ECE 1508: Reinforcement Learning

Chapter 2: Model-based RL

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

Last Piece: *Dynamic Programming*

Right now, we know what to do *when we know MDP of environment*

1. *We can find optimal values from Bellman optimality equations*
2. *We could then find the optimal action-values*
3. *We finally get the optimal policy from optimal action-values*

The only remaining challenge is to find

an algorithmic approach to solve Bellman optimality equations

We complete this last piece using

Dynamic Programming $\equiv$ *DP*

Dynamic Programming: *Basic Idea*

Assume, we want to solve the problem of

$$x = f(x)$$

for some function $f(x)$

We could solve it via direct approach:

1. *Rewrite is as* $f(x) - x = 0$
2. *Solve it via classic algorithms*
   - ↳ *Reduce it to a known form, e.g., a polynomial*
   - ↳ *Solve it via an iterative method, e.g., Newton-Raphson or method of intervals*

Dynamic Programming: *Basic Idea*

Assume, we want to solve the problem of

$$x = f(x)$$

for some function $f(x)$

We could also solve it by recursion:

1. *Start with an x^0 and set $x^1 = f(x^0)$*
2. *Until $x^{k+1} \approx x^k$, we do*
   - ↳ *Update recursively as $x^{k+1} = f(x^k)$*
   - ↳ *Set $k \leftarrow k + 1$*

Under some conditions on $f(\cdot)$, this approach can converge

Dynamic Programming: *Example*

We want to solve

$$x = \frac{-1}{2 + x}$$

1. *Start with an* $x^0 = 0$
2. *We now get into the recursion loop*
   - ↳ $x^1 = f\left(x^0\right) = -\frac{1}{2}$
   - ↳ $x^2 = f\left(x^1\right) = -\frac{2}{3}$
   - ↳ $x^3 = f\left(x^2\right) = -\frac{3}{4}$
   - ↳ $\cdots$
   - ↳ $x^k = f\left(x^{k-1}\right) = -\frac{k}{k+1}$

We asymptotically converge to $x^\infty = -1$ *which is the solution*

↳ *Note that we always converge no matter which point we start*

Dynamic Programming: *Example*

Now, let's write the *same equation* in a *different recursive form*

$$x = \frac{-1 - x^2}{2}$$

1. *Start with an* $x^0 = 0$
2. *We get into recursion loop*
   - ↳ $x^1 = f\left(x^0\right) = -0.5$
   - ↳ $x^2 = f\left(x^1\right) = -0.625$
   - ↳ $\cdots$
   - ↳ $x^\infty = -1$

1. *Start with an* $x^0 = 5$
2. *We get into recursion loop*
   - ↳ $x^1 = f\left(x^0\right) = -13$
   - ↳ $x^2 = f\left(x^1\right) = -85$
   - ↳ $\cdots$
   - ↳ $x^\infty = -\infty$

We can now diverge if we start with a wrong initial point!

Not all recursive forms are always converging!

Dynamic Programming: *Applications to Our Problem*

Our problem has a similar form: *we need to solve Bellman equations*

which are recursive equations

So, we could use DP to find the solution

There are two major DP approaches

- *Policy Iteration that uses recursion to iterate between*
  - ↳ *Policy Evaluation*
  - ↳ *Policy Improvement*
- *Value Iteration which applies recursion on optimal Bellman equation*

Let's look at these two approaches in detail

Policy Evaluation: *Step I*

The first step is *policy evaluation*: *we can formulate this problem as follows*

Ultimate Goal of Policy Evaluation

Given a policy π, we intend to evaluate values of all states by recursion

---

Before we start, let's recap a few definitions: recall expected policy reward

$$\bar{\mathcal{R}}_{\pi}\left(s\right) = \sum_{m=1}^{M} \bar{\mathcal{R}}\left(a^{m}, s\right) \pi\left(a^{m} | s\right)$$

For sake of compactness, we use the following notation

$$\bar{\mathcal{R}}_{\pi}\left(s\right) = \mathbb{E}_{\pi}\left\{\bar{\mathcal{R}}\left(A, s\right) | s\right\}$$

Policy Evaluation: *Step I*

Similarly, we define the notation

$$\mathbb{E}_\pi \left\{ v_\pi \left(\bar{S} \right) | s, a \right\} = \sum_{n=1}^{N} v_\pi \left(s^n \right) p \left(s^n | s, a \right)$$

and also denote its expected form over the action set by

$$\begin{aligned}\mathbb{E}_\pi \left\{ v_\pi \left(\bar{S} \right) | s \right\} &= \sum_{n=1}^{N} v_\pi \left(s^n \right) p_\pi \left(s^n | s \right) \\ &= \sum_{m=1}^{M} \underbrace{\sum_{n=1}^{N} v_\pi \left(s^n \right) p \left(s^n | s, a^m \right)} \pi \left(a^m | s \right) \\ &= \sum_{m=1}^{M} \mathbb{E}_\pi \left\{ v_\pi \left(\bar{S} \right) | s, a^m \right\} \pi \left(a^m | s \right)\end{aligned}$$

Policy Evaluation: *Step I*

We can then write the Bellman equations compactly as

$$v_\pi(s) = \bar{\mathcal{R}}_\pi(s) + \gamma \mathbb{E}_\pi \left\{ v_\pi\left(\bar{S}\right) | s \right\}$$

for value function and also as

$$q_\pi(s, a) = \bar{\mathcal{R}}(s, a) + \gamma \mathbb{E}_\pi \left\{ v_\pi\left(\bar{S}\right) | s, a \right\}$$

for action-value function

Now, we are ready to evaluate a policy by recursion

Policy Evaluation: *Value Computation via Recursion*

Recall our perspective on value computation:

values are N unknowns that we want to compute from Bellman equations

Now, if someone claims that *the values*

$$v_\pi \left(s^n\right) = v_n$$

for $n = 1 : N$ are values of policy π, can we confirm it?

+ *Shouldn't we simply use Bellman Equation?!*
– Exactly!

Policy Evaluation: *Value Computation via Recursion*

We could confirm

$$v_\pi\left(s^n\right) = v_n$$

by writing first finding for every state s

$$\begin{aligned}\mathbb{E}_\pi\left\{v_\pi\left(\bar{S}\right)|s\right\} &= \sum_{n=1}^{N} v_\pi\left(s^n\right) p_\pi\left(s^n|s\right) \\ &= \sum_{n=1}^{N}\sum_{m=1}^{M} v_\pi\left(s^n\right) p\left(s^n|s,a^m\right)\pi\left(a^m|s\right) \\ &= \sum_{n=1}^{N}\sum_{m=1}^{M} \underbrace{v_n}_{\text{claimed value}} \underbrace{p\left(s^n|s,a^m\right)}_{\text{transition model}} \underbrace{\pi\left(a^m|s\right)}_{\text{policy}}\end{aligned}$$

Policy Evaluation: *Value Computation via Recursion*

We could confirm

$$v_\pi\left(s^n\right) = v_n$$

by writing first finding for every state s

$$\mathbb{E}_\pi\left\{v_\pi\left(\bar{S}\right)|s\right\} = \text{computed from } v_n\text{'s} := F\left(\{v_1,\dots,v_N\},s\right)$$

and then checking if

$$\begin{aligned} v_\pi\left(s^n\right) = v_n &= \bar{\mathcal{R}}_\pi\left(s^n\right) + \gamma\mathbb{E}_\pi\left\{v_\pi\left(\bar{S}\right)|s^n\right\} \\ &= \bar{\mathcal{R}}_\pi\left(s^n\right) + \gamma F\left(\{v_1,\dots,v_N\},s\right) \end{aligned}$$

holds for all $n = 1:N$

Policy Evaluation: *Value Computation via Recursion*

If it happens that the claimed $v_\pi(\cdot)$ is not a valid claim; then, *we get out of Bellman equation*

$$\bar{v}_\pi(s^n) = \bar{v}_n = \bar{\mathcal{R}}_\pi(s^n) + \gamma \mathbb{E}_\pi\left\{v_\pi\left(\bar{S}\right)|s^n\right\}$$

which is different from the claimed $v_\pi(\cdot)$, *i.e.,* $v_n \neq \bar{v}_n$

Policy Evaluation

We iterate this procedure until we can confirm, i.e., we

1. *set* $v_\pi(\cdot) \leftarrow \bar{v}_\pi(\cdot)$
2. *repeat the same procedure and compute new* $\bar{v}_\pi(\cdot)$

We stop when $v_\pi(\cdot) = \bar{v}_\pi(\cdot)$, *or at least it happens approximately*

Policy Evaluation

`PolicyEval` $\left(\pi, v_\pi^0\right)$:

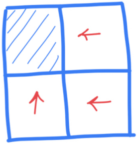
Initiate values with v_π^0 *and set* $k = 0$
Make sure that $v_\pi^0\left(s\right) = 0$ *for* ***terminal*** *states* s
Choose a small threshold ϵ *and initiate* $\Delta = +\infty$ `# stopping criteria`
for $n = 1 : N$ **do**
    Compute $\bar{\mathcal{R}}_\pi\left(s^n\right) = \mathbb{E}_\pi\left\{\bar{\mathcal{R}}\left(s^n, a\right)\right\}$ `# average rewards`
end for
while $\Delta > \epsilon$ **do**
    for $n = 1 : N$ **do**
        Update $v_\pi^{k+1}\left(s^n\right) = \bar{\mathcal{R}}_\pi\left(s^n\right) + \gamma \mathbb{E}_\pi\left\{v_\pi^k\left(\bar{S}\right) | s^n\right\}$ `# DP update`
    end for
    $\Delta = \max_n \left| v_\pi^{k+1}\left(s^n\right) - v_\pi^k\left(s^n\right)\right|$ `# check convergence`
    Update $k \leftarrow k + 1$
end while

Recursion Loop

Attention

We should make sure that terminal states are all initiated with zero value

Example: *Dummy Grid World*



Let's try with our dummy grid world: we saw that

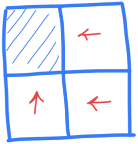
$$\bar{\mathcal{R}}_\pi(0) = 0 \qquad \bar{\mathcal{R}}_\pi(1) = -1 \qquad \bar{\mathcal{R}}_\pi(2) = -1 \qquad \bar{\mathcal{R}}_\pi(3) = -1$$

Now let's evaluate its values by recursion: we first note that, if we have

$$\mathbb{E}_\pi\left\{v_\pi^k(\bar{S})\,|0\right\} = v_\pi^k(0) \qquad \mathbb{E}_\pi\left\{v_\pi^k(\bar{S})\,|1\right\} = v_\pi^k(0)$$

$$\mathbb{E}_\pi\left\{v_\pi^k(\bar{S})\,|2\right\} = v_\pi^k(0) \qquad \mathbb{E}_\pi\left\{v_\pi^k(\bar{S})\,|3\right\} = v_\pi^k(2)$$

Example: *Dummy Grid World*



```
PolicyEval(π, v_π^0):
  Initiate values with v_π^0(1), v_π^0(2) and v_π^0(3) at random and set v_π^0(0) = 0
  Set ϵ = 0.001, and initiate Δ = 1000                         # stopping criteria
  while Δ > ϵ do
      Update v_π^{k+1}(1) = −1 + v_π^k(0)                               # DP update
      Update v_π^{k+1}(2) = −1 + v_π^k(0)                               # DP update
      Update v_π^{k+1}(3) = −1 + v_π^k(2)                               # DP update
      Δ = max_{s∈{1,2,3}} |v_π^{k+1}(s) − v_π^k(s)|               # check convergence
      Update k ← k + 1
  end while
```

It converges after only one recursion!

Policy Improvement

Let us know recall optimality constraint: *with optimal policy, we have*

$$v_\star(s) = \max_m q_\star(s, a^m)$$

which can be achieved by policy

$$\pi^\star(a^m|s) = \begin{cases} 1 & m = \operatorname*{argmax}_m q_\star(s, a^m) \\ 0 & m \neq \operatorname*{argmax}_m q_\star(s, a^m) \end{cases}$$

This means that if π is not optimal, we would have

$$\pi(a^m|s) \neq \begin{cases} 1 & m = \operatorname*{argmax}_m q_\pi(s, a^m) \\ 0 & m \neq \operatorname*{argmax}_m q_\pi(s, a^m) \end{cases}$$

Policy Improvement

In other words, if we change our policy to

$$\bar{\pi}\left(a^m|s\right) = \begin{cases} 1 & m = \operatorname*{argmax}_m q_\pi\left(s, a^m\right) \\ 0 & m \neq \operatorname*{argmax}_m q_\pi\left(s, a^m\right) \end{cases}$$

Then, it should give us better values, i.e., $\bar{\pi} \geqslant \pi$!

+ *Are you sure?! I don't see it immediately*
– We can actually show it!

This is what we call policy improvement theorem

Policy Improvement

Policy Improvement

Given (deterministic) policy π^k, we can always design a better policy π^{k+1} by setting it to

$$\pi^{k+1}\left(a^m|s\right) = \begin{cases} 1 & m = \operatorname*{argmax}_m q_{\pi^k}\left(s, a^m\right) \\ 0 & m \neq \operatorname*{argmax}_m q_{\pi^k}\left(s, a^m\right) \end{cases}$$

Policy Improvement

```
PolicyImprov(v_π):
1: for n = 1 : N do
2:     for m = 1 : M do
3:         Compute R̄(s^n, a^m)
4:         q_π(s^n, a^m) = R̄(s^n, a^m) + γE_π{v_π(S̄)|s^n, a^m}    # action-values
5:     end for
6:     Compute an improved policy as                            # policy improvement
```

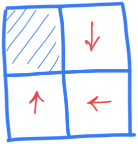
$$\bar{\pi}\left(a^m|s^n\right) = \begin{cases} 1 & m = \operatorname*{argmax}_m q_\pi\left(s^n, a^m\right) \\ 0 & m \neq \operatorname*{argmax}_m q_\pi\left(s^n, a^m\right) \end{cases}$$

```
7: end for
```

Attention

Here, we do no recursion

Example: *Dummy Grid World*



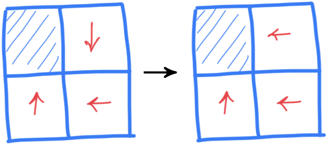
Let's try dummy grid world with above non-optimal policy: here, we have

$$v_\pi(0) = 0 \qquad v_\pi(1) = -3 \qquad v_\pi(2) = -1 \qquad v_\pi(3) = -2$$

We now look at acion-values at the problematic state $s = 1$

$$\begin{aligned} q_\pi(1,0) &= -1 \\ q_\pi(1,1) &= -3 \\ q_\pi(1,2) &= -3.5 \\ q_\pi(1,3) &= -3.5 \end{aligned} \rightsquigarrow -3 = v_\pi(1) \neq \max_a q_\pi(1,a) = -1$$

Example: *Dummy Grid World*



Now if we improve the policy, we get

$$\bar{\pi}(a|1) = \begin{cases} 1 & m = \operatorname*{argmax}_m q_\pi(1, a) \\ 0 & m \neq \operatorname*{argmax}_m q_\pi(1, a) \end{cases} = \begin{cases} 1 & a = 0 \\ 0 & a \neq 0 \end{cases}$$

which is actually optimal

Policy Iteration: *Improving Policy by Recursion*

Looking at the policy improvement theorem, we see

> *If we plug in $\pi^k = \pi^\star$ into algorithm; then, after policy improvement*
> ↳ *we get $\pi^{k+1} = \pi^\star$*
> ↳ *say we evaluate values for $\pi^{k+1} = \pi^\star$ and plug back to algorithm*
>   ↳ *we get $\pi^{k+2} = \pi^\star$*
>   ↳ *say we evaluate values for $\pi^{k+2} = \pi^\star$ and plug back to algorithm*
>   ↳ $\ldots$

So, optimal policy is a fixed-point for this recursion

Policy Iteration

We can start with an arbitrary policy π^0 and keep doing the above recursion until we see that $\pi^{k+1} = \pi^k$ which indicates that we reached optimal policy

Policy Iteration

```
PolicyItr():
1: Initiate with random v_π(s) for all non-terminal states s
2: Set v_π(s) = 0 for terminal states s
3: Initiate two random policies π and π̄
4: while π ≠ π̄ do
5:     v_π = PolicyEval(π, v_π) and π ← π̄        Recursion
6:     π̄ = PolicyImprov(v_π)
7: end while                                      Recursion
```

Note that this is a *nested recursive computation*

- *There is a loop for recursion inside the algorithm in which*
  - ↳ *at each iteration we evaluate the policy recursively*
- *But, we initiate each policy evaluation loop with the values of last iteration*
  - ↳ *this can improve the convergence speed*

Back-Tracking by Recursion

+ *But wait a Moment! We already talked about back-tracking optimal policy from Bellman optimality equation! Don't we implement that?!*
– Sure! We can do the same thing by recursion

We follow the same idea but we use recursion

1. *We can find optimal values from Bellman optimality equations*
   ↳ *This is where we use recursion*
2. *We could then find the optimal action-values*
3. *We finally get the optimal policy from optimal action-values*

Recall: *Back-Tracking from Optimal Values*

`OptimBackTrack():`

1: *Solve Bellman equations* `# we use recursion`
2: **for** $n = 1 : N$ **do**
3: **for** $m = 1 : M$ **do**
4: Set $q_\star(s^n, a^m) = \bar{\mathcal{R}}(s^n, a^m) + \gamma \mathbb{E}\left\{v_\star\left(\bar{S}\right) | s^n, a^m\right\}$ `# action-values`
5: **end for**
6: Compute optimal policy via optimality constraint

$$\pi^\star(a^m|s) = \begin{cases} 1 & m = \operatorname*{argmax}_m q_\star(s, a^m) \\ 0 & m \neq \operatorname*{argmax}_m q_\star(s, a^m) \end{cases}$$

7: **end for**

Recursion with Bellman Optimality

Recall Bellman optimality equation

$$v_\star(s) = \max_m \left(\bar{\mathcal{R}}(s, a^m) + \gamma \mathbb{E}\left\{v_\star(\bar{S}) | s, a^m\right\}\right)$$

We can again solve it by recursion: *we start with some* $v_\star^0(\cdot)$ *and then for every state* s *and action* a^m*, we compute*

$$\mathbb{E}\left\{v_\star^k(\bar{S}) | s, a^m\right\} = \sum_{n=1}^{N} \underbrace{v_\star^k(s^n)}_{\text{last computed value}} \underbrace{p(s^n | s, a^m)}_{\text{transition model}}$$

We then update the optimal value function as

$$v_\star^{k+1}(s) = \max_m \left(\bar{\mathcal{R}}(s, a^m) + \gamma \mathbb{E}\left\{v_\star^k(\bar{S}) | s, a^m\right\}\right)$$

Value Iteration vs *Policy Iteration*

Before we complete the value iteration algorithm: *it is interesting to put its recursion next to the one used for policy evaluation*

With optimality equation, we iterate as

$$v_\star^{k+1}(s) = \max_m \left[\bar{\mathcal{R}}(s, a^m) + \gamma \mathbb{E}\left\{ v_\star^k(\bar{S}) | s, a^m \right\} \right]$$

With Bellman equation for a given policy π, we iterate as

$$v_\pi^{k+1}(s) = \bar{\mathcal{R}}_\pi(s) + \gamma \mathbb{E}_\pi \left\{ v_\pi^k(\bar{S}) | s \right\}$$

$$= \sum_{m=1}^{M} \left(\bar{\mathcal{R}}(s, a^m) + \gamma \mathbb{E}\left\{ v_\pi^k(\bar{S}) | s, a^m \right\} \right) \pi(a^m | s)$$

Value Iteration vs *Policy Iteration*

With optimality equation, we iterate as

$$v_\star^{k+1}(s) = \max_m \left[\bar{\mathcal{R}}(s, a^m) + \gamma \mathbb{E}\left\{ v_\star^k(\bar{S}) | s, a^m \right\} \right]$$

With Bellman equation for a given policy π, we iterate as

$$v_\pi^{k+1}(s) = \sum_{m=1}^{M} \left(\bar{\mathcal{R}}(s, a^m) + \gamma \mathbb{E}\left\{ v_\pi^k(\bar{S}) | s, a^m \right\} \right) \pi(a^m | s)$$

This indicates that for both recursive loops

- *we compute M values per iteration per state*
  - ↳ *in policy iteration, we compute the average of these M via π*
  - ↳ *in value iteration, we take the largest among these M values*

Value Iteration

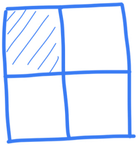
`ValueItr():`

Initiate with random $v_\star^0(s)$ *for all states, and set* $v_\star^0(s) = 0$ *for* ***terminal*** *states*
Choose a small threshold ϵ, *initiate* $\Delta = +\infty$ *and* $k = 0$
while $\Delta > \epsilon$ ***do***
    for $n = 1 : N$ ***do***
        for $m = 1 : M$ ***do***
            Compute $q_\star(s^n, a^m) = \bar{\mathcal{R}}(s^n, a^m) + \gamma \mathbb{E}\left\{ v_\star^k(\bar{S}) \,|s^n, a^m \right\}$
        end for
        Update $v_\pi^{k+1}(s^n) = \max_m q_\star(s^n, a^m)$ `# DP update`
    end for
    Set $\Delta = \max_n |v_\pi^{k+1}(s^n) - v_\pi^k(s^n)|$ *and* $k \leftarrow k+1$
end while
Compute an optimal policy as

$$\bar{\pi}(a^m|s) = \begin{cases} 1 & m = \underset{m}{\operatorname{argmax}}\, q_\star(s, a^m) \\ 0 & m \neq \underset{m}{\operatorname{argmax}}\, q_\star(s, a^m) \end{cases}$$

Recursion

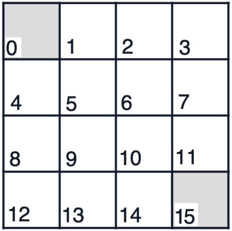
Example: *Dummy Grid World*



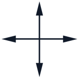
You may try policy and value iteration for this problem at home!

Easy as Pie ☺

Example: *A Bit Larger Grid World*[1]



Board ≡ states



moves ≡ actions

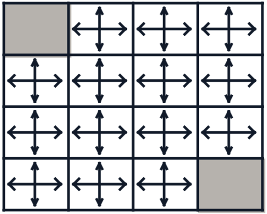
Let's do a bit of more serious example: we are now in a 4×4 *grid world*

- *We have two terminal states shown in gray*
- *Each move we do gets a -1 reward*
  - ↳ *We also get -1 reward if we hit a corner*
  - ↳ *We get zero reward at terminal state*

In simple words: we are looking for shortest path to the corners

[1] *This example is taken from Sutton and Barto's Book; Example 4.1 in Chapter 4*

Example: *A Bit Larger Grid World*



initial policy

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

initial values

Let's first try policy iteration: we start with

- *a uniform random policy* π^0
- *all values being zero, i.e.,* $v^0_{\pi^0}(s) = 0$ *for all* s

Example: *A Bit Larger Grid World*

Recall policy iteration:

```
PolicyItr():
1: Initiate with random v_π(s) for all non-terminal states s
2: Set v_π(s) = 0 for terminal states s
3: Initiate two random policies π and π̄
4: while π ≠ π̄ do
5:     v_π = PolicyEval(π, v_π) and π ← π̄        Recursion
6:     π̄ = PolicyImprov(v_π)
7: end while                                        Recursion
```

We should start with $v_{\pi^0}^0(\cdot)$ and do the red recusion first

- *at the end of this recursion we have evaluated the random policy*

Example: *A Bit Larger Grid World*

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$v_{\pi^0}^0$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$v_{\pi^0}^1$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$v_{\pi^0}^2$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$v_{\pi^0}^3$

$\cdots$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$v_{\pi^0}^\infty$

We now have evaluated the value of random policy $v_{\pi^0} = v_{\pi^0}^\infty$

Example: *A Bit Larger Grid World*

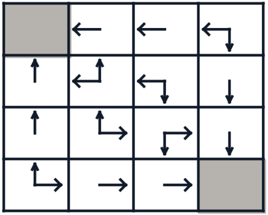
Recall policy iteration:

```
PolicyItr():
1: Initiate with random v_π(s) for all non-terminal states s
2: Set v_π(s) = 0 for terminal states s
3: Initiate two random policies π and π̄
4: while π ≠ π̄ do
5:     v_π = PolicyEval(π, v_π) and π ← π̄        Recursion
6:     π̄ = PolicyImprov(v_π)
7: end while                                      Recursion
```

Next, we do the outer recusion recursion, i.e.,

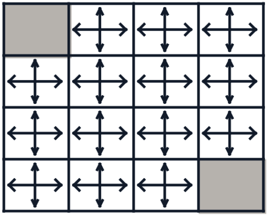
- *we improve the policy*

Example: *A Bit Larger Grid World*



π^1

$\longleftarrow$



π^0

We improve policy by taking actions with maximal action-values

- *if we have multiple maximal action-values we can behave randomly*

Example: *A Bit Larger Grid World*

Recall policy iteration:

PolicyItr():

1: *Initiate with random $v_\pi(s)$ for all **non-terminal** states s*
2: *Set $v_\pi(s) = 0$ for **terminal** states s*
3: *Initiate two random policies π and $\bar{\pi}$*
4: ***while** $\pi \neq \bar{\pi}$ **do***
5: $v_\pi =$ PolicyEval(π, v_π) *and* $\pi \leftarrow \bar{\pi}$ Recursion
6: $\bar{\pi} =$ PolicyImprov(v_π)
7: ***end while*** Recursion

We now *start with $v^0_{\pi^1} = v_{\pi^0} = v^\infty_{\pi^0}$* *and do the red recusion again*

- *at the end of this recursion we have evaluated the new policy π^1*

Example: *A Bit Larger Grid World*

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

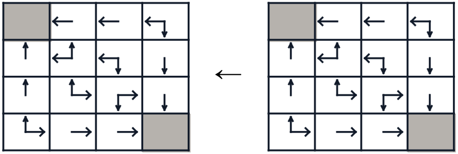
$v^0_{\pi^1}$

$\cdots$

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

$v^{+\infty}_{\pi^1}$

After evaluating policy π^1 *as* $v_{\pi^1} = v^{\infty}_{\pi^1}$, *we do the next improvement*



$\pi^2 = \pi^1$ $\qquad$ π^1

Well $\pi^2 = \pi^1$ *and we should stop!*

Example: *A Bit Larger Grid World*

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

initial values

Now we try value iteration: for start, we only need an initial value, so we set

- *all values being zero, i.e.,* $v_\star^0(s) = 0$ *for all* s

We keep recursion until we find the optimal values

Example: *A Bit Larger Grid World*

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$v_\star^0$

$\cdots$

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

$v_\star^{+\infty}$

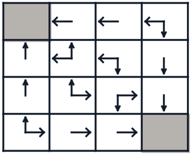
Now, we back-track the optimal policy $\pi^\star$

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

$v_\star$

$\longrightarrow$ action-values $\longrightarrow$

$q_\star$



$\pi^\star$

Complexity of Policy and Value Iteration

+ *It seems that value iteration has less complexity!*
- Well, it is not in order, but yes! It usually converge faster

---

In our example with policy iteration, we had to evaluate two policies

- *once for π^0 and once for π^1*
- *say the first recursion took K_1 iterations and the second took K_2*
  - ↳ *the total number of iterations is then $K_1 + K_2$*
  - ↳ *in practice, it often happens that $K_2 \ll K_1$*
    - ↳ *because we already start from good values with $v_{\pi^1}^0 = v_{\pi^0}^{+\infty}$*

With value iteration, we had to only evaluate optimal policy

- *say it takes $K_\star$ iterations: there is no reason that $K_\star$ be same as K_1 or K_2*
  - ↳ *each evaluation has a different initial and converging point*

↳ *in practice, it often happens that $K_\star > K_1$ and $K_\star \gg K_2$*

- ↳ *so it might be that $K_\star \approx K_1 + K_2$*
- ↳ *but usually withmultiple policy improvements, we see $K_\star < K_1 + K_2 + \dots$*

Complexity of Policy and Value Iteration

+ *If so, why should we use policy iteration?!*
– Well, not all problems are like a dummy grid world

In practice, it might be computationally hard to get very close to optimal values

- *in this case, we take non-converged values*
  ↳ *we consider them estimates of optimal values*
- *in value iteration we approximate optimal policy with on these estimates*
  ↳ *this might be a loose estimate*

If we do the same approximative computation with policy iteration

- *we often end up with a better policy*

Moral of Story

While value iteration typically show faster convergence, policy iteration can give better policies after convergence

Generalized Policy Iteration

In practice, we can terminate or change the order of computation in policy iteration to reduce its complexity: for instance, we could have

`GenPolicyItr():`

*Initiate with random $v_\pi(s)$ for all **non-terminal** states s*
*Set $v_\pi(s) = 0$ for **terminal** states s*
Initiate two random policies π and $\bar{\pi}$
while** $\pi \neq \bar{\pi}$ **do
  $v_\pi = \texttt{TerminPolicyEval}(\pi, v_\pi)$ *and* $\pi \leftarrow \bar{\pi}$ changed
  $\bar{\pi} = \texttt{PolicyImprov}(v_\pi)$
end while

where $\texttt{TerminPolicyEval}(\pi, v_\pi)$ *evaluates policy π from starting value function v_π with a terminating recursion loop*

Generalized Policy Iteration: *Terminating Evaluation*

TerminPolicyEval(π, v_π^0):

Initiate values with v_π^0 *and set* $k = 0$
Make sure that $v_\pi^0(s) = 0$ *for* ***terminal*** *states* s
Choose a small threshold ϵ *and initiate* $\Delta = +\infty$ # stopping criteria
for $n = 1 : N$ **do**
  Compute $\bar{\mathcal{R}}_\pi(s^n) = \mathbb{E}_\pi\{\bar{\mathcal{R}}(s^n, a)\}$ # average response
end for
while $\Delta > \epsilon$ *and* $k < K$ **do** changed
  for $n = 1 : N$ **do**
    Update $v_\pi^{k+1}(s^n) = \bar{\mathcal{R}}_\pi(s^n) + \gamma \mathbb{E}_\pi\{v_\pi^k(\bar{S}) | s^n\}$ # DP update
  end for
  $\Delta = \max_n |v_\pi^{k+1}(s^n) - v_\pi^k(s^n)|$ # check convergence
  Update $k \leftarrow k + 1$
end while

Obviously, TerminPolicyEval(π, v_π) *does not return the exact values of the policy* π*, but only an estimate of them*

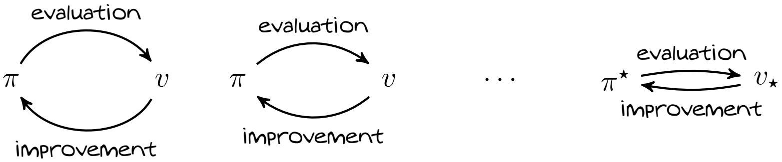
Generalized Policy Iteration

We can come up with various such ideas: *these variants are often called*

Generalized Policy Iteration $\equiv$ *GPI*

These approaches all rely on

back-and-forth computation of policies and values



If designed properly, they all converge to optimal policy and optimal values

Some Final Remarks

+ *We know the algorithms now, but how can we guarantee that they converge? You showed us an simple example that recursion could simply diverge!*
- Well, we can show that what we discussed in this chapter converge: *it comes from the nice properties of Bellman equations*
  - ↳ *There are several proofs; for instance see a proof in Tom Mitchell's notes*

When it comes to practice, most known algorithms are proved to converge to optimal policy and optimal values; however, note that

- *Convergence guarantee is different from the speed of convergence*
  - ↳ *An algorithm might converge, but very slow*
- *If you deal with an unknown algorithm; then, you should make sure that it converges to optimal policy and optimal values*