Explanatory Notes for 6.390

Shaunticlair Ruiz (Current TA)

Fall 2024

Contents

CHAPTER 10

Markov Decision Processes 2 - Optimal Policies, Q-Values

10.2 Finding policies for MDPs

In the previous section, we computed the total value of different **policies**: strategies for how to act, to get the **most rewards.**

- We designed **value functions** in order to **evaluate** policies, and find the best one.
- So, we'll do that: we search for the **optimal** policies, in the **finite** and **infinite** cases.

Definition 1

The **optimal policy** π^* is better than (or as good as) every other policy π, for **every state**.

$$\forall s \in \mathcal{S}: \qquad V_\pi(s) \leqslant V_{\pi^*}(s)$$

There can be **multiple** optimal policies.

10.2.1 Optimal Policies – Finite Horizon, $H = 0, 1$

We could try every possible policy, and compare their **values** directly.

- But that's way too expensive: the number of policies is typically **huge**.

Instead, let's do what we did before: we start with the optimal policy for $h = 0$, and build up a larger **horizon**.

Notation 2

Note that our **policy** can depend on our **horizon**. We'll add notation to accommodate this:

- The **optimal policy** for horizon h is π^*_h.

Example: If it takes 10 steps to reach a very valuable state, you should have a different policy if

- You have $h = 3$ (not enough time to reach it)
- You have $h = 100$ (more than enough time to reach it)

First: $H = 0$. There's no reward, no matter what we do: all policies are the same.

Concept 3

All policies for $h = 0$ are **optimal**.

Next, $H = 1$: we only have to take one action. Thankfully, this is simple: we just take the **action** that **maximizes** our reward.

- This looks like a job for **argmax**.

In the regression chapter, we discussed argmin, but the principle is exactly the same.

Notation 4

Review from the Regression chapter:

The **argmax function** tells you the value of the {input **variable** that gives the **maximum output**.

$$\Theta^* = \arg\max_{\Theta} J(\Theta)$$

The **function we want to maximize** is written to the right, while the **variable we adjust** is written below.

So, we want to know which **action** maximizes the **reward** function.

- We just compute this by comparing all the actions for a single **state**.

$$a^* = \arg\max_a \Big(R(s,a)\Big) \tag{10.1}$$

If we're in state s, we want our **optimal policy** to give this action.

Key Equation 5

The **optimal policy** for horizon $H = 1$ is:

$$\pi_1^*(s) = \arg\max_a \Big(R(s,a)\Big)$$

Remember that we can have multiple optimal policies.

10.2.2 Finite Horizon: $h = 2$

Now, we want to find the policy if $H = 2$.

This has a few complications:

- We have to choose **two** separate actions.
- Our $h = 2$ **action** will affect our state (and reward) at $h = 1$.
  - So, we can't just choose the **action** that gives us the best **immediate reward**: we need to account for *future* reward.

This is what we designed our **value function** V^h for! It allows us to compare policies, while accounting for **future** actions/states/rewards.

- But we mentioned that there are **too many** possible policies to compare all of them. Is there a way we can **narrow it down**?

Here's the trick: we use the same concept from before –

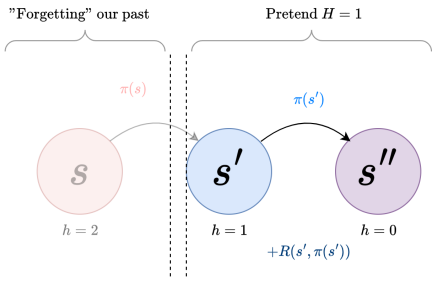
Concept 6

Review from chapter "Markov Decision Processes 1":

If we know our **current state**, we actually **don't care** about what happened during earlier timesteps.

- That means, if our current horizon is $h = 1$, we can pretend as if we're starting a new MDP run with $H = 1$, **ignoring** our original horizon.

Let's ignore whatever our first choice ($h = 2$) was, or our state transition. All we know is that we ended up in state s'.



Our problem is simpler if we simply "forget" our first step: we just pretend we started at s'.

- Now we're in $h = 1$. Thankfully, we already computed the optimal policy, π_1^*.
- Our reward will be

$$R\left(s', \pi_1^*(s')\right) = \max_{a'} \left(R(s', a')\right) \tag{10.2}$$

Since we want the reward, not the action, we'll use "max", not "argmax".

- Now, we know the **second half** of our optimal policy.

Concept 7

The **optimal** choice of action is independent of previous actions: it's only based on our **state** and **horizon**.

- That means that "**the last h steps** of a horizon-H MDP" is **the same** as "**every step** of a horizon-h MDP"
- If we've already computed the optimal policy for a horizon-h MDP, we can **reuse** them at the end of a **longer horizon**.

This **simplifies** the search for our **optimal policy**: we can only focus on policies where those last n steps **match** policy π_n^*.

Example: Suppose you know the best way to finish a game of chess in h turns, starting from any position.

- You can use that knowlegde earlier in the game: those will be the end of a longer, winning strategy, starting earlier.

One weakness of this analogy: in chess, we don't have a well-defined "horizon" for the end of the game. But the same general idea applies.

Since we already know the best action, for each state, at $h = 1$, we don't have to explore as many possible policies!

Now, we can return to our $h = 2$ step, with the knowledge of how we'll act in the future.

We'll re-introduce our value function, so we can figure out which policy is best:

$$V_\pi^2(s) = \overbrace{R\Big(s, \pi(s)\Big)}^{h=2} + \sum_{s'} T\Big(s, \pi(s), s'\Big) \cdot \overbrace{R\Big(s', \pi(s')\Big)}^{h=1} \tag{10.3}$$

We'll adjust this:

- Our current action is chosen by our policy π_2.
- Our next action is chosen by the **maximum** reward for the $h = 1$ step.

$$\overbrace{R\Big(s, \pi_2(s)\Big)}^{h=2} + \sum_{s'} T\Big(s, \pi_2(s), s'\Big) \cdot \overbrace{\max_{a'}\Big(R(s', a')\Big)}^{h=1} \tag{10.4}$$

10.2.3 Q-Values

This is a bit different from V, though. Instead of using the **same policy** for every step, we do something different:

- First, we take one **chosen action** $\pi_2(s)$
- Then for our **second action**, we choose the optimal policy automatically.

We'll come up with a new name for this: a **Q-value function**.

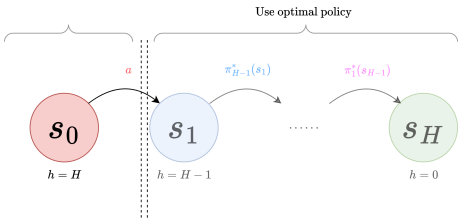
Definition 8

A **Q-value function** $Q^h(s, a)$ (a.k.a "**state-action value function**") is **similar** to a value function $V_\pi^h(s)$, but instead of using the **same policy** for every step, we:

- Choose one action a
- Every choice afterwards is **optimal**.

This is why the policy π has been replaced by a single action a: you only make **one choice**, and all following steps are optimal.

- $Q(s, a)$ tells us the **expected reward**, under these conditions.



We choose our first action, and the rest are optimal.

Despite appearing different, the Q-value function serves all of the roles we previously needed our value function V for.

Concept 9

We can think of our Q-value function $Q^H(s,a)$ as a value function V^H with a **special kind of policy:**

$$Q^H(s,a) \longrightarrow V_\pi^H(s)$$

This "special policy" now depends on **horizon**.

$$\pi_h(s) = \begin{cases} a & h = H \\ \pi_h^*(s) & \text{otherwise} \end{cases}$$

This perfectly matches our current strategy: we try out one action a, and then rely on the **optimal policies** we developed previously.

Q can be seen as a way to "try out" one single action, to add onto your, so far, optimal policy.

$$\overbrace{R\Big(s,\pi_2(s)\Big)}^{\text{One action}} + \sum_{s'} T\Big(s,\pi_2(s),s'\Big) \cdot \overbrace{\max_{a'}\Big(R\big(s',a'\big)\Big)}^{\text{Optimal policies}} \tag{10.5}$$

If we replace $\pi_2(s)$ with a, we have our Q-value function:

$$Q^2(s,a) = \overbrace{R\Big(s,a\Big)}^{\text{One action}} + \sum_{s'} T\Big(s,a,s'\Big) \cdot \overbrace{\max_{a'}\Big(R\big(s',a'\big)\Big)}^{\text{Optimal policies}} \tag{10.6}$$

10.2.4 $H = 2$ completed

Our new Q-value function already takes care of optimizing $h = 1$, so now, we just need to optimize $h = 2$: we need to find the **best action** at $h = 2$, a^*.

$$a^* = \arg\max_a \Big(Q^2(s,a)\Big) \tag{10.7}$$

Key Equation 10

We can use $Q^2(s, a)$ to determine the **optimal policy** for $H = 2$.

- $Q^2(s, a)$ encodes information about **immediate** and **future** rewards, while **optimizing** those future steps.

This allows us to directly compare different actions a, searching for an optimal policy:

$$\pi_2^*(\mathbf{s}) = a^* = \arg\max_a \Big(Q^2(s, a)\Big)$$

This action will be chosen for our optimal policy π_2^*.

10.2.5 $H = 2$ Extended Solution (Optional)

Here's the un-compressed version: it's a lot messier, but it shows more of what's going on.

Remark (Optional) 11

The **optimal policy** for horizon $H = 2$ comes in two stages:

- We compute the **optimal policy** π_1^* and **reward** for our second step, $h = 1$.
  - This tells us how **valuable** each $h = 1$ state s' is.
- We compute the **optimal action** a^* for our first step, $h = 1$, by factoring in
  - The **immediate** reward, $R\Big(s, \pi(s)\Big)$
  - The **average rewards in the next step**, based on all possible outcomes.

$$\pi_2^*(\mathbf{s}) = a^* = \arg\max_a \left\{ \overbrace{R\Big(s, a\Big)}^{\text{Immediate Reward}} + \sum_{s'} T\Big(s, a, s'\Big) \cdot \overbrace{\max_{a'}\Big(R(s', a')\Big)}^{\text{(Optimized) Future Reward}} \right\}$$

We can see that we **separately** optimize our two steps:

- π_1 first, to get π_1^*: this gives us our action a'.

$$\max_{a'}\Big(R(s', a')\Big) \tag{10.8}$$

- Then, we use π_1^* to find π_2^*: this gives us our action a.

$$\arg\max_a \Big(\dots\Big) \tag{10.9}$$

10.2.6 $H = 3$ and beyond

Introducing Q-values has given us the last tool we need to complete our finite-horizon MDP solution.

Let's use everything we've built so far:

- After our first step, we're in state s', with $H = 2$: we already determined what our policy should be, and what the **value** of that policy is:

The value includes both $h = 1$ and $h = 2$.

$$\pi_2^*(s') = \arg\max_{a'}\left(Q^2(s', a')\right) \tag{10.10}$$

$$\text{Optimal Value} = \max_{a'}\left(Q^2(s', a')\right) \tag{10.11}$$

- Our **first step** is the only one we choose, giving us an immediate reward:

$$R\left(s, a\right) \tag{10.12}$$

This matches our description for the Q-value function: all we have to do is account for different possible s' values, with $T(s, a, s')$.

$$Q^3(s, a) = R\left(s, a\right) + \sum_{s'} T\left(s, a, s'\right) \cdot \max_{a'}\left(Q^2(s', a')\right) \tag{10.13}$$

This strategy, conceptually, works exactly the same, for any possible value of H:

Key Equation 12

The Q-value function for horizon H can be written as:

$$Q^H(s, a) = \overbrace{R\left(s, a\right)}^{\text{First reward}} + \overbrace{\sum_{s'} \underbrace{T\left(s, a, s'\right)}_{\text{Chance of } s \to s'} \cdot \underbrace{\max_{a'}\left(Q^{H-1}(s', a')\right)}_{\text{Optimize next step}}}^{\text{Future rewards}}$$

Or, without extra annotation:

$$Q^H(s, a) = R\left(s, a\right) + \sum_{s'} T\left(s, a, s'\right) \cdot \max_{a'}\left(Q^{H-1}(s', a')\right)$$

Note that horizon H relies on optimizing horizon $H - 1$, similar to value iteration.

And based on this Q-value, we can determine the optimal action for our current state (and thus, our **optimal policy**):

Key Equation 13

We can use $Q^H(s, a)$ to determine the **optimal policy** for horizon H.

$$\pi^*_H(s) = \arg\max_a \left(Q^H(s, a)\right)$$

We can use this form of equation to get **every optimal policy**.

Note some important reminders. First, make sure to distinguish between Q and V.

Clarification 14

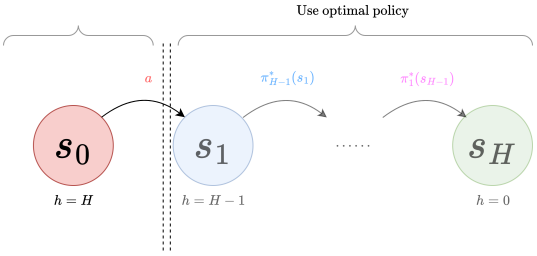
Let's compare our **value function** equation V, to our **Q-value function** equation:

- For our Q-value, we choose one action, and the remainder are **optimal**.

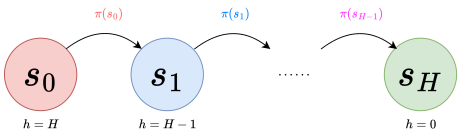
$$Q^H(s, a) = R(s, a) + \sum_{s'} T(s, a, s') \cdot \max_{a'} \left(Q^{H-1}(s', a')\right)$$

- Meanwhile, V relies on the **same policy** π for all actions.

$$V^H_\pi(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s') \cdot V^{H-1}_\pi(s')$$



Our first action is chosen manually: after that, we use an optimal policy.



All of our actions are chosen by our (potentially non-optimal) policy π.

10.2.7 Finite-Horizon Q-Value MDP solution

We have a method for creating the optimal policy:

- Choose any action for $\pi_0^*(s)$.
- Maximize $R(s,a)$ for $a = \pi_1^*(s)$
- Maximize $Q(s,a)$ for $a = \pi_h^*(s)$, if $h \geqslant 2$.

For horizon H, we can **re-use** the optimal policies for shorter horizons.

- This concept is encoded by the way Q-values work.
- So, we can use these Q-values to find the optimal policy:

So, our focus is on those Q-values. We'll compute them as we progressively increase the horizon:

$$Q^0(s,a) = 0 \tag{10.14}$$

$$Q^1(s,a) = R\Big(s,\pi(s)\Big) + 0 \tag{10.15}$$

$$Q^2(s,a) = R\Big(s,a\Big) + \sum_{s'} T\Big(s,a,s'\Big) \cdot \max_{a'}\Big(Q^1(s',a')\Big) \tag{10.16}$$

And if we keep going...

$$Q^H(s,a) = R\Big(s,a\Big) + \sum_{s'} T\Big(s,a,s'\Big) \cdot \max_{a'}\Big(Q^{H-1}(s',a')\Big) \tag{10.17}$$

10.2.8 Dynamic Programming

For value function V and Q-values, we've been using a very particular strategy for computing our solutions.

- We solve **simpler subproblems** (like V^1) and **save the results**.
- Later, we **re-use** those solutions for more complicated problems (like V^2).

This saves us a lot of work:

- **Example:** We could re-compute all the V^1 values, every time we need a V^2 value.
- But if, instead, we just store those values and use them later, we save them.
- The benefits are more obvious for computing V^{200}: it would be a nightmare to have to consider every possible sub-problem.

We don't have to pay attention to V^2 when doing V^{200}: that information is stored in V^{199}.

We call this strategy **dynamic programming**, despite it being neither "dynamic", nor "programming".

Definition 15

Dynamic Programming is a strategy for solving complex problems that can be broken up into simpler, similar problems ("subproblems").

- First, we solve the **subproblems**, and **save** the result.
- These "mini-solutions" are re-used as a part of larger, more **complicated** problems.

Because a single sub-solution can be used **many times**, you save time by storing the answer, rather than re-computing it every time you need it.

We use it for every step of our value/Q-value calculation:

$$Q^H(s,a) = R\Big(s,a\Big) + \sum_{s'} T\Big(s,a,s'\Big) \cdot \max_{a'}\Big(Q^{H-1}(s',a')\Big) \tag{10.18}$$

Q^H always requires us to use Q^{H-1}.

- $Q^H(s,a)$ has an equation for each state-action pair (s,a).
  - Each of these equations will use Q^{H-1}: that involves using Q^{H-1} **many times**.
- It would be much slower if, for every $Q^H(s,a)$, we had to **re-compute** all of Q^{H-1}.

10.2.9 Dynamic Programming Performance (Optional)

Let's compare the performance of working with, and without dynamic programming. Our goal is to compute $Q^h(s, a)$.

$$|\mathcal{S}| = m, \qquad |\mathcal{A}| = n \tag{10.19}$$

We'll need this concept:

Concept 16

If we have a pair of elements, (a, b), the total number of **possibilities** is:

- The number of possible a values, **times** the number of possible b values.

This idea works for larger sequences: you just multiply together the **possible choices** at each step.

We'll count the amount of work we need with, and without dynamic programming.

First: **without dynamic programming**.

One way to compute the value of $Q^H(s, a)$ is to consider **every possible outcome** of that action, and find the optimal one.

- **How many** outcomes are there? One outcome has one **state-action pair** (mn pairs), for each **timestep** (h timesteps).

m states, n actions: mn pairs of states and actions.

The total number of possible outcomes is $(mn)^h$. So, our total work is proportional to that:

At each step, we have mn possible pairs. So, for two timesteps, we have to choose from mn options, twice: $(mn)^2$.

$$O\left((mn)^h\right) \tag{10.20}$$

We won't teach O-notation here.

Let's **use dynamic programming** this time.

- For our **final timestep**, we have mn possible state-action pairs. We pick the optimal action for each state, and we **save** its value as $Q^h(s, a)$.
- For our previous timestep, we **don't care** about the possibilities in the final timestep: we just trust $Q^h(s, a)$, and use it to select one of our mn actions.

So, at each timestep, we compare mn elements: we don't have to consider combinations across different timesteps.

However, we do still have to do this calculation once per timestep: h times. We get mnh.

$$O(mnh) \tag{10.21}$$

Concept 17

Using **dynamic programming** for Q-value computation dramatically increases **efficiency**.

Instead of taking $O((mn)^h)$ time, we need $O(mnh)$ time.

- That's **exponentially** faster!

In short: the difference is that dynamic programming allows us to "ignore" other timesteps, and just rely on Q-values.

- Without dynamic programming, we have to "remember" other timesteps, and consider **combinations** of state-action pairs.

10.2.10 Optimal Policies – Infinite Horizon

Now, we need to figure out how to get the optimal **infinite-horizon** policy.

We'll start off with our finite Q-value equation:

$$Q^H(s,a) = R(s,a) + \sum_{s'} T(s,a,s') \cdot \max_{a'} \left(Q^{H-1}(s',a')\right) \tag{10.22}$$

All we do is **remove the horizon**, and **add the discount factor**.

- In this situation, we take one action, and then take **optimal** actions for every step after, forever (or until our model terminates).
- We notate this "optimal value" as Q.

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \cdot \max_{a'} \left(Q^*(s',a')\right) \tag{10.23}$$

Definition 18

$Q^*(s,a)$ is the **total discounted reward**, assuming you:

- Take action a in state s.
- Choose the **optimal action** for all future states.

Now, our Q-value function is an equation of **itself**.

Key Equation 19

The Q**-value function** for **infinite horizon** can be written as:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \cdot \max_{a'} \left(Q^*(s',a')\right)$$

Something similar happened before, when we were doing value functions. Why is that?

- We're in the **same situation** before and after we take one (successful) step: we **don't know** how long it will be until the model terminates.
- In fact, nothing has changed: the chance of failing after one more step is still γ.

Because "maximizing Q" is how we determine our policy, we see that:

Concept 20

The optimal policy for an **infinite-horizon MDP** is **the same**, at any timestep, regardless of past events.

We call this a **stationary** optimal policy, because it doesn't change over time.

Our goal is to look for one of these "stationary" optimal policies.

There can be several optimal policies. We're only looking for one.

10.2.11 Finding an Optimal Policy: Value Iteration

We have a problem, though. When computing the **infinite value function**, we were able to solve a system of **linear equations**.

- But our Q-value function is **non-linear** this time, because of the **max** operation. We can't solve that!

Is there even a solution, in this complex system? It turns out there **definitely is**:

Theorem 21

Our system of Q-values has a **unique solution**.

This allows us to compute an **optimal policy**.

There may be more than one optimal policy, but they'll all have the same, **unique** Q-values.

In other words, each policy is worth the same average reward.

Instead of linear solving, we'll try something different:

- Our Q-value function could be said to have an "infinitely far" horizon, with a $1-\gamma$ chance of terminating every timestep.
- We could **approximate** this by computing horizon $h = 1, 2, 3, \ldots$: as our horizon gets really big, we hope this is **similar** to an infinite horizon.

Concept 22

We want to approximate an **infinite horizon** with a **really large, finite horizon**.

To match the infinite case, we'll include a chance of **termination**, $(1-\gamma)$.

This is identical to our equations for finite horizon, but with a γ term included.

$$Q^H(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \cdot \max_{a'} \left(Q^{H-1}(s',a') \right) \tag{10.24}$$

This is called **value iteration**.

However, we don't really care about the H of our fake, **finite** horizon: our goal is to approximate the infinite-horizon Q.

Notation 23

For each step of our **value iteration** process, we'll distinguish between two parts of our Q^* approximation:

- Q_{old}: the $H-1$ horizon: it's our **previous**, less-accurate Q^* approximation.
  - We use it to compute Q_{new}.
- Q_{new}: the H horizon: it's our **newest**, more accurate approximation.

After each timestep, Q_{new} **replaces** Q_{old}:

- Every approximation is used to create a new, better approximation.

With this, we can properly represent Q^*.

$$Q_{new}(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \cdot \max_{a'} \left(Q_{old}(s',a') \right) \tag{10.25}$$

Definition 24

Infinite-horizon value iteration is a process where we approximate the **infinite horizon Q-value** Q^* with a **large, finite horizon Q-value** Q^H.

To accomplish this, we compute Q^H, combined with a **termination** chance ($p = 1-\gamma$), from the **infinite horizon** problem.

- Each Q^h term is used to aim for a **better approximation** of Q^* than the one before.

It's more accurate to call these approximations of Q^*, rather than finite-horizon values Q^h. So, we'll change up our notation:

- We use Q_{old} (our old approximation) to compute Q_{new} (a new approximation), using the **finite-horizon equation** (discounted with γ).

Throughout the process, we use the following equation:

Key Equation 25

The equation for **infinite-horizon value iteration** is:

$$Q_{new}(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \cdot \max_{a'} \left(Q_{old}(s',a') \right)$$

Once our Q value is **close to** Q^*, we can find an **optimal policy**:

$$Q_{new} \approx Q^* \implies \pi^*(s) = \arg\max_a \left(Q_{new}(s,a) \right)$$

How do we know that value iteration converges at all? We have a **theorem** for that, in 12.2.14.

Next, we'll figure out when to **stop** value iteration: when are we ready to use our Q^* approximation?

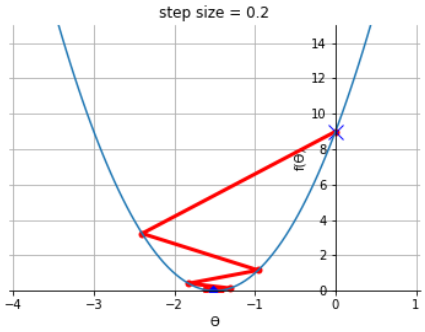
10.2.12 Convergence of Value Iteration

How do we know when to stop running our value iteration? We want Q_{new}, to **converge** to Q^*: it gets **close** to the answer we want!

How do we know *when* we're converging?

- We can use gradient descent as an example:

We start at the blue x on the right. Each red line shows us "moving" to a new, better position.



As we get **close** to the solution, our steps have to become **very small**.

If we're close to the solution and take a **big** step, we'd get **further away** again! So, our steps **must** get smaller.

We can see a pattern as we converge:

Concept 26

As we get close to **convergence**, our sequence will typically **stabilize**: the values become more and more **similar**.

We can use this "slowing down" progress to **detect** convergence.

Let's translate this to **value iteration**:

- As we converge, each Q_{new} value will be **very close** to the previous one.
- So, we'll detect convergence by seeing when our approximation is **changing by a very small amount**.

Concept 27

We **terminate** our value-iteration process when our **newest approximations** Q_{new} become **very similar** to each other.

If our approximations are similar (they change very little between timesteps), that means we're **converging** to Q^*.

Even if Q_{new} converges, how do we know it converges to Q^*? We have **another theorem** for that, in 12.2.14.

10.2.13 Value Iteration: Termination Condition

Let's figure out how to represent this mathematically.

- We'll compare the two most recent approximations: Q_{new} and Q_{old}.
- We want these two **value functions** to be **similar**. How do we measure this?

Concept 28

To compare two functions, we see how different their **outputs** are, for each **input**.

$$\left|\left|f(x) - g(x)\right|\right|$$

So, we'll compare the values that we get for each (s, a) pair.

$$\left|\left|Q_{new}(s,a) - Q_{old}(s,a)\right|\right| \tag{10.26}$$

We want our value functions to be **similar**: that means they need to be similar for **all inputs**.

- We'll set a **maximum** for how different each output can be, called ε.

Definition 29

We want to see when Q_{new} and Q_{old} are **similar**.

We'll say that they're similar if **every output** is closer than a distance of ε:

$$\overbrace{\forall s : \forall a :}^{\text{Every } (s,a) \text{ pair}} \quad \overbrace{\left|\left|Q_{new}(s,a) - Q_{old}(s,a)\right|\right| < \varepsilon}^{\text{...Must be at least this similar}}$$

If we want to check if **all** of them are similar, we can focus on the **worst case**:

- Which (s, a) makes them the **most different**?

$$\max_{s,a}\left(\left|\left|Q_{new}(s,a) - Q_{old}(s,a)\right|\right|\right) \tag{10.27}$$

Notation 30

When we need to take the maximum over **multiple inputs**, we include **both of them** underneath the **max** notation.

$$\max_{x,y}\Big(f(x,y)\Big)$$

The above asks, "if we can choose x and y to be anything, what's the **largest value** of f we can get?"

This equation will tell us whether we're finished.

Key Equation 31

The following equation tells us, "if we compare Q_{new} to Q_{old}, what's the **biggest difference** between their outputs?"

$$\max_{s,a}\left(\Big|\Big|Q_{new}(s,a)-Q_{old}(s,a)\Big|\Big|\right)$$

If their "biggest difference" is small ($<\varepsilon$), then you could say the two functions are **similar** for every input.

$$\max_{s,a}\left(\Big|\Big|Q_{new}(s,a)-Q_{old}(s,a)\Big|\Big|\right)<\varepsilon$$

This is our **termination condition**: once we reach this condition, our Q^* approximation is "good enough".

Finally, we have a complete value-iteration process.

Definition 32

To do Q-**value iteration**, we follow these steps:

1. Start with $Q_{new} = 0$.
2. Set $Q_{old} = Q_{new}$. This moves us forward 1 timestep.
3. Update Q_{new}.

$$Q_{new}(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \cdot \max_{a'} \left(Q_{old}(s',a')\right)$$

4. Repeat step 2-3 until we **converge**:

$$\max_{s,a} \left(\left|\left| Q_{new}(s,a) - Q_{old}(s,a) \right|\right| \right) < \varepsilon$$

5. Return the optimal policy.

$$\pi^*(s) = \arg\max_a \left(Q^*(s,a)\right)$$

If we write this as pseudocode:

INFINITE-HORIZON-VALUE-ITERATION($\mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon$)

```
for s ∈ S, a ∈ A :                    # Every input pair
    Q_old(s, a) = 0                   # Start from Q^0 = 0

while True: # Continue until converged

    for s ∈ S, a ∈ A :                # Update Q_new

        Q_new(s, a) = R(s, a) + γ Σ_s' T(s, a, s') · max_a' (Q_old(s', a'))

    if max_{s,a} (||Q_new(s, a) − Q_old(s, a)||) < ε      #If converged, we're finished
        return Q_new

    Q_old = Q_new                     #Re-use for next iteration
```

10.2.14 Convergence Theorems

We've made a couple pretty large **assumptions**: thankfully, each one has a **theoretical** justification.

In order to discuss these theorems, we need to clarify some notation:

Notation 33

Suppose we have run our value iteration for n steps (not necessarily enough for convergence within ε). For simplicity, let's say $Q = Q_{new}$: it's our current best approximation of Q^*.

Our policies:

- π^* is the optimal policy, based on Q^*.
- π_Q guesses the optimal policy, based on Q.

Our value functions:

- V_{π_Q} is the reward of using π_Q forever.
- V_{π^*} is the reward of using π^* forever.

If we succeed, then $Q \approx Q^*$, and $\pi_Q = \pi^*$.

Our first assumption: "value-iteration makes our approximation better".

Theorem 34

$$\max_s \left(\left| \left| V_{\pi_Q}(s) - V_{\pi^*}(s) \right| \right| \right) \quad \text{never increases}$$

or,

$$\overbrace{\max_s \left(\left| \left| V_{\pi_Q}(s) - V_{\pi^*}(s) \right| \right| \right)}^{\text{Our worst-performing state } \pi(s)} \quad \overbrace{\text{never increases}}^{\text{Never gets worse than this}}$$

By taking the max, we're getting the "worst" performing state for π_Q: we'll call this s_{bad}.

- This sets a **limit** on how bad our model can be.
- Iteration can't make any state worse than s_{bad} currently is.
- s_{bad} can only stay the same, or get better!

Another way to say "never increases" is "decreases monotonically".

This seems pretty weak: it's nice to know that value-iteration can't make π_Q much worse, but how do we know it gets better? How do we know it converges?

Theorem 35

When we run **value iteration** with ϵ, we will eventually **improve** Q enough to meet our requirement:

$$\text{After value iteration,} \quad \overbrace{\max_s \left(\left| \left| V_{\pi_Q}(s) - V_{\pi^*}(s) \right| \right| \right) < \epsilon}^{\text{Q will approach } Q^* \text{ (within distance } \epsilon)}$$

This means that if we run value iteration long enough, it **will converge**, and get as close to Q^* as we want.

That's great! Q approaches Q^*, when we run value iteration.

We just have one more problem: just because Q is **close** to Q^*, doesn't mean they're close enough to get the optimal policy we want, π^*.

- Can get that policy?

Theorem 36

If we choose the right ϵ, our policy π_Q **will be** the optimal policy π^*.

$$\overbrace{\exists \epsilon > 0}^{\text{There is an } \epsilon \text{ where}} : \qquad \overbrace{\pi_Q = \pi^*}^{\text{We find the optimal policy}}$$

We don't know what value of ϵ is required, but it's good to know that it always exists: we can always find the optimal policy.

One last useful computational trick:

Concept 37

We can run the different parts of value-iteration **in parallel**, out-of-sync with each other.

As long as we update each (s, a) "infinitely many times" (as many times as we need), we will still **converge**.

10.3 Terms

Section 12.0

- Timestep t (Review)
- State s_t
- Input x_t
- Transition function f_s
- Output y_t
- Output function f_o
- State Machine
- Finite State Machine
- State Transition Diagram
- Linearity
- Time-Invariance
- Linear Time-Invariant System (LTI)
- Actions a_t
- Deterministic State Machine
- Probabilistic State Machine
- Reward function $R(s, a)$
- State-action pair
- Stochastic (Review)

Section 12.1

- Action space
- State space
- Transition Model $T(s, a, s')$
- (Probabilistic) State-Transition Diagram
- Transition Matrix $\mathcal{T}(a)$
- Markov Decision Process $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$

- Policy $\pi(s)$
- Value Function
- Finite Horizon
- Horizon h
- Timestep t (Review)
- MDP Termination
- Finite Horizon Value Function $V_\pi^h(s)$
- Expected Value
- Weighted Average (Review)
- Probability Distribution (Review)
- Infinite Horizon
- Infinite Horizon Value Function $V_\pi(s)$
- Discounting
- Discount Factor
- Discounted Average
- MDP Lifespan
- Memorylessness
- Markov Property (Optional)

Section 12.2

- Optimal Policy
- Argmax (Review)
- Q-Value Function (State-action value function)
- Dynamic Programming
- Value Iteration
- Convergence (Review)