STA 4273: Minimizing Expectations

Lecture 2 - Basic tools

Chris J. Maddison

University of Toronto

Announcements

- Project handout out later today, sorry for delay (grad admissions).
- Please fill out the presentation sign-up Google Form (see Announcements tab on Quercus)!
  - I will settle those assignments soon.
  - If you haven't signed up, I will assign randomly.
- Office hours now held on Zoom—GatherTown was not stable enough.
- A word about code notebooks.

Code notebook

- Do not re-implement the whole paper.
- The requirement is to demonstrate *one or two* key ideas:
  - Can reimplement a toy experiment.
  - Can create an illustrative figure.
- We will be generous, but ask me for tips if you need them.
- Example codebook here (and on course website).

Recall

Let X be random variable taking values in $\mathcal{X}$ (some simple space, e.g. finite set, countably infinite set, or $\mathbb{R}^D$). Let $f : \mathcal{X} \times \mathcal{Q} \to \mathbb{R}$ be a function, where $\mathcal{Q}$ is a set of probability densities over $\mathcal{X}$. Recall, that in this course we are interested in problems of the form

$$\inf_{q \in \mathcal{Q}} \mathbb{E}_{X \sim q}[f(X, q)]$$

- f itself may be an expectation over another random variable whose distribution we do not control, as in reinforcement learning.
- Maximizing over q is equivalent.
- Captures entropy penalties.
- We assume that the minimum is achieved (clearly will depend on $\mathcal{Q}$).

Recall

- An example are Markov Decision Processes (MDP), defined by
  - $\mathcal{S}$: State space. Discrete or continuous, but let's assume it is finite.
  - $\mathcal{A}$: Action space. We consider finite action space.
  - $p(s_{t+1}|s_t, a_t)$: Environment transition probability distribution.
  - $p(s_0)$: Initial state distribution.
  - $r(s_t, a_t)$: Bounded reward function (can be a random variable).
  - γ: Discount factor ($0 < \gamma \leq 1$).

Recall

- The agent operates in an MDP environment using a policy:
  - $\pi(a_t|s_t)$: a stochastic policy that the agent uses to choose action.
  - $\pi(s_t)$: a deterministic policy that the agent uses to choose action.
- The agent's objectives is to maximize its return:

$$\arg\max_{\pi} \mathbb{E}_{\tau\sim p}\left[r_\gamma(\tau)\right] = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

where $\tau = (s_0, a_0, s_1, a_1, \ldots) \sim p(s_0)\pi(a_0|s_0)p(s_1|s_0, a_0)\ldots$

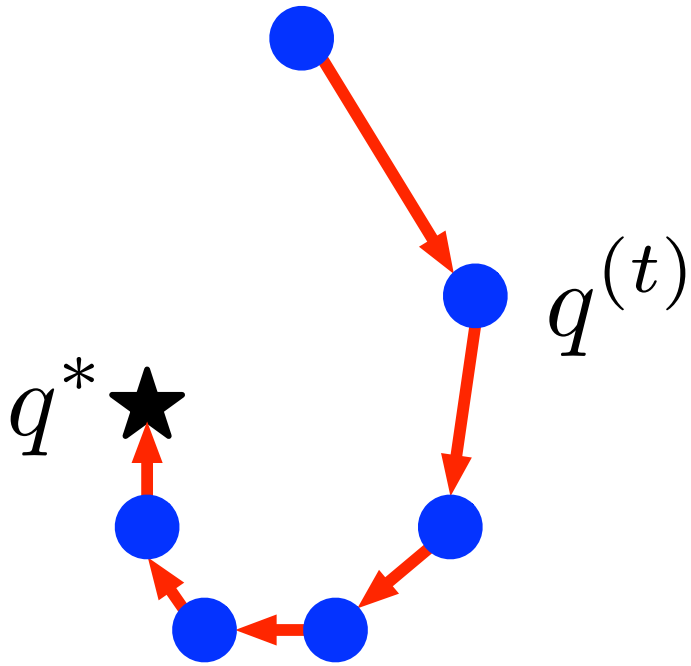
Today

We will study iterative methods, which are methods that construct a sequence of $q^{(t)}$ (typically in $\mathcal{Q}$) that converge

$$q^{(t)} \to q^*$$

to an element of $\mathcal{Q}$ that is either in its argminimum or a local minimum.

These ideas form the foundation of the methods that we will look at throughout the course.



Outline

- Direct gradient-based methods
  - Gradient descent
  - Stochastic gradient descent
- Dynamic programming
  - Value iteration
  - Policy iteration

Direct gradient-based methods

Policy parameters and gradients

- For the first half of the lecture, let's assume that $\mathcal{Q}$ is a parametric family of mass functions, i.e.,

$$\mathcal{Q} = \{q_\theta \mid \theta \in \mathbb{R}^D\}$$

  Inspired by reinforcement learning, let's call $q_\theta \in \mathcal{Q}$ a policy.
- We will also assume that

$$J(\theta) = \mathbb{E}_{X \sim q_\theta}[f(X, \theta)]$$

  is continuously differentiable in θ (where we abused notation for the second argument of f).

Policy parameters and gradientss—example

As an example, consider a simple softmax policy family,

$$\mathcal{Q} = \left\{ q_\theta = \left(\frac{\exp(\theta_i)}{\sum_{j=1}^{N} \exp(\theta_j)} \right)_{i=1}^{N} \middle| \, \theta \in \mathbb{R}^N \right\}$$

- $f : \{1, \ldots, N\} \to \mathbb{R}$ as any function that doesn't depend on θ.
- $\mathcal{X} = \{1, \ldots, N\}$.
- Does $\nabla J(\theta) := (\partial J(\theta)/\partial \theta_i)$ exist and is it continuous?

Policy parameters and gradients—example

$$\begin{aligned}\nabla J(\theta) &= \nabla \sum_{i=1}^{N} f(i) q_\theta(i) \\ &= \sum_{i=1}^{N} f(i) \nabla q_\theta(i) \\ &= \sum_{i=1}^{N} f(i) q_\theta(i) \left(e_i - q_\theta\right)\end{aligned}$$

where $e_i \in \mathbb{R}^N$ is the ith standard basis vector. This is continuous in θ.

Gradient descent

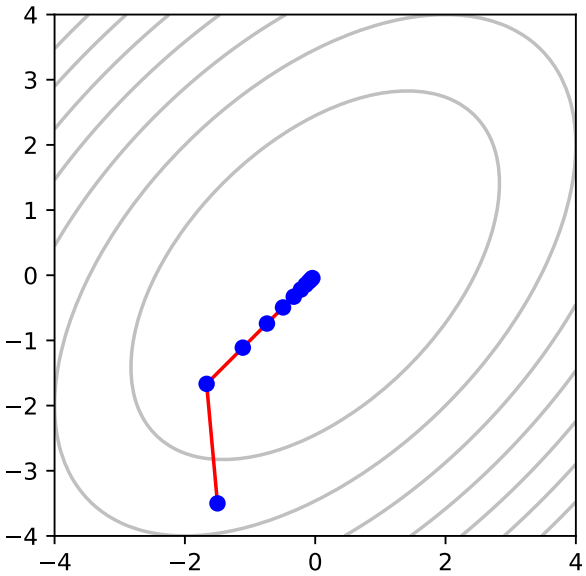
- We can use $-\nabla J(\theta)$ to optimize J.
- Note, that $\nabla J(\theta)$ is the direction from θ in which J has the highest rate of decrease.

$$-\nabla J(\theta) \in \arg\min \left\{ \frac{d^\top \nabla J(\theta)}{\|d\|} \,\middle|\, d \neq 0 \right\}$$

- Gradient descent: move in direction of $-\nabla J(\theta)$ with step size $\eta > 0$,

$$\theta^{(t+1)} := \theta^{(t)} - \eta \nabla J(\theta^{(t)})$$

which forms our sequence of iterates $q^{(t)} = q_{\theta^{(t)}}$. Does it converge?



Lipschitz gradients

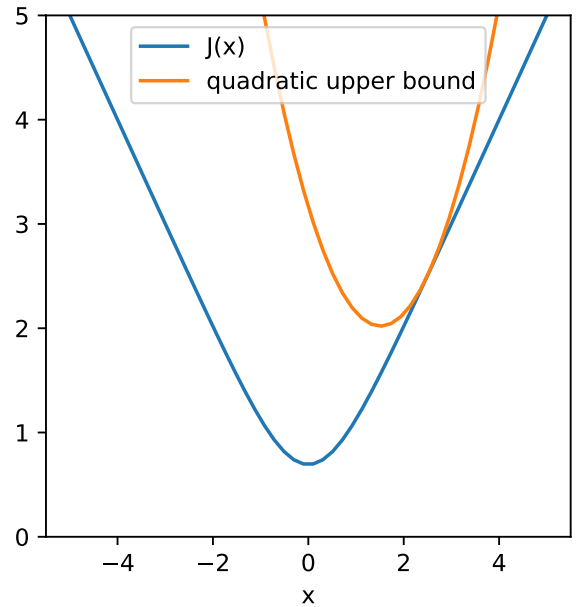
- To guarantee convergence, the most common sufficient condition is L-Lipschitz gradients for $L > 0$, i.e., for all $x, y \in \mathbb{R}^D$

$$\|\nabla J(x) - \nabla J(y)\| \leq L\|x - y\|$$

- One can show (good practice) that this implies

$$J(x) \leq J(y) + \nabla J(y)^\top (x - y) + \frac{L}{2}\|x - y\|^2 := U_L(x, y)$$

- I.e., we can form a quadratic upper bound $U_L(x, y)$ of $f(x)$ about y.



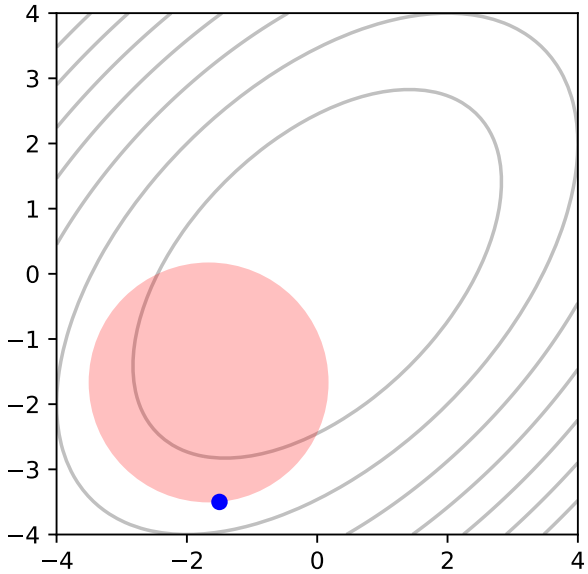
Convergence of gradient descent

- Consider iterate $\theta^{(t)}$ (blue dot).
- Assume J has L-Lipschitz gradients and form the quadratic upper bound $U_L(x, \theta^{(t)})$ about the point $\theta^{(t)}$.
- The red region is the sublevel set

$$S = \{x : U_L(x, \theta^{(t)}) \leq J(\theta^{(t)})\}$$

- We can pick any point x in this sublevel set and be guaranteed a descent in J:

$$\forall x \in S,\ J(x) \leq U_L(x, \theta^{(t)}) \leq J(\theta^{(t)})$$

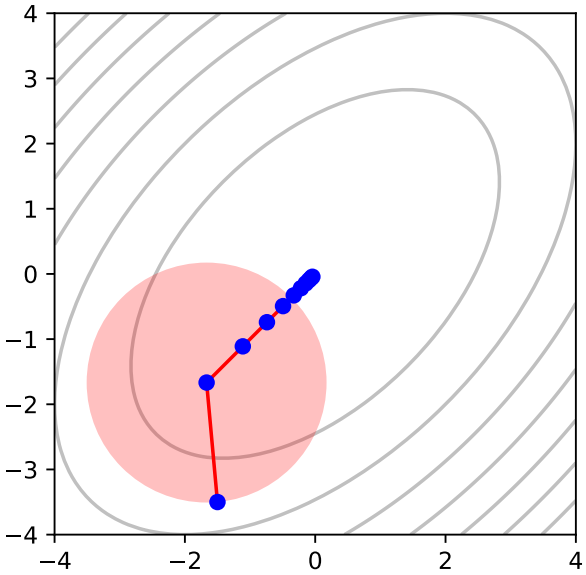


Convergence of gradient descent

- Gradient descent can be re-written as a minimization (good practice).

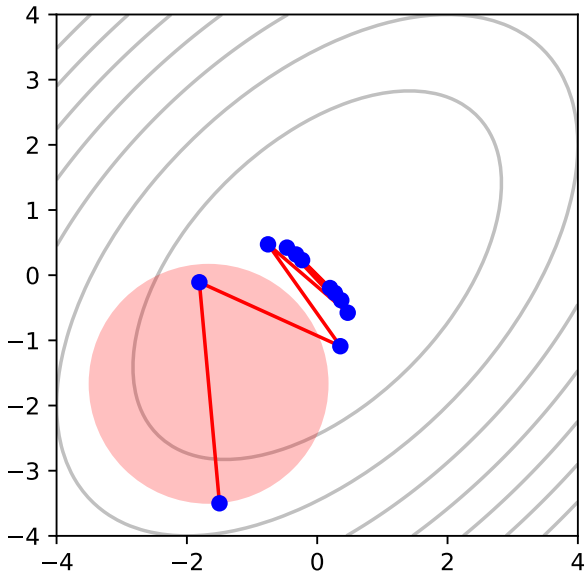
$$\theta^{(t+1)} = \arg\min_x U_{1/\eta}(x, \theta^{(t)})$$

- So, if $\eta = 1/L$, we step right to the centre of the sublevel set.
- Because the quadratic upper bound holds for all $\mathbb{R}^D$, we get guaranteed convergence in f (assuming fixed step-size).



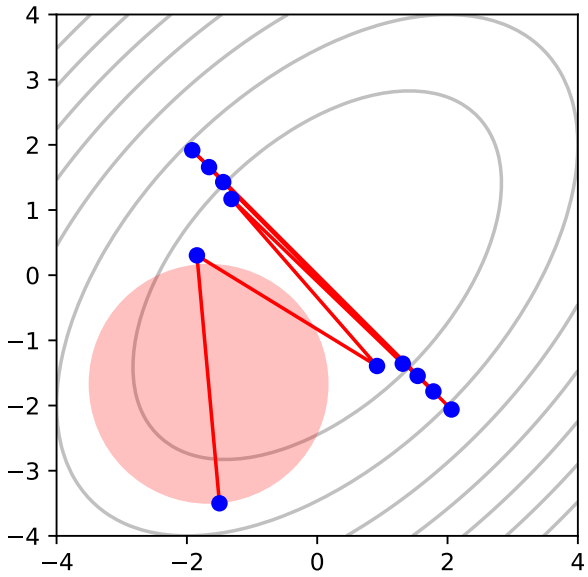
Convergence of gradient descent

- We can actually step a bit farther than $\eta > 1/L$, as long as we do not step past the boundary of the sublevel set.



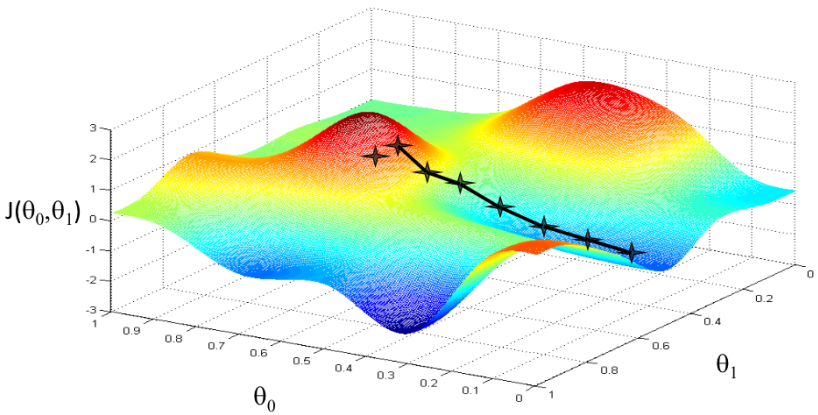
Convergence of gradient descent

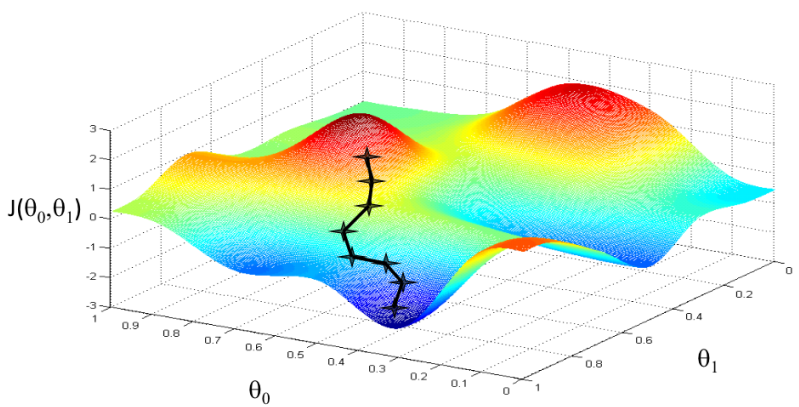
- But, if $\eta > 2/L$, we step past the boundary of the sublevel set and we lose our descent guarantee.
- Divergence may happen.



Convergence of gradient descent

- If $\eta < 2/L$, gradient descent converges to a critical point θ^*, i.e., $\nabla J(\theta^*) = 0$, which is not necessarily an optimum.
  - This just tells us that the function is perfectly flat at this point.
- Typically, gradient descent will converge to a local minimum, and which local minimum depends on initialization.





Stochastic gradient descent

- Computing ∇J is usually prohibitively expensive.
- Consider an episodic MDP with finite, fixed T, policy $\pi_\theta(a_t|s_t)$ that is continuosly differentiable in $\theta \in \mathbb{R}^D$, and discount $\gamma = 1$. I.e., we are interested in

$$J(\theta) = \sum_\tau r(\tau) p(s_0) \pi_\theta(a_0|s_0) \dots p(s_T|s_{T-1}, a_{T-1}) \pi_\theta(a_T|s_T)$$

- Computing ∇J exactly scales like $\mathcal{O}(|\mathcal{A}|^T |\mathcal{S}|^T)$.

Computing gradients and stochastic gradient descent

On the other hand,

$$\begin{aligned}
\nabla J(\theta) &= \nabla \sum_{\tau} r(\tau) p(s_0) \pi_\theta(a_0|s_0) \dots p(s_T|s_{T-1}, a_{T-1}) \pi_\theta(a_T|s_T) \\
&= \sum_{\tau} r(\tau) \nabla \left(\prod_{t=0}^{T} \pi_\theta(a_t|s_t) \right) p(s_0) \prod_{t=1}^{T} p(s_t|s_{t-1}, a_{t-1}) \\
&= \sum_{\tau} \sum_{t=0}^{T} r(\tau) \nabla \pi_\theta(a_t|s_t) \prod_{t' \neq t} \pi_\theta(a_{t'}|s_{t'}) p(s_0) \prod_{t=1}^{T} p(s_t|s_{t-1}, a_{t-1}) \\
&= \sum_{\tau} \sum_{t=0}^{T} r(\tau) \nabla \log \pi_\theta(a_t|s_t) p(\tau) \\
&= \mathbb{E}_{\tau \sim p} \left[\sum_{t=0}^{T} r(\tau) \nabla \log \pi_\theta(a_t|s_t) \right]
\end{aligned}$$

where we used $\nabla p_\theta(x) = p_\theta(x) \nabla \log p_\theta(x)$.

Computing gradients and stochastic gradient descent

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim p}\left[\sum_{t=0}^{T} r(\tau) \nabla \log \pi_\theta(a_t|s_t)\right]$$

We can use this to form a gradient estimator:

- Simulate a random trajectory $\tau \sim p$ in $\mathcal{O}(T)$ by interacting with the MDP using π_θ.
- Form a Monte Carlo estimate of $\nabla J(\theta)$ using $\sum_{t=0}^{T} r(\tau) \nabla \log \pi_\theta(a_t|s_t)$.

Stochastic gradient descent (SGD)

- The main idea behind stochastic gradient descent is to use a Monte Carlo estimate of $\nabla J(\theta)$ instead.
- More precisely, we can use a gradient estimator, which is built from a random variable $\xi \sim p$ taking value in Ξ and a function $g : \mathbb{R}^D \times \Xi \to \mathbb{R}^D$ such that

$$\mathbb{E}_\xi[g(\theta, \xi)] = \nabla J(\theta) = \nabla \mathbb{E}_{X \sim q_\theta}[f(X, \theta)]$$

  - E.g. for our MDP $\xi = \tau$ and $g(\theta, \tau) = \sum_{t=0}^{T} r(\tau) \nabla \log \pi_\theta(a_t|s_t)$.
- Let's analyze the convergence properties.

Stochastic gradient descent (SGD)

What if we use $g(\theta, \xi)$ instead of $\nabla J(\theta)$?

$$\theta^{(t+1)} = \theta^{(t)} - \eta^{(t)} g(\theta^{(t)}, \xi^{(t)}) \tag{1}$$

Theorem

Let J have L-Lipschitz gradients and be bounded below by J^*. Let $g(\theta, \xi)$ be an unbiased estimate of $\nabla J(\theta)$ for some random variable ξ with bounded $\mathbb{E}_{\xi \sim p}[\|g(\theta, \xi)\|^2] \leq \sigma^2$. Let $\xi^{(t)}$ be i.i.d. as ξ and let $\eta^{(t)} > 0$.

The iterates of (1) satisfy,

$$\sum_{t=1}^{K} \eta^{(t)} \mathbb{E}[\|\nabla J(\theta^{(t)})\|^2] \leq J(\theta^{(1)}) - J^* + \frac{L\sigma^2}{2} \sum_{t=1}^{K} (\eta^{(t)})^2.$$

Proof

$$\begin{aligned}
&\mathbb{E}[J(\theta^{(t+1)}) - J(\theta^{(t)})|\theta^{(t)}] \\
&\qquad \leq \mathbb{E}\left[\nabla J(\theta^{(t)})^\top(\theta^{(t+1)} - \theta^{(t)}) + \frac{L}{2}\|\theta^{(t+1)} - \theta^{(t)}\|^2 \middle| \theta^{(t)}\right] \\
&\qquad = \mathbb{E}\left[-\eta^{(t)}\nabla J(\theta^{(t)})^\top g(\theta^{(t)}, \xi^{(t)}) \middle| \theta^{(t)}\right] + \frac{L(\eta^{(t)})^2\sigma^2}{2} \\
&\qquad \leq -\eta^{(t)}\|\nabla J(\theta^{(t)})\|^2 + \frac{L(\eta^{(t)})^2\sigma^2}{2}
\end{aligned}$$

Summing from for $t = 1, \ldots, K$ we get

$$\begin{aligned}
\sum_{t=1}^{K} \eta^{(t)}\mathbb{E}[\|\nabla J(\theta^{(t)})\|^2] &\leq J(\theta^{(1)}) - \mathbb{E}[J(\theta^{(K)})] + \sum_{t=1}^{K} \frac{L(\eta^{(t)})^2\sigma^2}{2} \\
&\leq J(\theta^{(1)}) - J^* + \sum_{t=1}^{K} \frac{L(\eta^{(t)})^2\sigma^2}{2}
\end{aligned}$$

Implications of this theorem I

Under some smoothness assumptions on J and if the step-sizes satisfy

$$\sum_{t=1}^{\infty} \eta^{(t)} = \infty \qquad \sum_{t=1}^{\infty} (\eta^{(t)})^2 < \infty,$$

then Cor. 4.12 of (Bottou et al., 2018) shows that

$$\lim_{t \to \infty} \mathbb{E}[\|\nabla J(\theta^{(t)})\|^2] = 0$$

This is intuitive from the form of the bound

$$\sum_{t=1}^{K} \eta^{(t)} \mathbb{E}[\|\nabla J(\theta^{(t)})\|^2] \leq \mathcal{O}\left(\sum_{t=1}^{K} (\eta^{(t)})^2\right) < \infty$$

which means we get convergence in expectation to a stationary point.

Implications of this theorem II

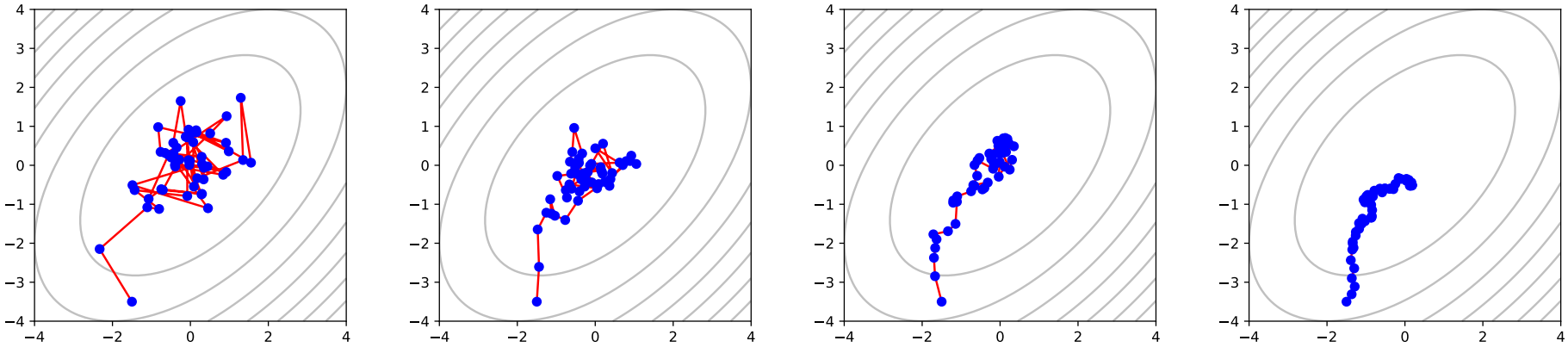
If the step-sizes are constant, then the result simplifies to

$$\sum_{t=1}^{K} \frac{1}{K} \mathbb{E}[\|\nabla J(\theta^{(t)})\|^2] \leq \frac{J(\theta^{(1)}) - J^*}{K\eta} + \frac{L\eta\sigma^2}{2}$$

For a fixed step-size there's some irreducible error, which shrinks as $\eta \to 0$.

Implications of this theorem II

Stochastic gradient descent



smaller fixed step-size $\eta \to 0$ for a fixed σ, K.

Implications of this theorem II

so, we can reduce this error by decreasing the step-size. In particular, if $\eta \propto \sqrt{1/K}$, using a crude simple Markov inequality we get

$$\lim_{K\to\infty} \mathbb{P}(\text{at least one } \|\nabla J(\theta^{(t)})\| < \epsilon) \geq \lim_{K\to\infty} 1 - \mathcal{O}\left(\frac{1}{K^{1/4}\epsilon}\right)$$
$$= 1$$

This has some implications for machine learning.

Implications of this theorem II

By far the most common application of SGD in machine learning is for empirical risk minimization. Here, we have an objective of the form

$$J(\theta) = \sum_{i=1}^{m} \frac{1}{m} f_i(\theta)$$

for $f_i : \mathbb{R}^D \to \mathbb{R}$ differentiable. A valid gradient estimator for this would be to take $\xi \sim \text{unif}\{1, \ldots, m\}$, and take

$$g(\theta, \xi) = \nabla f_\xi(\theta)$$

Implications of this theorem II

- To find a point $\|\nabla J(\theta^{(t)})\| < \epsilon$, gradient descent requires $\mathcal{O}(\epsilon^{-2})$ iterations that each cost $\mathcal{O}(m)$.
- Plugging through the inequality again, we find that to find a point $\|\nabla J(\theta^{(t)})\| < \epsilon$ with high probability, stochastic gradient descent requires $\mathcal{O}(\epsilon^{-4})$ iterations that each cost $\mathcal{O}(1)$.
- So, in a very handwavy way, if $m \gg \epsilon^{-2}$, we're much better off using stochastic gradient descent.

Variants of (S)GD that are common in ML

(S)GD is just the beginning of the story. You should be aware of variants.

- Momentum-based methods.
  - Polyak momentum (Polyak, 1964).
  - Nesterov accelerated gradient method (Nesterov, 1983)
- Adaptive gradient methods.
  - AdaGrad (Duchi et al., 2011)
  - Adam (Kingma and Ba, 2014) $\leftarrow$ most popular for deep learning

If you want to think of all of these as "just (S)GD with some tweaks", that's probably fine for this course, but the details get quite interesting.

Variance reduction and control variates

- The $\mathbb{E}[\|g(\theta,\xi)\|^2] \leq \sigma^2$ is a bit brutal, but it can be loosened (Bottou et al., 2018), and you should think of this as a variance bound.
- There are a few ways to reduce the variance of g, but one of the most popular is the use of control variates. The idea is to use

$$g'(\theta,\xi) = g(\theta,\xi) + c \cdot h(\theta)$$

  instead of g for $c > 0$ and some random $h(\theta)$ such that $\mathbb{E}[h(\theta)] = 0$.
- For one-dimensional g and h, the optimal choice c^* of c that minimizes the variance of $g + ch$ gives an overall variance of

$$\mathrm{Var}\left(g(\theta,\xi) + c^* h(\theta)\right) = \mathrm{Var}(g(\theta,\xi)) - \frac{\mathrm{Cov}(g(\theta,\xi), h(\theta))^2}{\mathrm{Var}(h(\theta))}$$

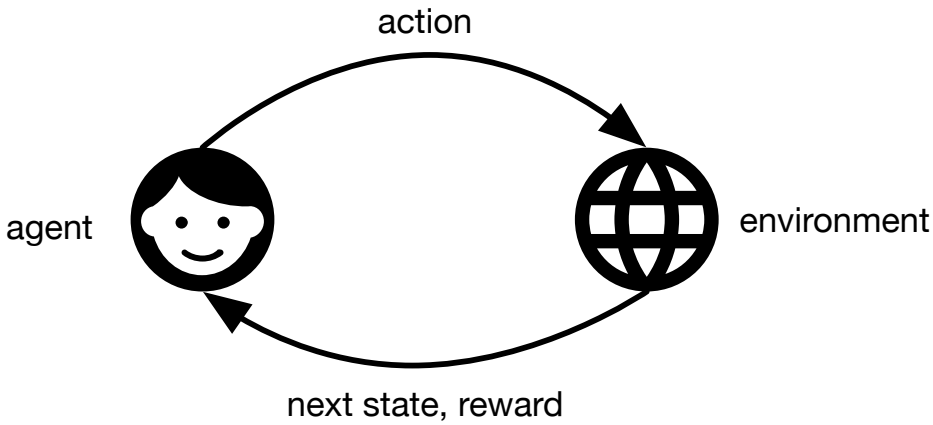
- So you want correlated h with expectation 0. Will return to this!

Next two weeks—gradient estimation

- Over the next two weeks, we will cover gradient estimation in great detail.
- The themes that we will consider:
  - Gradient estimation for different families $\mathcal{Q}$.
  - More elaborate control variates.
  - Gradient estimators for higher-order derivatives.
  - Etc.

Dynamic programming

Dynamic programming



- We motivated SGD for RL, but is it good enough?
  - often the variance of the gradient estimator is very large and convergence is prohibitively slow,
  - it is sometimes prone to convergence to bad local optima (deterministic, but suboptimal policies).
- SGD does not make much use of essential structure in RL. We will now consider dynamic programming ideas for reinforcement learning that exploit the additive structure of the return in RL.

Value Functions

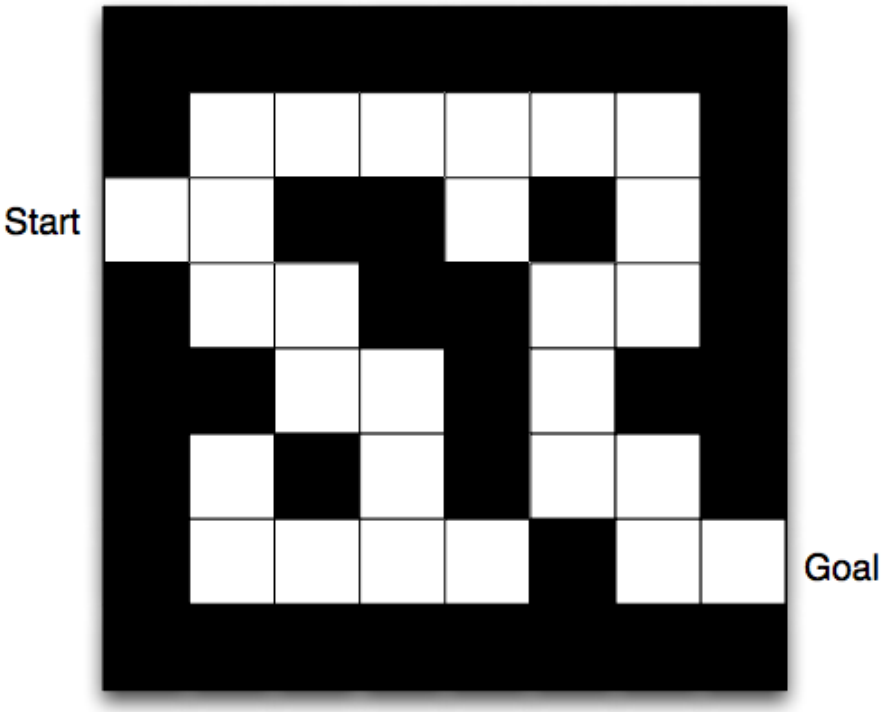
- Let's consider the infinite horizon setting.
- The value function V^π for a policy π measures the expected return if you start in state s and follow policy $\pi(a|s)$.

$$V^\pi(s) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s \right].$$

- V^π measures the desirability of the state s.
- Our ultimately goal is

$$\arg\max_\pi J(\pi) := \arg\max_\pi \mathbb{E}_{s_0 \sim p}[V^\pi(s_0)]$$

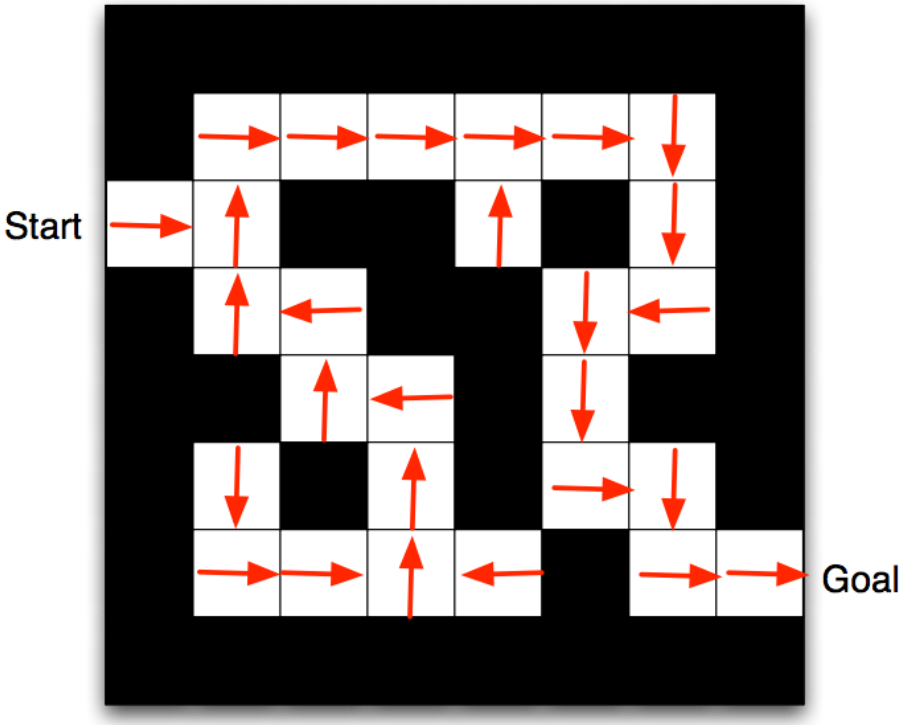
Value Function



- Rewards: -1 per time-step and $\gamma = 1$
- Actions: N, E, S, W
- States: Agent's location
- Goal is a terminal state

[Slide credit: D. Silver]

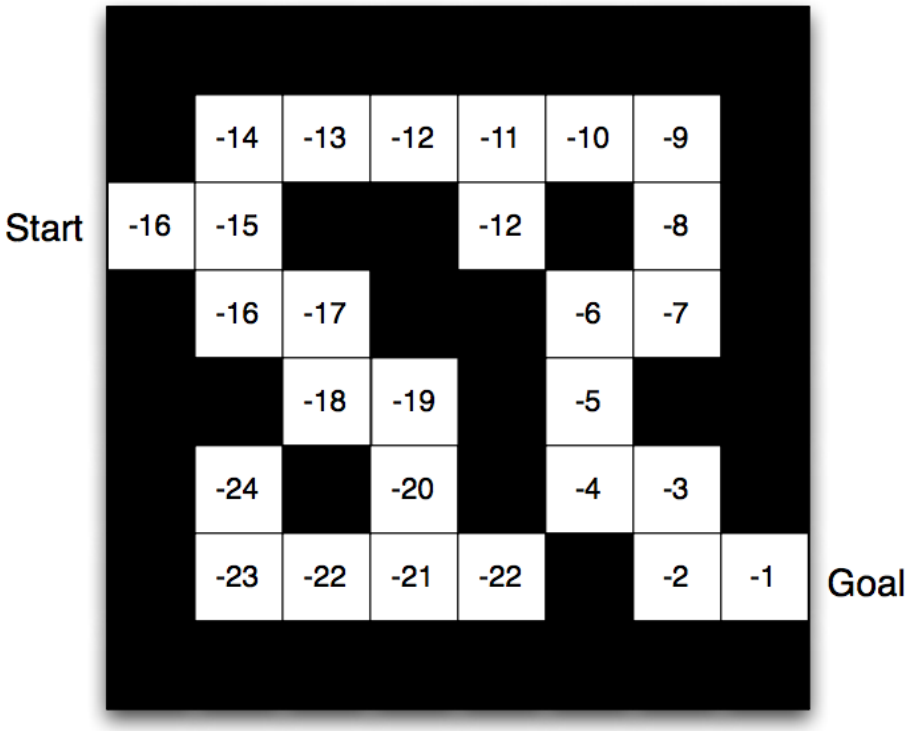
Value Function



- Arrows represent a deterministic policy $\pi(s)$ for each state s

[Slide credit: D. Silver]

Value Function



- Numbers represent value $V^{\pi}(s)$ of each state s

[Slide credit: D. Silver]

Bellman equations

- The foundation of many RL algorithms is the fact that value functions satisfy a recursive relationship, called the Bellman equation:

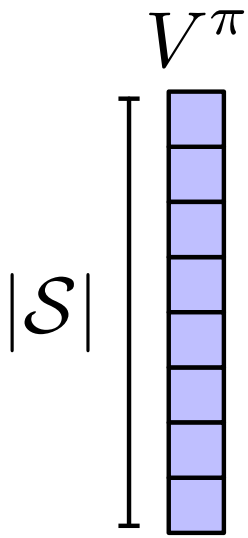
$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}[r(s, a_0) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \,|\, s_0 = s] \\
&= \sum_{a} \pi(a \,|\, s) \left[r(s, a) + \gamma \sum_{s'} p(s' \,|\, a, s) \, \mathbb{E}_{\pi}[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \,|\, s_1 = s'] \right] \\
&= \sum_{a} \pi(a \,|\, s) \left[r(s, a) + \gamma \sum_{s'} p(s' \,|\, a, s) \, V^{\pi}(s') \right]
\end{aligned}
$$

Bellman equations

- Viewing V^π as a vector (where entries correspond to states), define the Bellman backup operator T^π.

$$(T^\pi V)(s) :=$$

$$\sum_a \pi(a \mid s) \left[r(s,a) + \gamma \sum_{s'} p(s' \mid a, s)\, V(s') \right]$$

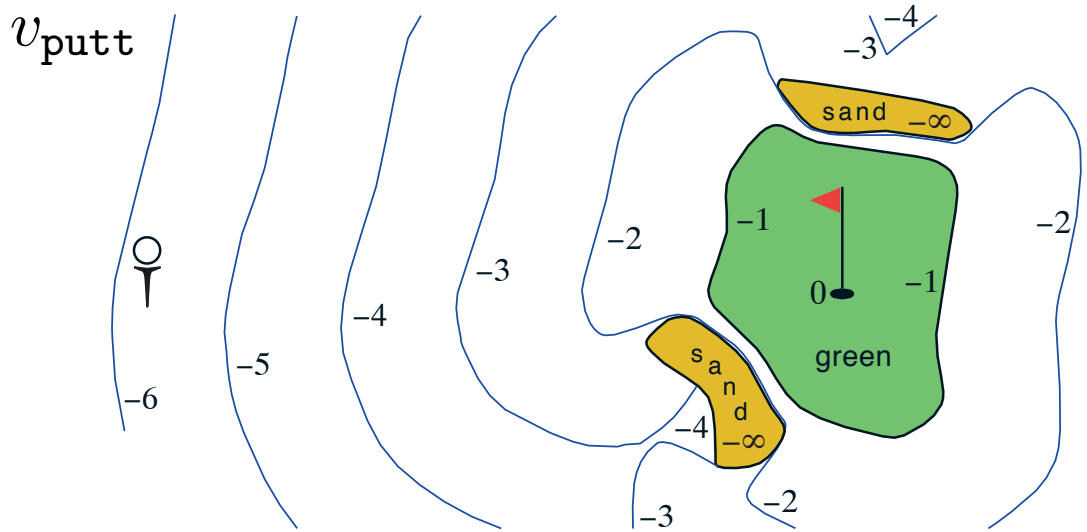


- The Bellman equation states that the value function V^π is fixed point of the Bellman operator:

$$T^\pi V^\pi = V^\pi.$$

Value Function

A value function for golf:

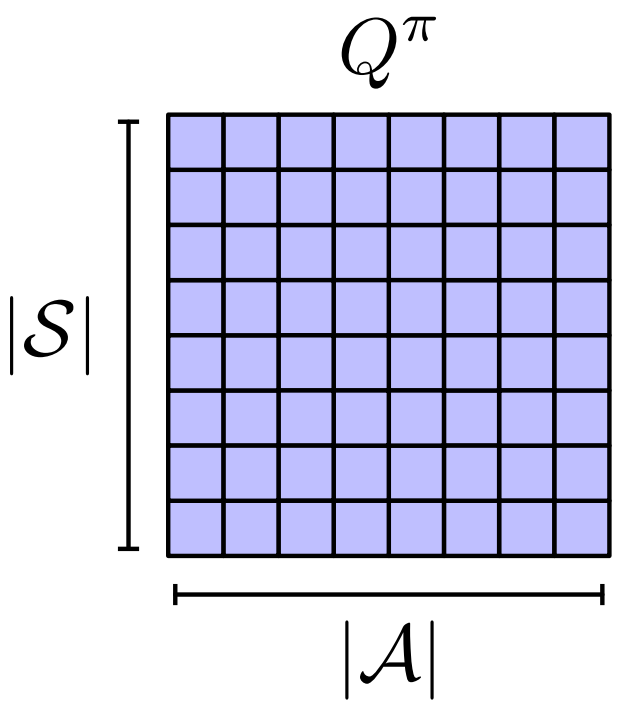


— Sutton and Barto, *Reinforcement Learning: An Introduction*

State-Action Value Function

A closely related but usefully different function is the state-action value function, or Q-function, Q^π for policy π, defined as:

$$Q^\pi(s,a) := \mathbb{E}_\pi \left[r(s,a) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \,|\, s_0 = s, a_0 = a \right].$$



State-Action Value Function

- If you knew Q^π, how would you obtain V^π?

$$V^\pi(s) = \sum_a \pi(a \mid s)\, Q^\pi(s, a).$$

- If you knew V^π, how would you obtain Q^π?
  - Apply a Bellman-like equation:

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} p(s' \mid a, s)\, V^\pi(s')$$

  - This requires knowing the dynamics, so in general it's not easy to recover Q^π from V^π.
- Q^π satisfies a Bellman equation

$$Q^\pi(s, a) = \underbrace{r(s, a) + \gamma \sum_{s'} p(s' \mid a, s) \sum_{a'} \pi(a' \mid s') Q^\pi(s', a')}_{:= (T^\pi Q^\pi)(s,a)}$$

Value Iteration

Optimal State-Action Value Function

- A remarkable feature of MDPs is that that exists a deterministic, stationary policy π that is optimal w.r.t. $J(\pi)$ (even when considering π possibly non-stationary!) (e.g., Thm 1.7 of Agarwal et al., 2020).
- If a deterministic policy π^* is optimal, then it must be the case that for any state s:

$$\pi^*(s) = \arg\max_a Q^{\pi^*}(s, a),$$

otherwise you could improve the policy by changing $\pi(s)$ (see Sutton & Barto for a proper proof). Hence

$$\begin{aligned} Q^{\pi^*}(s, a) &= r(s, a) + \gamma \sum_{s'} p(s', \mid s, a) Q^{\pi^*}(s', \pi^\star(s')) \\ &= r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) \max_{a'} Q^{\pi^\star}(s', a') \end{aligned}$$

Optimal State-Action Value Function

- Q^* is an optimal state-action value function if

$$Q^*(s,a) = \underbrace{r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q^*(s',a')}_{\triangleq (T^* Q^*)(s,a)}$$

  Note this is satisfied by Q^{π^*}

- Turns out this is *sufficient* (e.g., Thm 1.8 of Agarwal et al., 2020) to characterize the optimal policy. So we simply need to solve the fixed point equation $T^* Q^* = Q^*$, and then we can choose $\pi^*(s) = \arg\max_a Q^*(s,a)$.

Bellman Fixed Points

- So far: showed that some interesting problems could be reduced to finding fixed points of Bellman backup operators:
  - Evaluating a fixed policy π

$$T^{\pi} Q^{\pi} = Q^{\pi}$$

  - Finding the optimal policy

$$T^{*} Q^{*} = Q^{*}$$

Bellman Fixed Points

- So far: showed that some interesting problems could be reduced to finding fixed points of Bellman backup operators:
  - Evaluating a fixed policy π

$$T^{\pi} Q^{\pi} = Q^{\pi}$$

  - Finding the optimal policy

$$T^{*} Q^{*} = Q^{*}$$

- Idea: keep iterating the backup operator over and over again.

$$Q \leftarrow T^{\pi} Q \qquad \text{(policy evaluation)}$$

$$Q \leftarrow T^{*} Q \qquad \text{(value iteration)}$$

  - We're treating Q^{π} or Q^{*} as a vector with $|\mathcal{S}| \cdot |\mathcal{A}|$ entries.
  - This type of algorithm is an instance of dynamic programming.

Bellman Fixed Points

- An operator f (mapping from vectors to vectors) is a contraction map if

$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| \leq \alpha \|\mathbf{x}_1 - \mathbf{x}_2\|$$

for some scalar $0 \leq \alpha < 1$ and vector norm $\|\cdot\|$.

Bellman Fixed Points

- An operator f (mapping from vectors to vectors) is a contraction map if

$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| \leq \alpha \|\mathbf{x}_1 - \mathbf{x}_2\|$$

  for some scalar $0 \leq \alpha < 1$ and vector norm $\|\cdot\|$.
- Let $f^{(k)}$ denote f iterated k times. A simple induction shows

$$\|f^{(k)}(\mathbf{x}_1) - f^{(k)}(\mathbf{x}_2)\| \leq \alpha^k \|\mathbf{x}_1 - \mathbf{x}_2\|.$$

Bellman Fixed Points

- An operator f (mapping from vectors to vectors) is a contraction map if
$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| \leq \alpha \|\mathbf{x}_1 - \mathbf{x}_2\|$$
for some scalar $0 \leq \alpha < 1$ and vector norm $\|\cdot\|$.
- Let $f^{(k)}$ denote f iterated k times. A simple induction shows
$$\|f^{(k)}(\mathbf{x}_1) - f^{(k)}(\mathbf{x}_2)\| \leq \alpha^k \|\mathbf{x}_1 - \mathbf{x}_2\|.$$
- Let $\mathbf{x}^*$ be a fixed point of f. Then for any $\mathbf{x}$,
$$\|f^{(k)}(\mathbf{x}) - \mathbf{x}^*\| \leq \alpha^k \|\mathbf{x} - \mathbf{x}_*\|.$$
- Hence, iterated application of f, starting from any $\mathbf{x}$, converges exponentially to a unique fixed point.

Finding the Optimal Value Function: Value Iteration

- Let's use dynamic programming to find Q^*.
- Value Iteration: Start from an initial function Q_1. For each $k = 1, 2, \ldots,$ apply

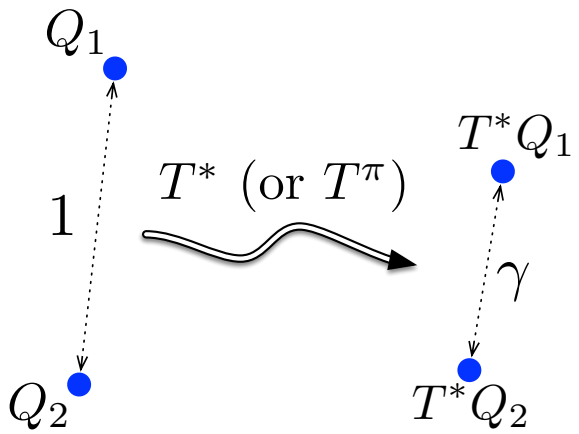
$$Q_{k+1} \leftarrow T^* Q_k$$

- Writing out the update in full,

$$Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \max_{a' \in \mathcal{A}} Q_k(s', a')$$

- Observe: a fixed point of this update is exactly a solution of the optimal Bellman equation, which we saw characterizes the Q-function of an optimal policy.

Value Iteration



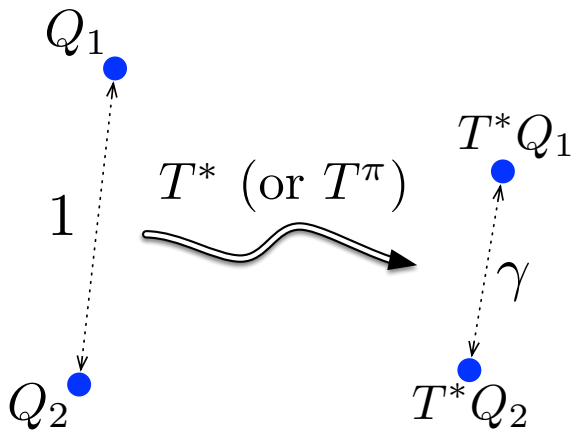
- Claim: The value iteration update is a contraction map:

$$\|T^*Q_1 - T^*Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$$

- $\|\cdot\|_\infty$ denotes the L^∞ norm, defined as:

$$\|\mathbf{x}\|_\infty = \max_i |x_i|$$

Value Iteration



- Claim: The value iteration update is a contraction map:

$$\|T^*Q_1 - T^*Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$$

- $\|\cdot\|_\infty$ denotes the L^∞ norm, defined as:

$$\|\mathbf{x}\|_\infty = \max_i |x_i|$$

- If this claim is correct, then value iteration converges exponentially to the unique fixed point.
- The exponential decay factor is γ (the discount factor), which means longer term planning is harder.

Bellman Operator is a Contraction

$$
\begin{aligned}
|(T^* Q_1)(s,a) - (T^* Q_2)(s,a)| &= \gamma \left| \sum_{s'} p(s' \mid s,a) \left[\max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a') \right] \right| \\
&\leq \gamma \sum_{s'} p(s' \mid s,a) \left| \max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a') \right| \\
&\leq \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} \left| Q_1(s',a') - Q_2(s',a') \right| \\
&= \gamma \max_{s',a'} \left| Q_1(s',a') - Q_2(s',a') \right| \\
&= \gamma \left\| Q_1 - Q_2 \right\|_\infty
\end{aligned}
$$

- This is true for *any* (s,a), so

$$\left\| T^* Q_1 - T^* Q_2 \right\|_\infty \leq \gamma \left\| Q_1 - Q_2 \right\|_\infty ,$$

From which we get that for the iterates Q_k of value iteration:

$$\left\| T^* Q_{k+1} - Q^* \right\|_\infty \leq \gamma \left\| T^* Q_k - Q^* \right\|_\infty ,$$

Policy iteration

Policy evaluation

For $k = 1, \ldots$

$$Q_{k+1} \leftarrow T^{\pi} Q_k$$

converges to Q^{π}

Value iteration

For $k = 1, \ldots$

$$Q_{k+1} \leftarrow T^{*} Q_k$$

converges to Q^{*}

- Both iterations are contractions.
- They seem different, but we can actually re-write value iteration using a policy evaluation update.

Policy iteration

- Define greedy policy with respect to Q:

$$\pi_Q(s) \in \arg\max_a Q(s,a).$$

- With this notation, we can re-write the operator T^* using T^π

$$Q \leftarrow T^* Q \qquad \Longleftrightarrow \qquad \begin{array}{l} \pi \leftarrow \pi_Q \\ Q \leftarrow T^\pi Q \end{array}$$

- This suggests an algorithm that runs the policy evaluation $Q_{k+1} \leftarrow T^{\pi_k} Q_k$ to convergence in an inner loop. This is the generalization known as policy iteration.

Policy iteration

Value iteration

For $k = 1, \ldots$

$$\pi_{k+1} \leftarrow \pi_{Q_k}$$
$$Q_{k+1} \leftarrow T^{\pi_{k+1}} Q_k$$

Policy iteration

For $k = 1, \ldots$

$$\pi_{k+1} \leftarrow \pi_{Q_k}$$
$$Q_{k+1} \leftarrow Q^{\pi_{k+1}}$$

- Recall that Q^π is the Q function corresponding to policy π, so the second line of policy iteration is equivalent to running policy evaluation to convergence with policy π_{Q_k}.
- Policy iteration has more expensive iterations, but it sometimes converges in fewer iterations.

Policy iteration—convergence

- Policy iteration's convergence is guaranteed by the following facts:
  1. $Q^{\pi_{k+1}} \geq T^* Q^{\pi_k} \geq Q^{\pi_k}$
  2. $\|Q^{\pi_{k+1}} - Q^*\|_\infty \leq \gamma \|Q^{\pi_k} - Q^*\|_\infty$
- Sketch (see, e.g., Lemma 1.13 in Agarwal et al., 2020)
  1. $T^* Q^{\pi_k} \geq Q^{\pi_k}$, because T^* is a greedy improvement of Q^{π_k} for each (s, a), i.e., maximize Q^{π_k} over the next action.
  2. $Q^{\pi_{k+1}} \geq Q^{\pi_k}$. Q^{π_k} gives us the value from (s, a) assuming we will continue playing with π_k. π_{Q_k} locally optimizes Q^{π_k} at each iteration. Induction over times gives us our result.
  3. $Q^{\pi_{k+1}} \geq T^* Q^{\pi_k}$, follows from 1. and 2. You can think of it like, take one greedy step using π_{Q_k} and then follow π_k.
  4. Finally, using these results and the fact that $Q^* \geq Q$ for all Q:

$$\begin{aligned}\|Q^* - Q^{\pi_{k+1}}\|_\infty \leq \|Q^* - T^* Q^{\pi_k}\|_\infty &= \|T^* Q^* - T^* Q^{\pi_k}\|_\infty \\ &\leq \gamma \|Q^* - Q^{\pi_k}\|_\infty\end{aligned}$$

- Policy iteration is no worse than value iteration. One iteration is typically more expensive, but overall can have better complexity.

Recap

We looked at

- Gradient descent and stochastic gradient descent.
  - Assumed a (differentiable) parametric family for the policy.
  - Assumed that we could simulate $X \sim q$.
  - Might converge to a local optima.
- Dynamic programming (value iteration and policy iteration).
  - Assumed that we have access to the transition distribution mass function that $|\mathcal{S}|$ and $|\mathcal{A}|$ are small. An instance of planning.
  - Can converge quickly to global solution.

Recap

- What are the limitations of dynamic programming?
  - requires explicitly representing Q^* as a vector
    - $|\mathcal{S}|$ can be extremely large, or infinite
    - $|\mathcal{A}|$ can be infinite (e.g. continuous voltages in robotics)
- But value iteration is still a foundation for a lot of more practical RL algorithms.
  - Most approaches soften the requirements.
  - Can we implement under the same assumptions as SGD?
  - We could simulate from the policy and the environment, can we still perform value iteration? policy iteration?
  - Stochastic variants harder to understand, but well-studied.

Classifying agents

More broadly, the central theme in this course will be how different methods are appropriate under different assumptions on the family of policies $\mathcal{Q}$, the structure of the function $f(X, q)$, and the access we assume to other sources of randomness. Roughly speaking:

- Model-based: in RL, we assume full access to the environment's transition dynamics. Sometimes we learn this.
- Model-free: in RL, we assume the ability to simulate an interaction, but otherwise we do not have access to the transition dynamics.
- Implicit distribution: we do not assume access to the densities of $q \in \mathcal{Q}$, but we assume the ability to simulate from them.
- On-policy: We use $X \sim q$ to improve our policy q.
- Off-policy: We use some other random variable Y taking values in $\mathcal{X}$ to improve our policy q.

Some things to ponder

- Can we use value iteration or policy iteration for problems in probabilistic inference (e.g. latent time series models)?
- Can we exploit the temporal structure of the return in RL to implement better SGD methods?