ESE 6510 Notes: Q-Learning, Value Learning, Fitted Value Iteration & Fitted Q-Iteration

Vaibhav Thakkar

Contents

Background & Setup

The Reinforcement Learning Problem

We consider a Markov Decision Process (MDP) defined by:

- State space $\mathcal{S}$, action space $\mathcal{A}$
- Transition dynamics $p(s' \mid s, a)$
- Reward function $r(s, a)$
- Discount factor $\gamma \in [0, 1)$

The goal is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes expected discounted return:

$$\mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right]$$

Value Functions

Key Definitions

State-Value function:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s \right]$$

Action-Value (Q) function:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s, a_0 = a \right]$$

Advantage function:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

Bellman equation:

$$Q^\pi(s, a) = r(s, a) + \gamma \, \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[V^\pi(s') \right]$$

Policy Iteration (General Idea)

The general template for policy-based RL:

1. Estimate A, Q, or V from state trajectories of π
2. Set $\pi' \leftarrow \pi$ (policy update)

Two main cases:

1. Dynamics known, discrete actions ⇒ Dynamic Programming
2. Dynamics unknown, discrete or continuous actions ⇒ Q or Value Learning

Dynamic Programming (Simplified Review)

When dynamics $p(s'|s, a)$ are *known* and state/action spaces are small, we can use exact dynamic programming by working *backwards* from the final state.

Algorithm 1 Dynamic Programming (Tabular, Finite Horizon)

Start from final state s_T: $V(s_T) = r(s_T)$ (final rewards only)
for $t = T-1$ down to 0 **do**
  $Q(s_t, a_t) = r(s_t, a_t) + \gamma\, V(s_{t+1})$
  $a_t^* = \arg\max_{a_t} Q(s_t, a_t)$
  $V(s_t) = \max_{a_t} Q(s_t, a_t)$
end for
Return optimal actions $a^*_{0:T-1}$

Limitations of DP:
1. Requires known transition dynamics to compute Q
2. Not feasible when action/state space is large

Value Learning (Fitted Value Iteration)

Motivation: Moving to Continuous / Large State Spaces

When the state space is continuous or large, we cannot maintain a table for $V(s)$. Instead, we *fit a function* (e.g., a neural network) parameterized by ϕ:

$$V_\phi(s) \approx V^\pi(s)$$

Core Identity

The key Bellman-based identity for the optimal value function is:

$$V_\phi(s) = \max_a \hat{Q}(s, a)$$

Define a loss over collected data:

$$\mathcal{L}(\phi) = \frac{1}{2} \left\| V_\phi(s) - \max_a \hat{Q}(s, a) \right\|^2$$

Fitted Value Iteration Algorithm

Algorithm 2 Fitted Value Iteration (FVI)

Collect states $\{s_i\}$ using some policy
repeat
  Set targets: for each s_i, for each action a, compute $s_i' = f(s_i, a)$ using known dynamics:

$$y_i = \max_a \left(r(s_i, a) + \gamma\, V_\phi(s_i') \right)$$

(true if π is optimal)

  Compute gradient:

$$\hat{g} = \frac{\partial \mathcal{L}}{\partial \phi} = \frac{\partial}{\partial \phi} \sum_i \| V_\phi(s_i) - y_i \|^2$$

  Update parameters:

$$\phi^{\text{new}} = \phi^{\text{old}} - \alpha \cdot \hat{g}$$

until converged

Runtime Policy Extraction: After learning V_ϕ, the greedy policy at runtime requires a one-step lookahead using the known dynamics $f(s, a)$:

$$\pi(s) = \arg\max_a \left[r(s, a) + \gamma \, V_\phi(f(s, a)) \right]$$

Key Advantage

You can use **any policy** to collect data for FVI – it is off-policy and does not depend on transition dynamics.

Caveats / Limitations of FVI:

- Only one network trained on a regression loss
- No convergence guarantees
- You are not directly optimizing the RL loss $\mathbb{E}_\tau[R(\tau)]$

Q-Learning

Motivation: Unknown Dynamics

What if we don't know the transition dynamics $p(s'|s, a)$? We fit $Q(s, a)$ directly!

$$S \longrightarrow Q(s, a) \quad \text{given } s_i, a_i, s'_i \quad \text{(does not depend on dynamics)}$$

Bellman Residual Objective

The Q-learning loss (Bellman residual) is:

$$\mathcal{L}(\phi) = \sum_i \left(Q_\phi(s_i, a_i) - y_i \right)^2$$

where the *target* y_i is:

$$y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a')$$

Tabular Q-Learning (Classic)

For small, discrete state/action spaces the Q-table is updated as:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Deep Q-Learning / Fitted Q-Iteration

For large or continuous state spaces, parameterize $Q_\phi(s, a)$ with a neural network.

Algorithm 3 Fitted Q-Iteration (FQI)

1: Initialize $Q_\phi(s, a)$ with random parameters ϕ
2: **Collect data** $\{(s_i, a_i, r_i, s'_i)\}$ using some policy
3: **repeat**
4: Compute targets: $y_i = r_i + \gamma \max_{a'} Q_\phi(s'_i, a')$
5: Compute gradient: $\hat{g} = \frac{\partial}{\partial \phi} \sum_i \| Q_\phi(s_i, a_i) - y_i \|^2$
6: Update: $\phi \leftarrow \phi - \alpha \, \hat{g}$
7: **until** converged
8: **Return** greedy policy $\pi(s) = \arg\max_a Q_\phi(s, a)$

Exploration Policies

When collecting data for Q-learning, we need a policy. Key options:

1. ε**-greedy:**

$$\pi(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \arg\max_a Q_\phi(s_t, a) \\ 0 & \text{otherwise} \end{cases} \quad \text{with prob. } 1-\varepsilon$$

   With probability ε, take a random action (explore).

2. **Boltzmann (Softmax) Exploration:**

$$\pi(a_t|s_t) \propto \exp\left(\frac{Q(s_t, a_t)}{\tau}\right), \quad \tau \in (0, 1]$$

   Temperature τ controls exploration/exploitation trade-off.

Advantages of Q-Learning over Policy Gradient

- Off-policy: any data can be reused
- No need to explicitly represent the policy
- More sample efficient in many settings