Policy Improvement

While we saw in the previous notes that value iteration can be used to find an optimal policy, it can be quite slow to converge. Interestingly, in later rounds of value iteration, the best action at each state rarely changes. In other words, the policy implicitly defined by the value function appears to converge more rapidly than the value function itself. This insight leads to a new approach that attempts to update the policy rather than the value function, until the policy converges.

Policy Evaluation

In order to update the policy, we need some way to measure its performance. Fortunately, we have a way to do this: we can simply compute the value function for a fixed policy. we will use $V^\pi(x,t)$ to denote the cost-to-go of a policy π in state x at time t. We can use the policy evaluation algorithm to tell us how good one policy is compared to another.

Algorithm `Value(`x,π,t,T`)`
  if $t = T-1$ **then**
    return $c(x,\pi(x,t))$
  end
  else
    return $c(x,\pi(x,t)) +$
    $\gamma \sum_{x' \in \mathbb{X}} p(x'|x,\pi(x,t))$`Value(`$x'$,$t+1$,$T$`)`
  end

Algorithm 7: Policy evaluation: a recursive algorithm that computes the value function for a given policy (same as in Chapter 1)

Recall from Chapter 1 that at $t \to \infty$, the value function reaches a fixed point defined by the Bellman equation

$$V(x,t) \xrightarrow{t\to\infty} V(x) = \min_a \left[c(x,a) + \gamma \sum_{x'} p(x'|x,a) V(x') \right]$$

If π is stationary (not a function of time), and if, with probability 1,

π enters a terminal state having zero cost, then as $t \to \infty$ the value function converges to the following fixed point

$$V^\pi(x,t) \xrightarrow{t\to\infty} V^\pi(x) = c(x,\pi(x)) + \gamma \sum_{x'} p(x'|x,\pi(x)) V^\pi(x')$$

Note that this equation is *linear* in $V^\pi(x)$. While this can be solved via policy iteration, an *alternate* way to solve this is to write a system of linear equations.

Let $\overrightarrow{c^\pi}$ and $\overrightarrow{V^\pi}$ be vectors of length $|\mathbb{X}|$ listing the cost and cost-to-go, respectively for $\forall x \in \mathbb{X}$.

$$\overrightarrow{V^\pi} = \overrightarrow{c} + \gamma P^\pi \overrightarrow{V^\pi} \tag{35}$$

$$\implies \overrightarrow{V^\pi} - \gamma P^\pi \overrightarrow{V^\pi} = \overrightarrow{c} \tag{36}$$

where P^π is the row stochastic transition matrix (its rows sum to 1) given the the fixed policy π

$$P^\pi = \begin{pmatrix} p(x_0|x_0,\pi(x_0)) & p(x_1|x_0,\pi(x_0)) & \dots \\ \vdots & \vdots & \vdots \\ p(x_0|x_n,\pi(x_n)) & p(x_1|x_n,\pi(x_n)) & \dots \end{pmatrix} \tag{37}$$

The operation of multiplying by P^π is the equivalent of calculating expectation.

This is a linear equation in $\overrightarrow{V^\pi}$ and its solution is

$$\overrightarrow{V^\pi} = (I - \gamma P^\pi)^{-1} \overrightarrow{c} \tag{38}$$

For $\gamma < 1$ this equation always has a solution (the Eigenvalues of P^π are always less than one, so $I - \gamma P^\pi$ is always invertible). The process of finding V^π is called policy evaluation.

Policy Improvement

If someone hands you a policy, and you evaluate that policy and discover that it is not optimal, then it is natural to want to improve the policy (see Figure 12).

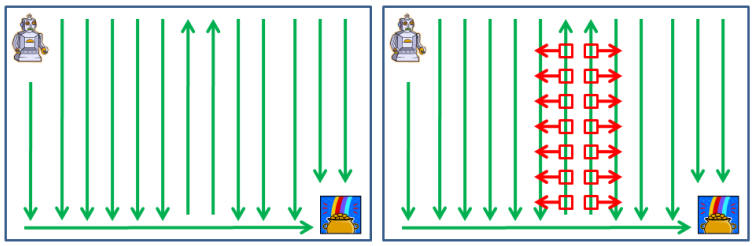


Figure 12: The left image shows a non optimal policy (green arrows). The right image shows how the policy could be improved by changing the action taken on a state-by-state basis. The new policy is still not optimal and could be improved by another round of policy iteration.

The policy can be improved by solving the following equation $\forall x \in \mathbb{X}$:

$$\pi'(x) = \underset{a}{\operatorname{argmin}}\, \mathrm{c}(x,a) + \gamma \mathbb{E}_{p(x'|x,a)}[V^{\pi}(x')] \tag{39}$$

Policy improvement can also be expressed in terms of $Q^{\pi}(x,a)$, the *quality function*, sometimes called the *Q-function* or *action value function*. The Q-function $Q^{\pi}(x,a)$ is the sum of the cost of performing an action a at state x and the cost to go from the resulting state under policy π.

$$Q^{\pi}(x,a) = \mathrm{cost}(x,a) + \gamma \mathbb{E}_{p(x'|x,a)}[V^{\pi}(x')] \tag{40}$$

A new policy π' can, therefore, be formed from an existing policy π by tweaking the action selected at a state. If π' is selected such that

$$\pi'(x) = \underset{a}{\operatorname{argmin}} Q^{\pi}(x,a) \tag{41}$$

then the new policy π' is guaranteed to be at least as good as π.

Policy Iteration Algorithm

Combining policy evaluation and policy improvement, we can get an algorithm for finding a good policy from an initial estimate of $\overrightarrow{V^{\pi}}$.

Start with arbitrary π_0
while *not converged* **do**
  Policy Evaluation: compute V^{π_k}
  for $\forall x \in \mathbb{X}$ **do**
    $\pi_{k+1}(x) = \underset{a}{\operatorname{argmin}}\, \mathrm{c}(x,a) + \gamma \sum_{x' \in \mathbb{X}} p(x'|x,a) V^{\pi_k}(x')$
  end
  $k++$
end
return $\pi_k(x), \forall x$

Algorithm 8: Policy Iteration. Here the policy evaluation step can be computed by, e.g. value iteration or solving a linear system.

Almost all dynamic programming algorithms use Value Iteration or Policy Iteration, and both form a nice basis for approximation algorithms (as we will see next Chapter).

Policy Iteration Optimality

During the policy iteration, the difference in value of the current policy π and the optimal value function $|V^{\pi}(x) - V^{*}(x)|$, decreases *exponentially*. As a result, this algorithm generally requires fewer iterations than Value Iteration, but it does require more work on each

iteration. Understanding whether Policy Iteration will converge to the best policy is not trivial - it is not at all obvious that it will converge. The standard argument, outlined below, uses contradiction to show that there are no local optima, so, since each step is an improvement, the algorithm will converge to the optimum. To see this we need to show that:

- The algorithm monotonically improves
- The algorithm reaches a global optimum

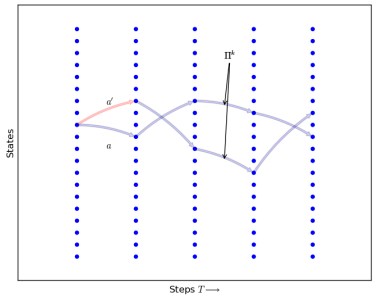
Monotonic Improvement

To show that the algorithm montonically improves, we look at the improvement in the value function between policies. We switch actions *only if* (see Figure 14) the policy from that point onwards is an improvement (if $V^{\pi_k}(x_{a'}) \geq V^{\pi_k}(x_a)$). Value improvement for Step 1 is:

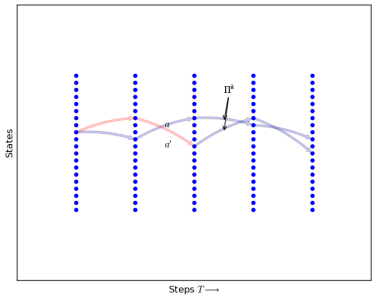
$$\gamma E_{p(x_o)}\left[\mathbb{E}_{p(x'|x_0,\pi_{k+1}(x_0))}[V^{\pi_k}(x')] - \mathbb{E}_{p(x'|x_0,\pi_k(x_0))}[V^{\pi_k}(x')]\right]$$

Value improvement for Step 2 is:

$$\gamma \mathbb{E}_{p^{\pi_{k+1}}(x_1)}\left[\mathbb{E}_{p(x'|x_1,\pi_{k+1}(x_1))}[V^{\pi_k}(x')] - \mathbb{E}_{p(x'|x_1,\pi_k(x_1))}[V^{\pi_k}(x')]\right]$$



: Choosing action a' in state x_0 which maximises $V(x_0)$, then following π^k



: Choosing action a' in state x_1 which maximises $V(x_1)$, then following π_k

Figure 14: Improvement in the value function: Blue dots denote states, red arrows denote actions that maximise the value for a state, blue arrows denote actions in π_k.

Optimality

The difference between value functions can be calculated using the following lemma.

Lemma 1. *Performance Difference Lemma:*

$$V^{\pi'}(x_0) - V^{\pi}(x_0) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{p_t} \left[V_{\pi', \pi_{t+1}, \pi_{t+2}, \ldots}(x) - V_{\pi, \ldots}(x) \right]$$

$$p_t = Pr[x_t = x \; x_o, \pi']$$

This lemma implies $V^{\pi'}(s_0) - V^{\pi}(s_0) \geq 0$ if the value function, $V(s)$, is such that:

$$\forall \; \pi, V^{\pi}(s,t) \rightarrow V^{\pi}(s, \infty)$$

The above equation means that the value function must converge. Provided that the value function converges, we see that the policy iteration algorithm climbs uphill.

When policy iteration has stopped making improvements, a local optimum is reached, i.e.

$$(\pi_k, V^{\pi_k}) = (\pi_{k+1}, V^{\pi_{k+1}})$$

If this is not a global optimum then:

$$(\pi^*, V^{\pi^*}) \neq (\pi_{k+1}, V^{\pi_{k+1}})$$

The value iteration step in policy iteration is

$$V^{\pi'}(x) = \min_a c(x,a) + \gamma \mathbb{E}_{p(x'|x,a)} \left[V^{\pi'}(x') \right]$$

which satisfies the Bellman Equation. Therefore $\pi' = \pi^*$ since there is no non-optimal solution to the Bellman Equation.

Access Models

For different reinforcement learning problems, there may be different levels of system access. For the Tetris problem assigned as homework for this class, we can create the exact same state over and over again while learning (or testing our algorithms). For robotic systems, we have much less access - we can never create *exactly* the same state again. Some common access models include:

1. Full Probabilistic Description

   In this model, we have access to the cost function and the transition function for every action P^a. A downside of having this kind of model is that it can become so large as to be computationally intractable for any non-trivial problem.

2. Deterministic Generative Model

This is the model that we have for the upcoming Tetris assignment. In this case, we have a function that maps $f(x, a) \to x'$, deterministically. Deterministic can mean that we have access to the random seed in a computer program, so we can recreate the same system including the randomness.

3. Generative Model

   In this model, we have programmatic access. We can put the system into any state we want.

4. Reset Model

   In this model, we can execute a policy or roll out dynamics any time we want, and we can always reset to some known state or distribution over states. This is a good model for a robot in the lab that can be reset to stable configurations.

5. Trace Model

   This is the model that best describes the real world. One good quote words it as "life is like a solo violin performance where you're learning how to play the violin".

Implementation Notes

Most people use *Modified Policy Iteration* in practice. Modified Policy Iteration warm-starts the policy evaluation step with the value function from the previous step and then does a single iteration of policy evaluation. Since the expensive part of policy iteration is the policy evaluation step, this warm-start can greatly speed up the algorithm.

Value iteration is expensive if the state space is too large. It can be used in its closed form (solving a linear system) if the value function is sparse. Otherwise the value function can be approximated by using:

- Linear function approximator $V(x) = w^T \phi(x)$
- Nearest Neighbour - for any x find the closest x' (in the sampled space) and return that value.