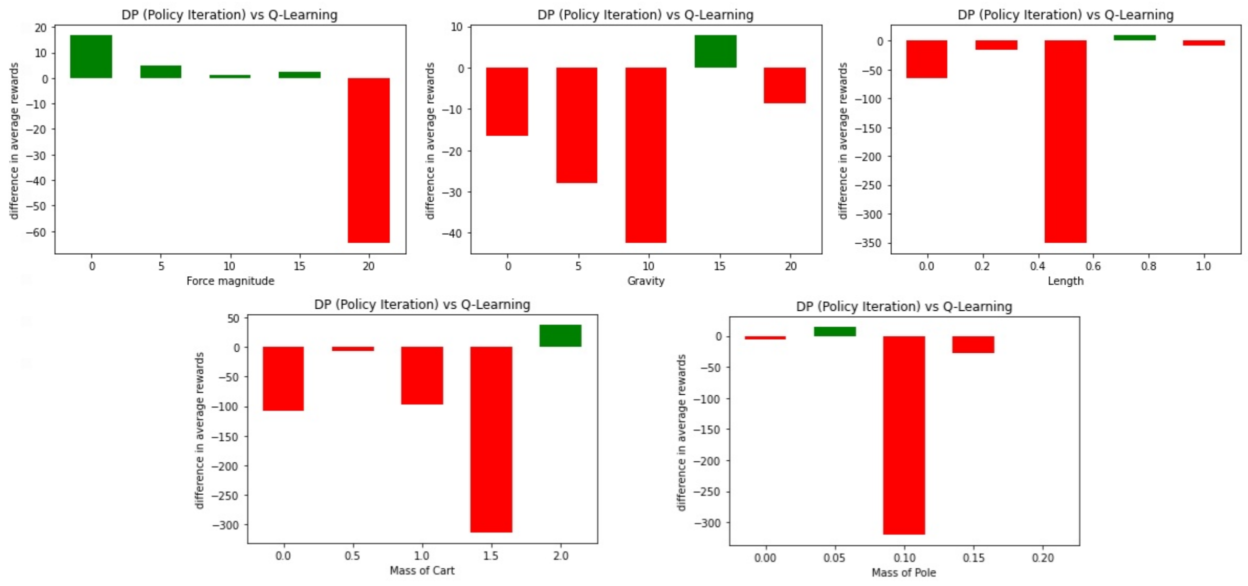

Figure 6: Policy Iterations vs Q-Learning for average rewards. A red bar indicates that Q-Learning performs better than Policy Iteration while a green bar indicates otherwise.

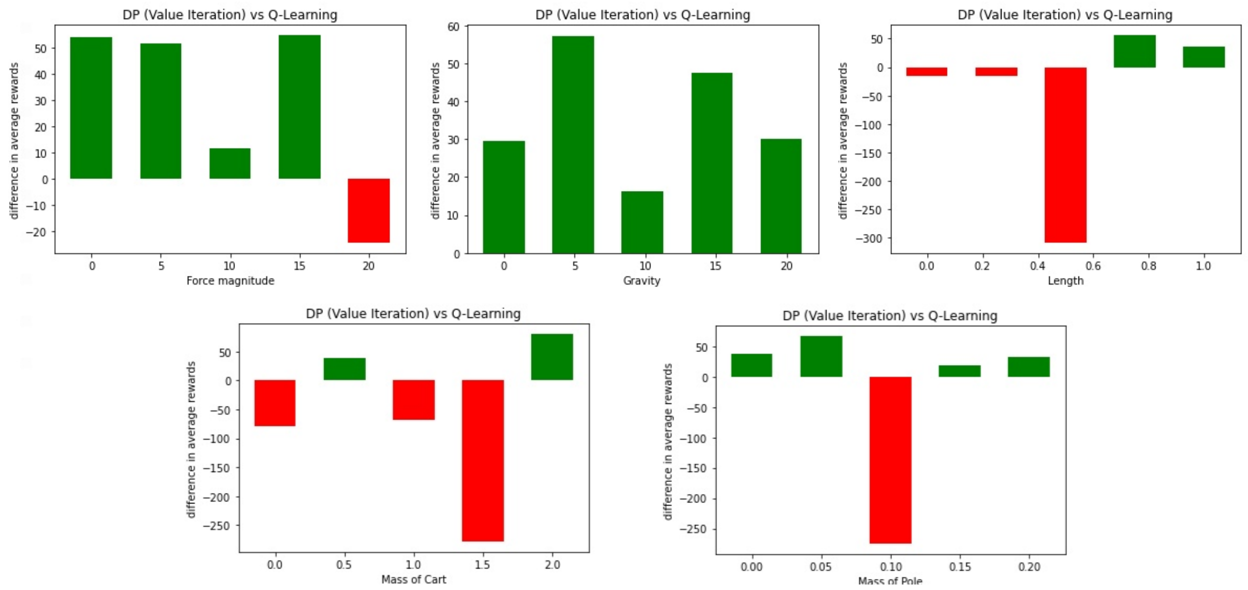

Figure 7: Value Iterations vs Q-Learning for average rewards. A red bar indicates that Q-Learning performs better than Policy Iteration while a green bar indicates otherwise.

3 Appendix

Algorithm 2 Iterative **Policy Evaluation**, for estimiating V $\approx v_\pi$

```
Require: Input $\pi$, the policy to be evaluated
 Algorithm parameter: a small threshold $\theta$ ¿ 0 determining accuracy of estimation
 Initialize V(s) arbitrarily, for s $\in$ S, and V(terminal) to 0
loop
    $\Delta \leftarrow 0$
    loop For each s $\in$ S
        v $\leftarrow$ V(s)
        V(s) $\leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end loop
end loop
until $\Delta < \theta$
```

Algorithm 3 Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

```
 Initialization
 V(s) $\in \mathbb{R}$ and $\pi$(s) $\in \mathbb{A}$(s) arbitrarily for all s $\in$ S; V(terminal) = 0
 Policy Evaluation
loop
    $\Delta \leftarrow 0$
    loop for each s $\in$ S;
        v $\leftarrow$ V(s)
        V(s) $\leftarrow \sum_{s',r} p(s',r|s,\pi(s))[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end loop
end loop
 until $\Delta < \theta$ (a small positive number determining the accuracy of estimation
 Policy Improvement
policy-stable $\leftarrow$ true
For each s $\in$ S;
    old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \mathrm{argmax_a} \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
   If old-action $\neq \pi(s)$, then policy-stable $\leftarrow$ false
If policy-stable, then stop and return V $\approx v_*$ and $\pi \approx \pi_*$; else go to Policy Evaluation
```

Algorithm 4 Value Iteration, for estimating $\pi \approx \pi_*$

```
Require: Algorithm parameter: a small threshold $\theta$ >0 determining accuracy of estimation
  Initialize V(s), for all s $\in S^+$, arbitrarily except that V(terminal) = 0
  loop
    $\Delta \leftarrow 0$
    loop For each s $\in$ S
      V(s) $\leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end loop
  end loop
  until $\Delta < \theta$
  Output a deterministic policy, $\pi \approx \pi_*$, such that
   $\pi(s) = \mathrm{argmax_a} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
```

Algorithm 5 Q-Learning algorithm

```
Require: Step size $\alpha \in [0, 1)$ and small $\epsilon \geq 0$
  Initialize $Q(s, a) \forall s \in S, a \in A$
  loop For each episode
    Initialize $S$
    loop For each step of episode
      Choose $A$ from $S$ using policy derived from $Q$ ($\epsilon$-greedy)
      Take action $A$, observe $R$, $S'$
      $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(s, a)]$
      $S' \leftarrow S$
    end loop
  end loop
```

References

Brockman G., Cheung V., Pettersson L., Schneider J., Schulman J., Tang J., and Zaremba W. (2016). *OpenAI Gym.*

Sutton R. S. and Barto A. G. (2018). *Reinforcement Learning: An Introduction.* Second. The MIT Press. URL: http://incompleteideas.net/book/the-book-2nd.html.