TÉCNICO LISBOA

Planning, Learning and Intelligent Decision Making

Lecture 11

PADInt 2024

Computing J^π

Monte Carlo policy evaluation

- Given a trajectory obtained with policy π

$$\tau = \{x_t, c_t, x_{t+1}, c_{t+1}, \dots, c_{t+N}, x_{t+N+1}\}$$

- Compute loss

$$L(\tau) = \sum_{n=0}^{N} \gamma^n c_{t+n}$$

- Update

We want $J_t(x_t)$ to approach $L(\tau)$

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t(L(\tau) - J_t(x_t))$$

Computing J^π

N-step TD learning

- Given a sample $(x_t, c_t, c_{t+1}, \ldots, c_{t+N}, x_{t+N+1})$ obtained with policy π
- Compute

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t \left[\sum_{n=0}^{N} \gamma^n c_{t+n} + \gamma^{N+1} J_t(x_{t+N+1}) - J_t(x_t) \right]$$

We want $J_t(x_t)$ to approach $\sum_n \gamma^n c_{t+n} + \gamma^{N+1} J_t(x_{t+N+1})$

Computing J^π

TD(λ)

- Given a sample (x_t, c_t, x_{t+1}), where the action was selected from π
- Compute

$$z_{t+1}(x) = \lambda\gamma z_t(x) + \mathbb{I}(x = x_t)$$

$$J_{t+1}(x) = J_t(x) + \alpha_t z_{t+1}(x)[c_t + \gamma J_t(x_{t+1}) - J_t(x_t)]$$

We want $J_t(x_{t+n})$ to approach $c_{t+n} + \gamma J_t(x_{t+n+1})$

Computing Q^π

Monte Carlo policy evaluation

- Given a trajectory obtained with policy π

$$\tau = \{x_t, a_t, c_t, x_{t+1}, a_{t+1}, c_{t+1}, \dots, a_{t+N}, c_{t+N}, x_{t+N+1}\}$$

- Compute loss

$$L(\tau) = \sum_{n=0}^{N} \gamma^n c_{t+n}$$

- Update

We want $Q_t(x_t, a_t)$ to approach $L(\tau)$

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t(L(\tau) - Q_t(x_t, a_t))$$

Computing Q^π

SARSA

- Given a sample $(x_t, a_t, c_t, x_{t+1}, a_{t+1})$,
- Compute

We want $Q_t(x_t, a_t)$ to approach $c_t + \gamma Q_t(x_{t+1}, a_{t+1})$

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t) \right]$$

Computing Q^*

Q-learning

- Given a sample (x_t, a_t, c_t, x_{t+1}),
- Compute

We want $Q_t(x_t, a_t)$ to approach $c_t + \gamma \min_{a'} Q_t(x_{t+1}, a')$

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha_t \left[c_t + \gamma \min_{a' \in \mathcal{A}} Q_t(x_{t+1}, a') - Q_t(x_t, a_t) \right]$$

TÉCNICO LISBOA

Value based RL: Episode II

Large problems

Large domains

- How do these methods work in large domains (e.g., Tetris, Gammon, Go)?

The curse of dimensionality

- We must resort to some form of approximation
- Instead of allowing arbitrary functions, methods restrict to pre-specified families of functions
  - Families provide good representations → methods perform well
  - Families provide bad representations → methods perform poorly

Another view of VB methods

Prediction (computing $\mathbf{J}^\pi$)

- We can no longer represent arbitrary cost-to-go functions
- Parametric family of functions $\{J_\theta, \theta \in \mathbb{R}^M\}$
- Ideally, we want to compute J such that

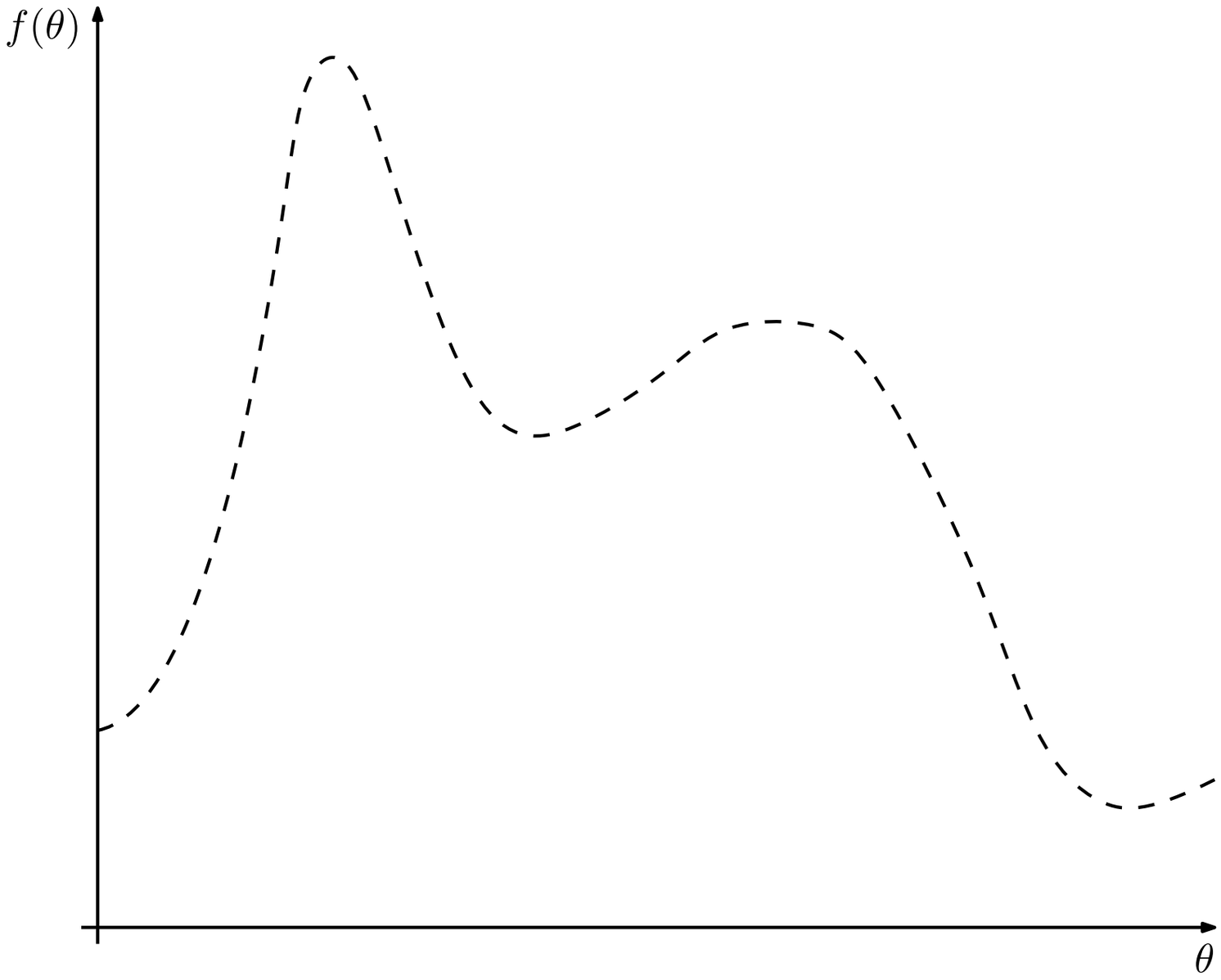
$$J = \arg\min_\theta \mathbb{E}_\mu \left[(J^\pi(\mathrm{x}) - J_\theta(\mathrm{x}))^2 \right]$$

Target (what we want to learn)

Gradient descent

Continuous settings

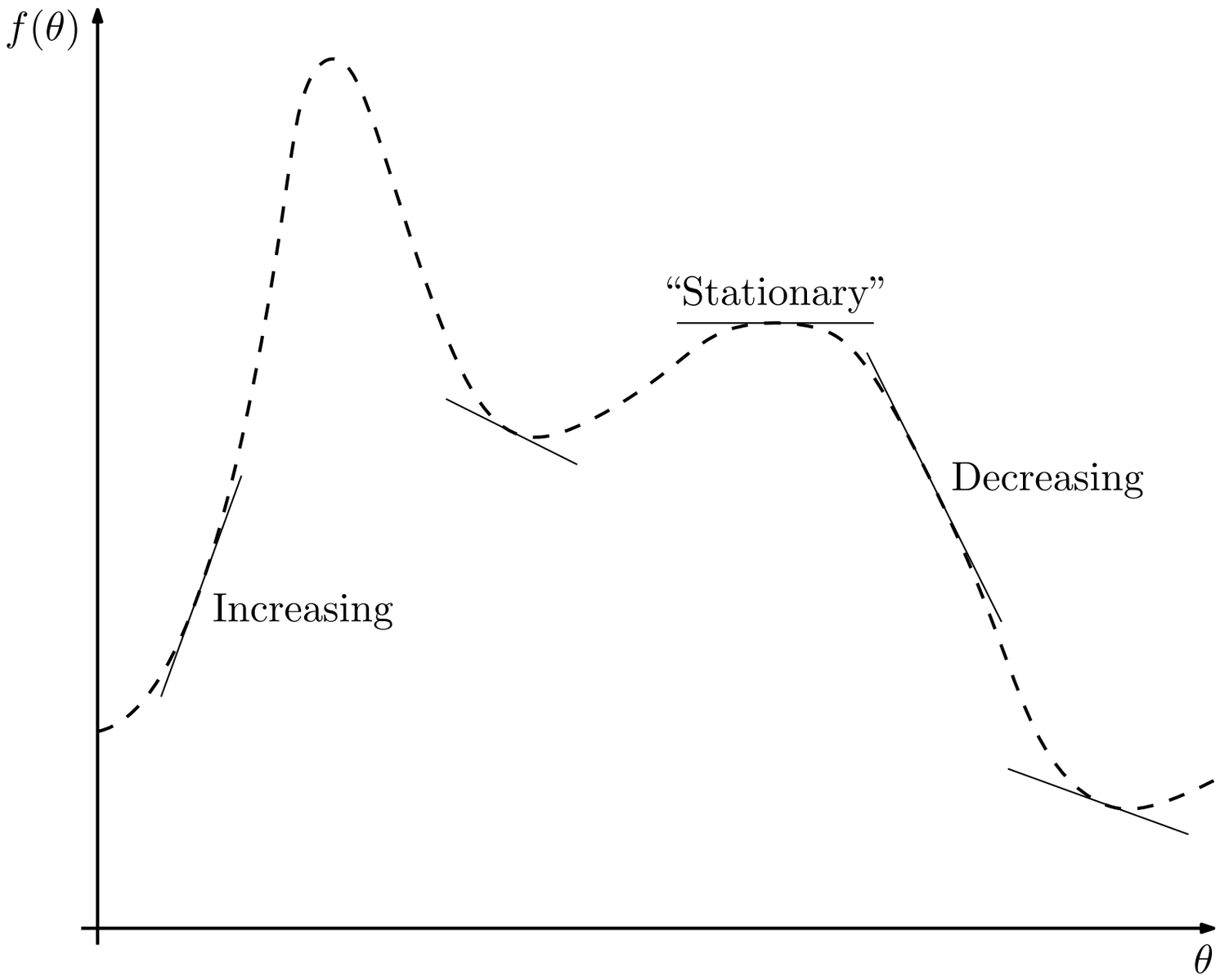
- How can we compute the minimum of a function f?



Derivatives

- Derivatives provide clues on the behavior of the function
  - A **positive derivative** means that the function is "increasing"
  - A **negative derivative** means that the function is "decreasing"
  - A **null derivative** means that the function is "stationary"

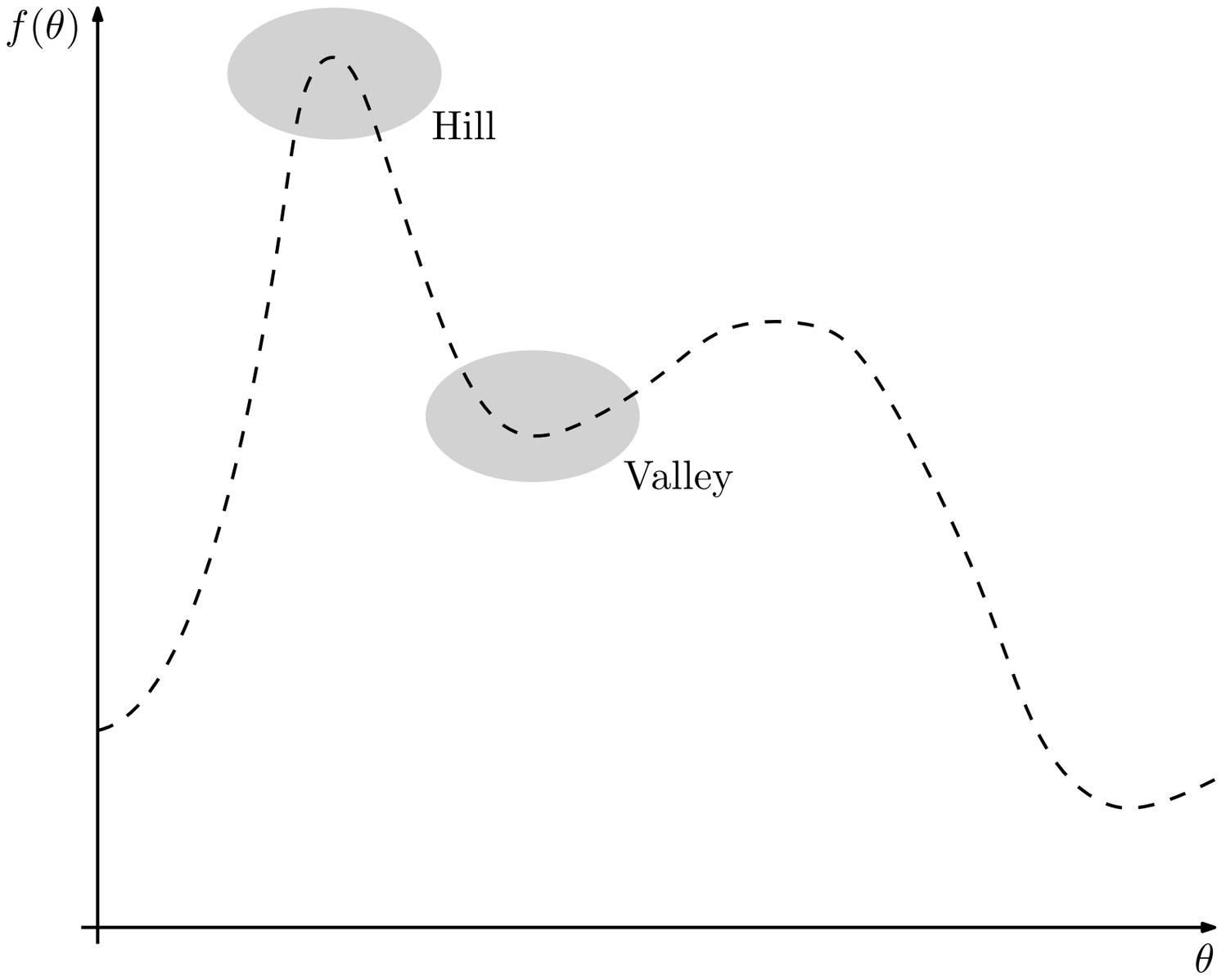
Derivatives



Derivatives

- Second derivatives also help
  - A **positive second derivative** means that we're in a "valley"
  - A **negative second derivative** means that we're in a "hill"

Derivatives



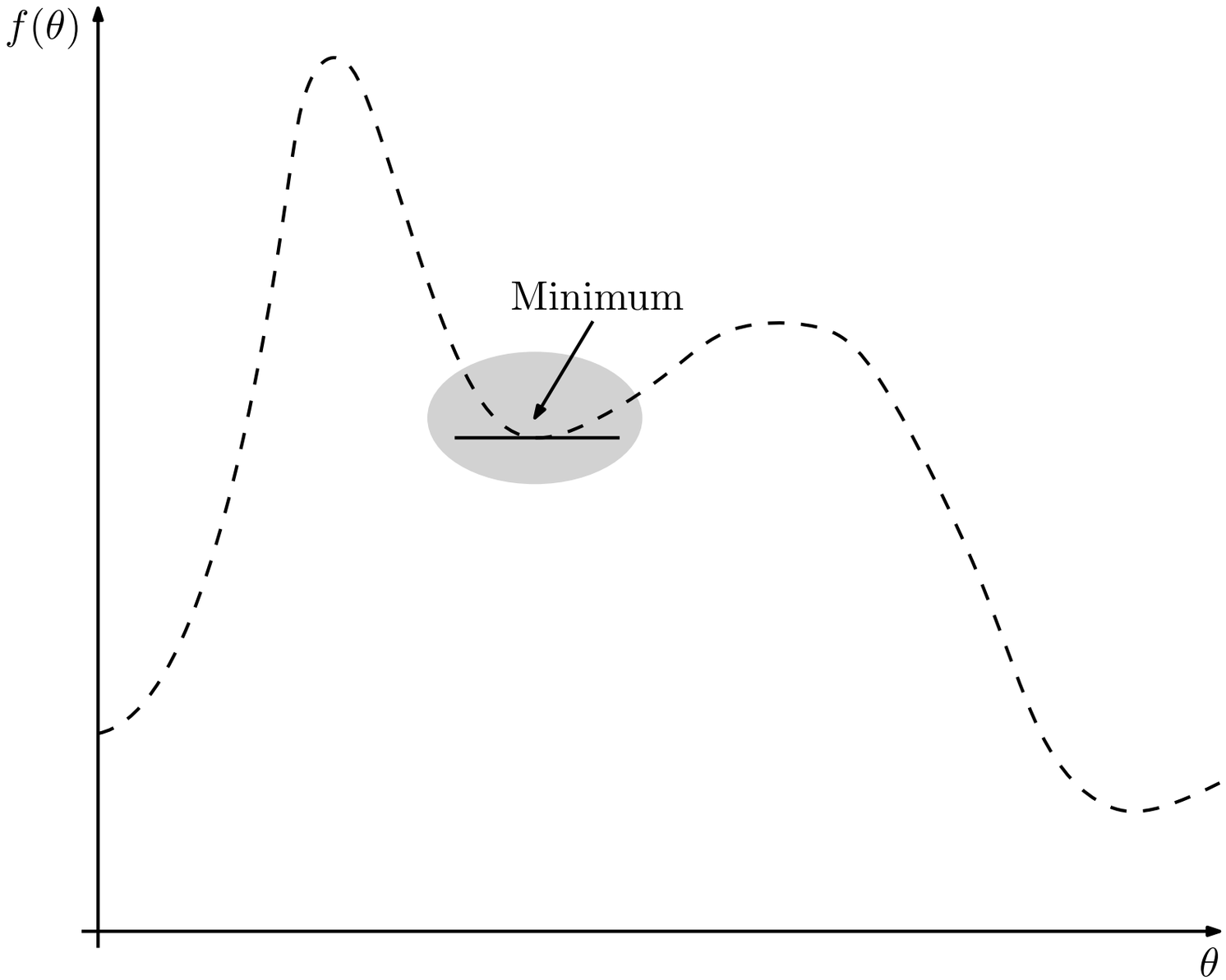
Putting everything together

- If we have:
  - **Null first derivative** (stationary function)
  - **Positive second derivative** ("valley")

... we must be at a minimum (local)!

Putting everything together

- We can use derivatives to determine if we found a solution

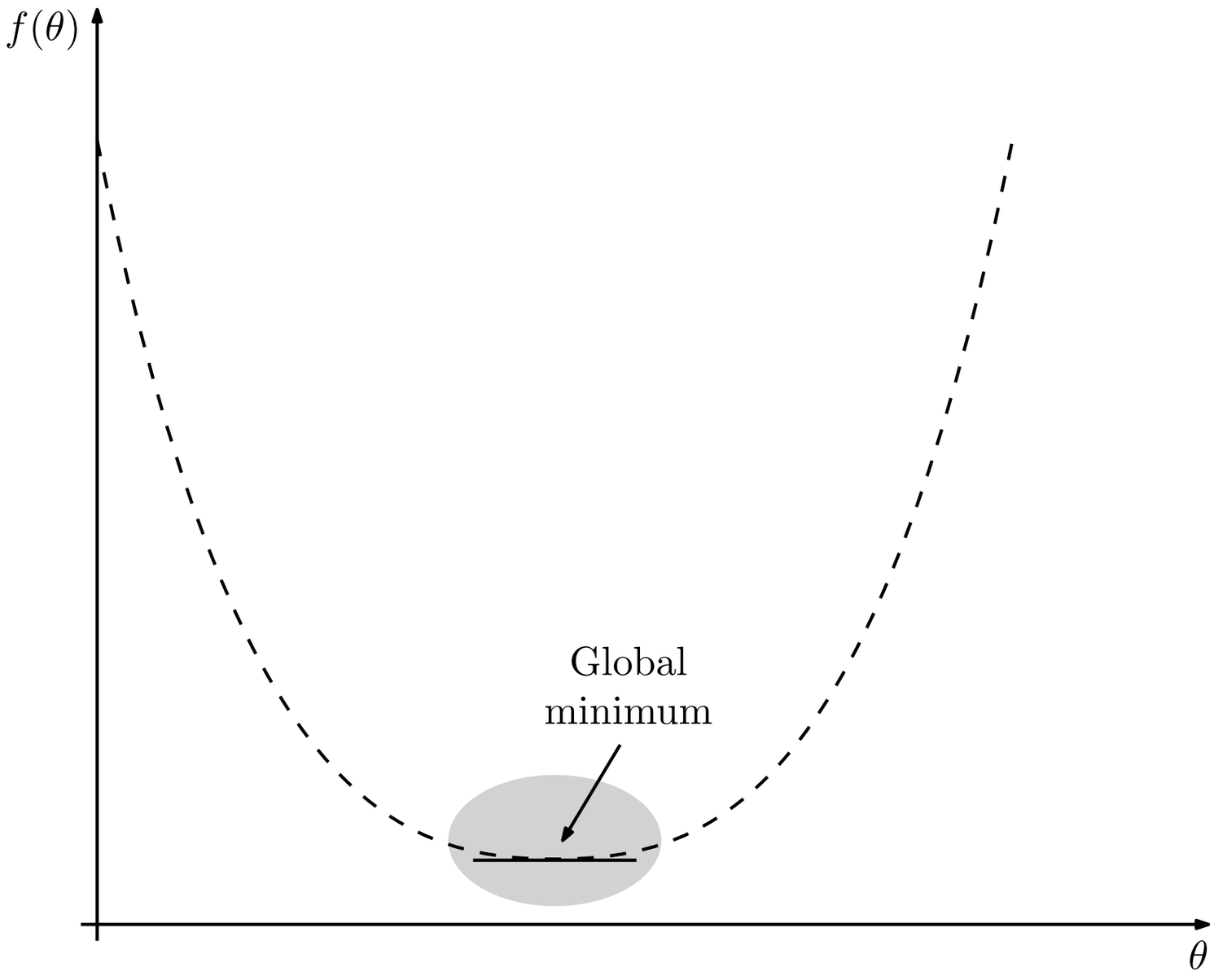


Conditions of optimality 1

- Then,
  - If the second derivative is **everywhere** non-negative ("valley" everywhere)
  - Null first derivative (stationary function)

... we must be at a global minimum!

Conditions of optimality 1



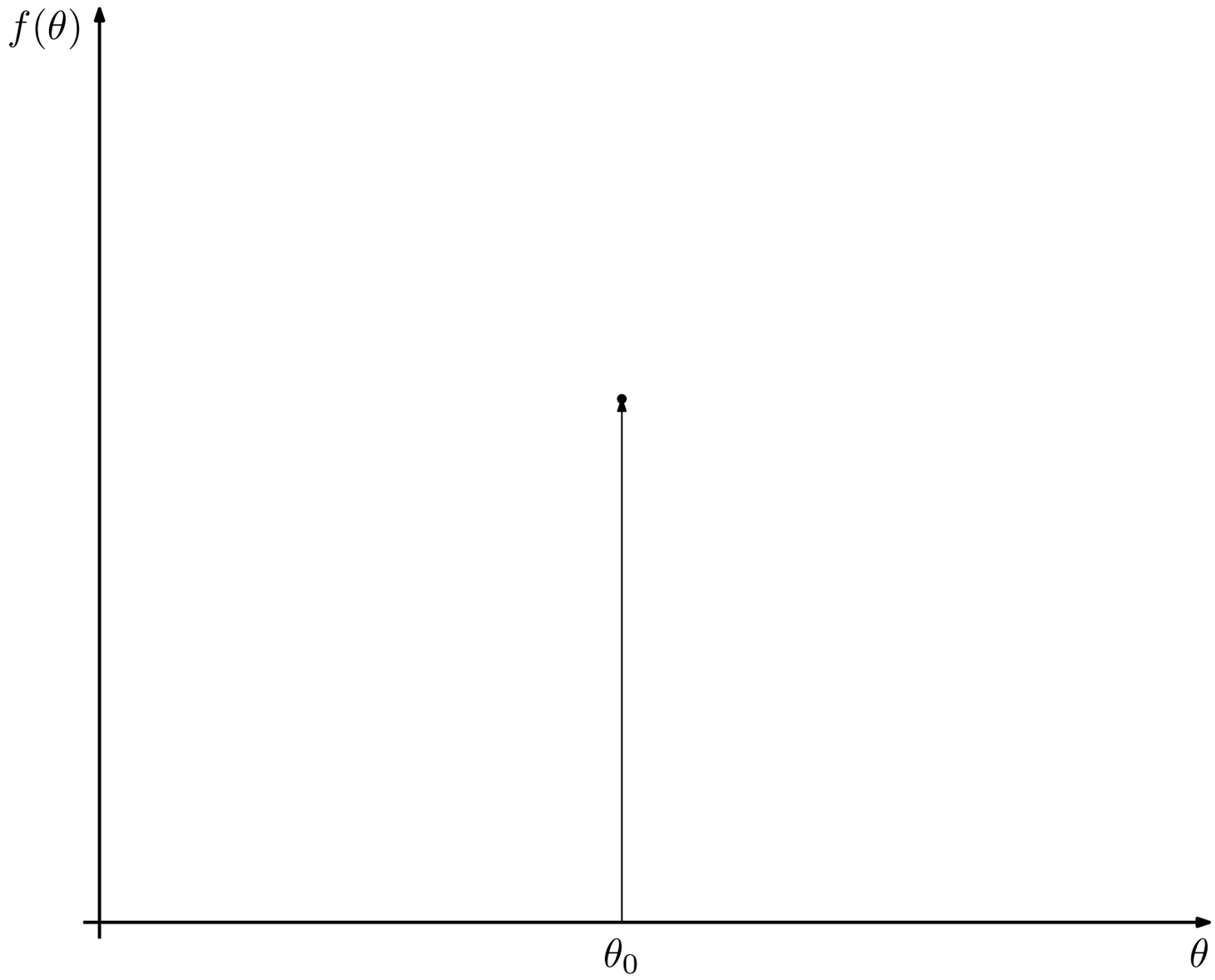
Iterative descent

- Optimality conditions are used to **identify** a minimum
- Can we use derivative information to **compute** a minimum?

Iterative descent

Idea:

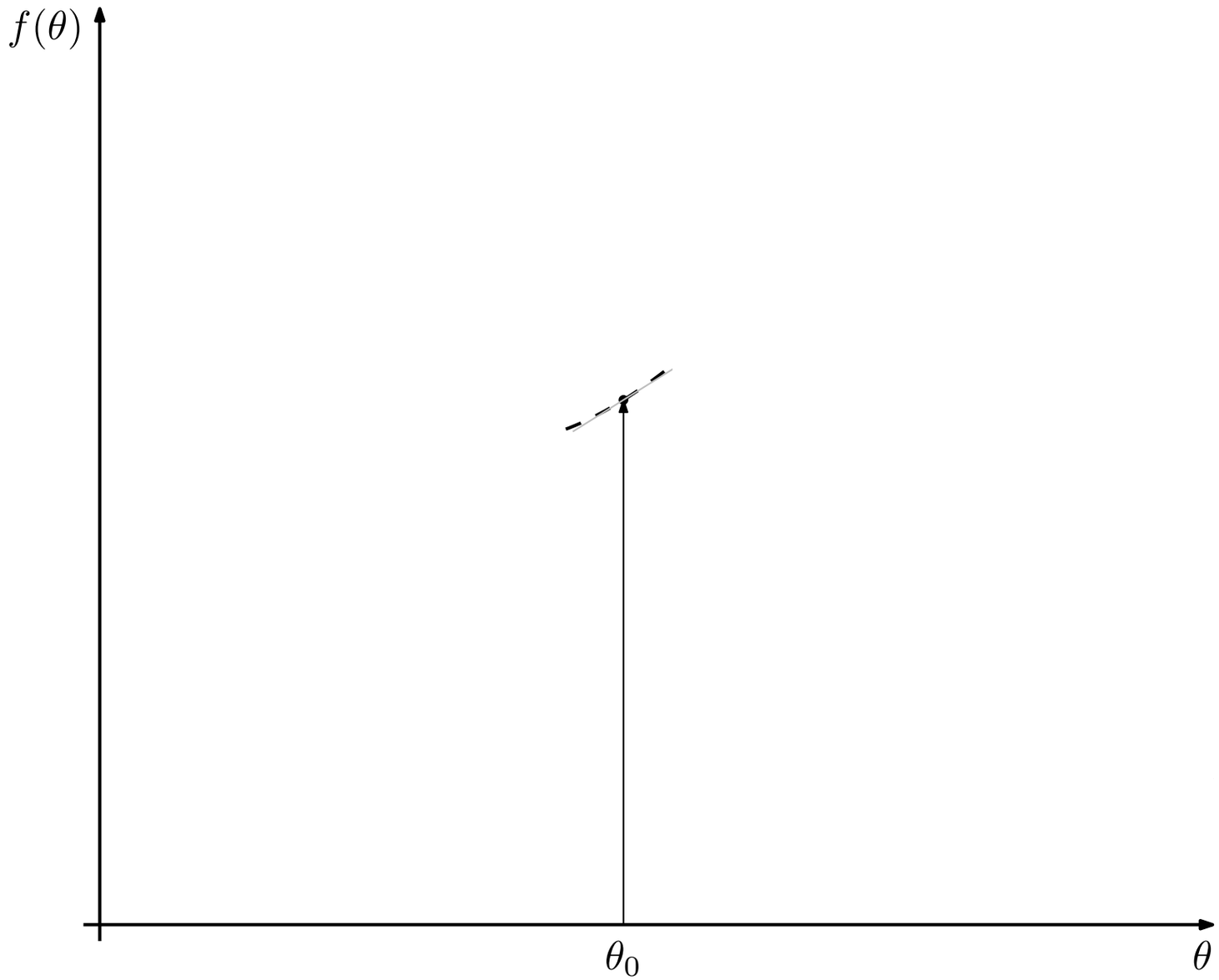
- Start somewhere



Iterative descent

Idea:

- Start somewhere
- Check derivative

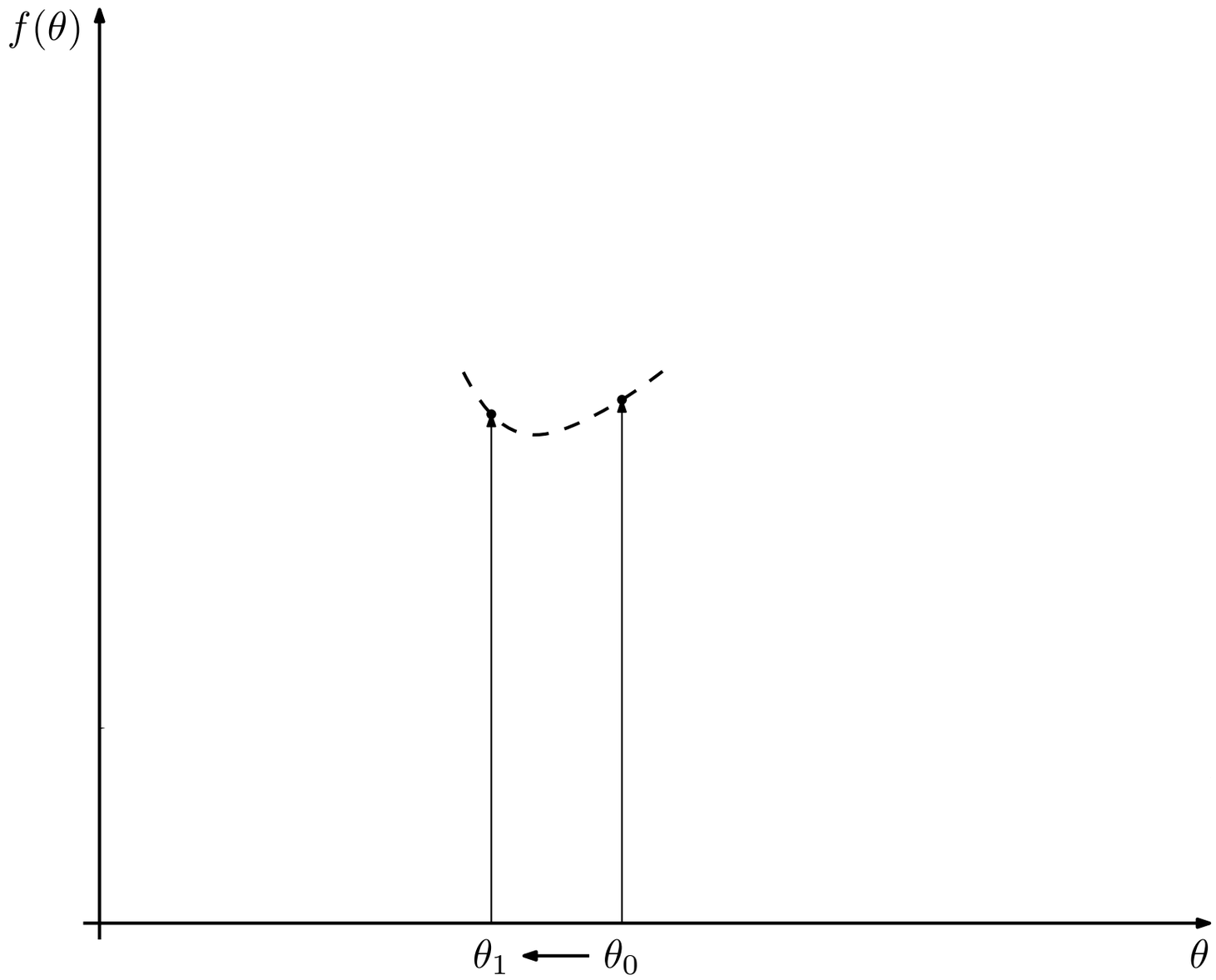


Iterative descent

Idea:

- Start somewhere
- If derivative > 0,

  move "back"



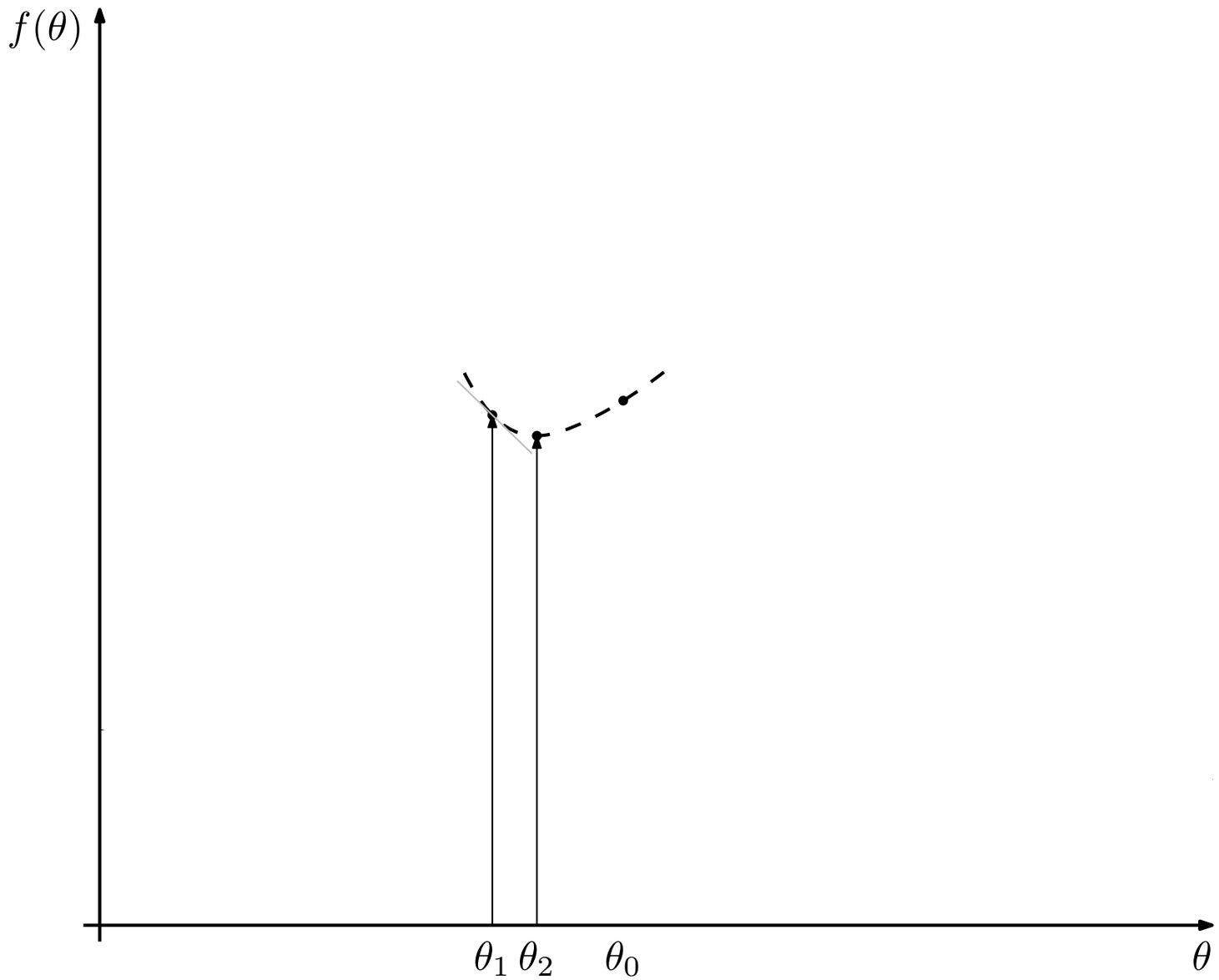
Iterative descent

Idea:

- Start somewhere
- If derivative > 0,

  move "back"
- If derivative < 0,

  move "forward"



Formally...

- Iterative update:

$$\theta_{t+1} = \theta_t - \alpha f'(\theta_t)$$

- Recall the Taylor series expansion:

$$f(\theta_{t+1}) \approx f(\theta_t) + \boxed{(\theta_{t+1} - \theta_t) f'(\theta_t)} + \text{small stuff}$$

Previous value

Small linear increment

Formally...

- Iterative update:

$$\theta_{t+1} = \theta_t - \alpha f'(\theta_t)$$

- Recall the Taylor series expansion:

$$f(\theta_{t+1}) \approx f(\theta_t) + (\theta_{t+1} - \theta_t) f'(\theta_t) + \text{small stuff}$$

$$-\alpha f'(\theta_t)$$

Formally...

- Iterative update:

$$\theta_{t+1} = \theta_t - \alpha f'(\theta_t)$$

- Recall the Taylor series expansion:

$$f(\theta_{t+1}) \approx \boxed{f(\theta_t) - \alpha(f'(\theta_t))^2} + \text{small stuff}$$

We indeed decrease!

What about many dimensions?

Idea is the same

- Derivatives still provide clues on the behavior of the function

Idea is the same

- Derivative $D_{\mathbf{u}} f(\theta)$ along a direction $\mathbf{u}$ can be computed as

$$D_{\mathbf{u}} f(\theta) = \langle \mathbf{u}, \boxed{\nabla_\theta f(\theta)} \rangle$$

Gradient of f at θ

$$\begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \vdots \\ \frac{\partial f}{\partial \theta_M} \end{bmatrix}$$

Idea is the same

- Derivative $D_{\mathbf{u}} f(\theta)$ along a direction $\mathbf{u}$ can be computed as

$$D_{\mathbf{u}} f(\theta) = \langle \mathbf{u}, \nabla_\theta f(\theta) \rangle$$

... and again, the Taylor expansion...

$$f(\theta + \alpha \mathbf{u}) \approx f(\theta) + \alpha D_{\mathbf{u}} f(\theta) + \text{small stuff}$$

Idea is the same

- Derivative $D_{\mathbf{u}} f(\theta)$ along a direction $\mathbf{u}$ can be computed as

$$D_{\mathbf{u}} f(\theta) = \langle \mathbf{u}, \nabla_\theta f(\theta) \rangle$$

... and again, the Taylor expansion...

$$f(\theta + \alpha \mathbf{u}) \approx f(\theta) + \alpha \langle \mathbf{u}, \nabla_\theta f(\theta) \rangle + \text{small stuff}$$

Original value

Small linear increment

Idea is the same

- But...

Angle between $\mathbf{u}$ and $\nabla_\theta f(\theta)$

$$\langle \mathbf{u}, \nabla_\theta f(\theta) \rangle = \|\mathbf{u}\| \|\nabla_\theta f(\theta)\| \cos \beta$$

- ... the largest increment occurs when the update direction is that of the gradient!

- ... the largest decrement occurs when the update direction is **opposite** to the gradient

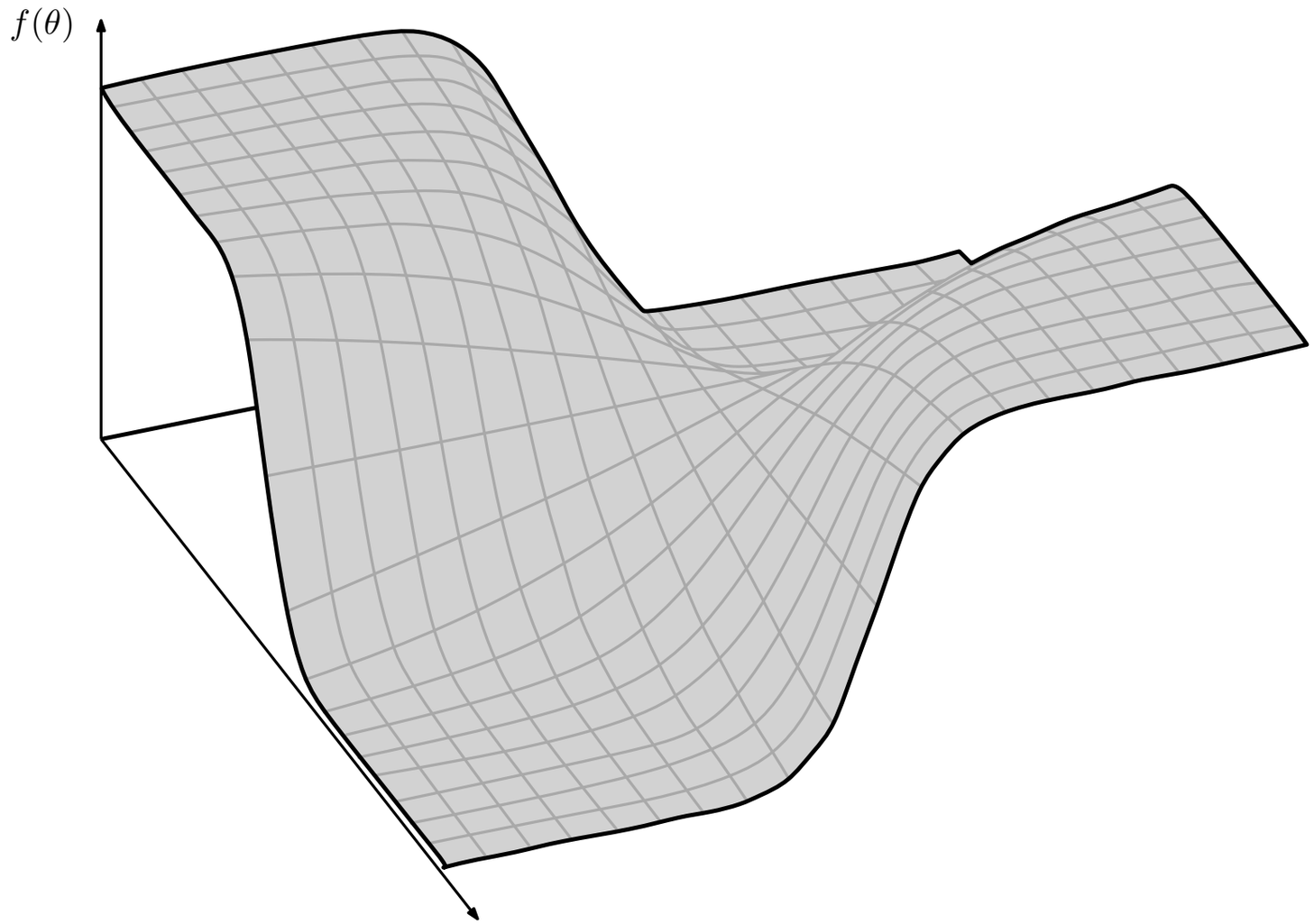
Gradient descent

- Iterative update:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta f(\theta_t)$$

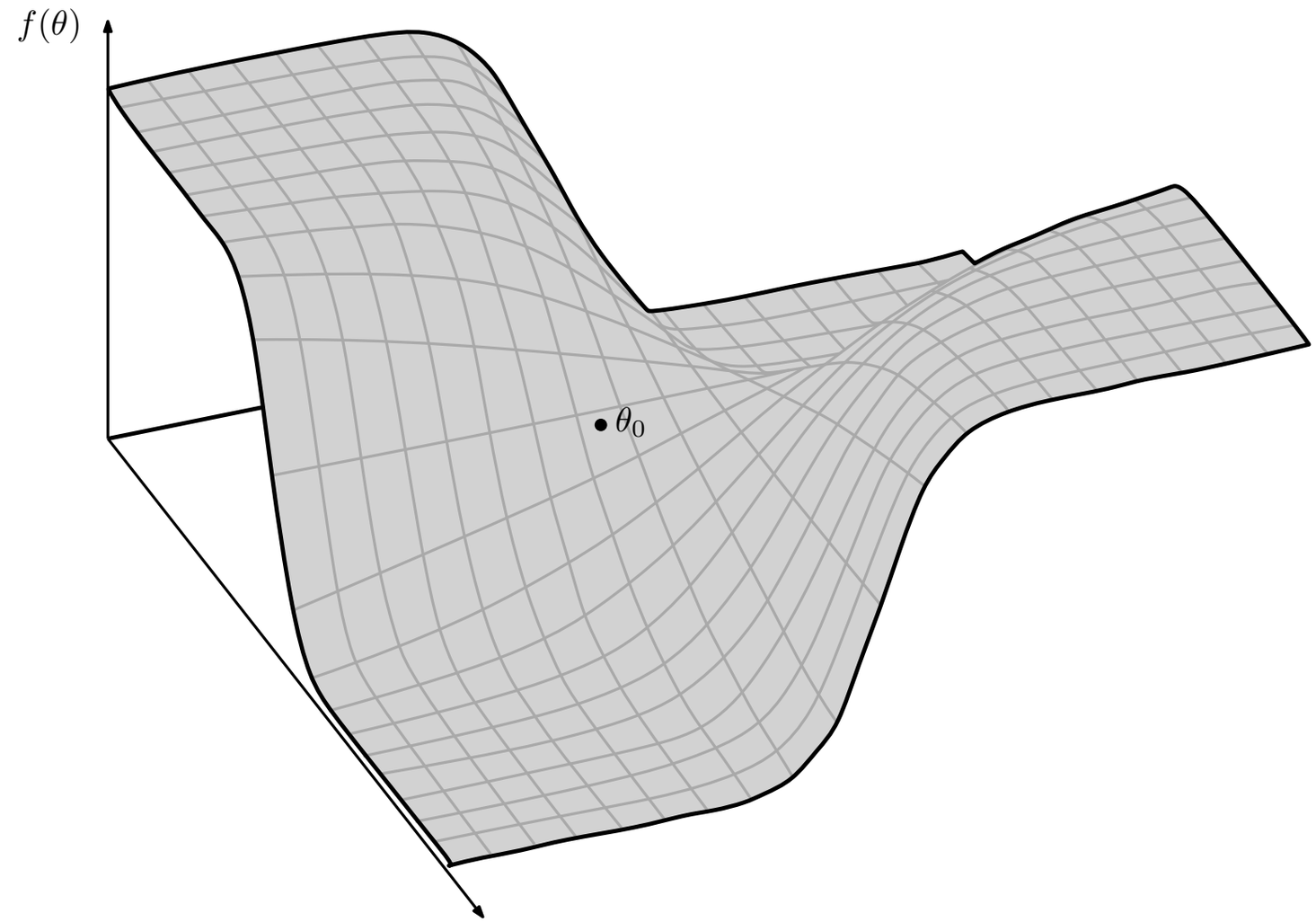
Gradient descent

- Start somewhere



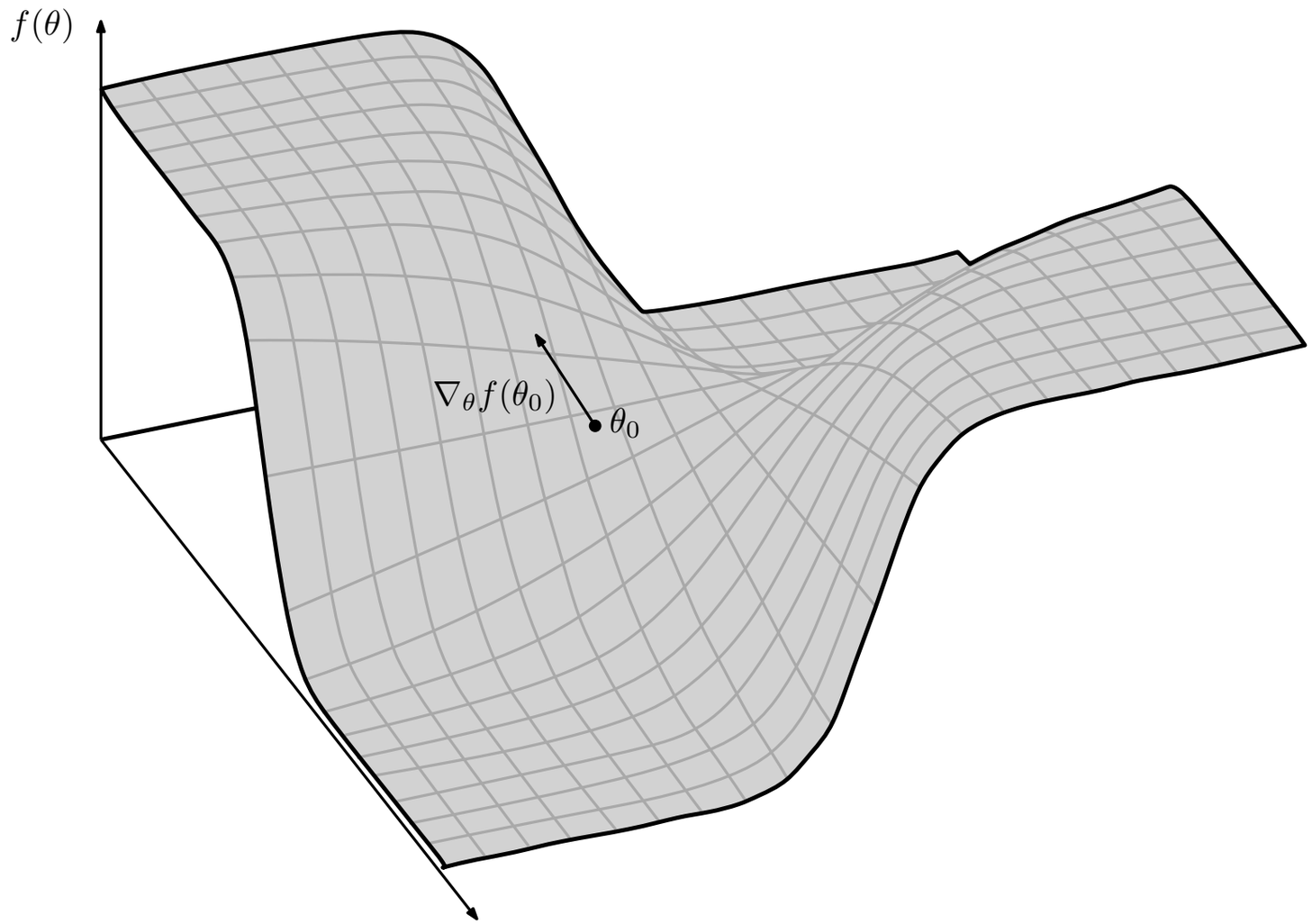
Gradient descent

- Start somewhere



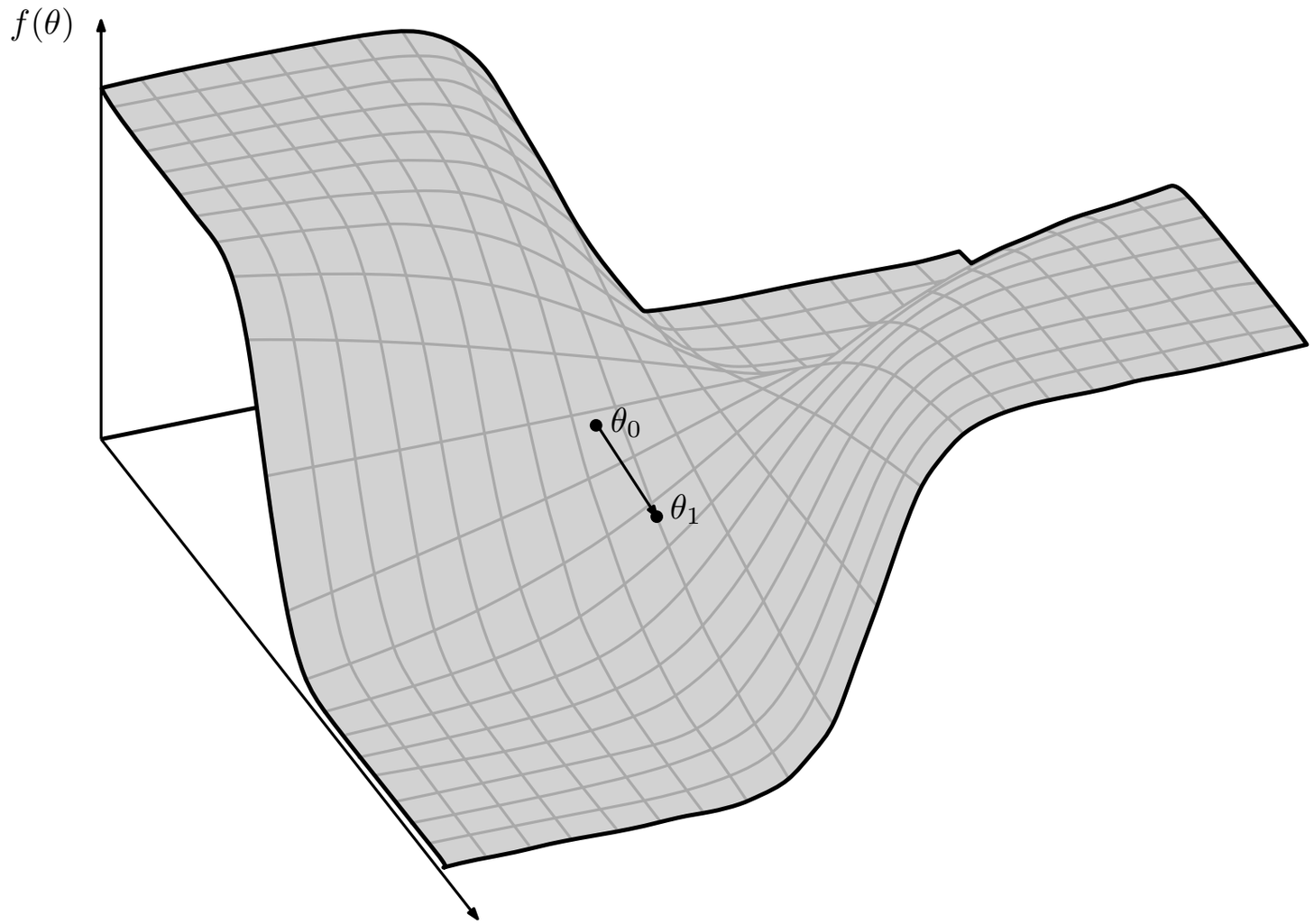
Gradient descent

- Start somewhere
- Compute gradient



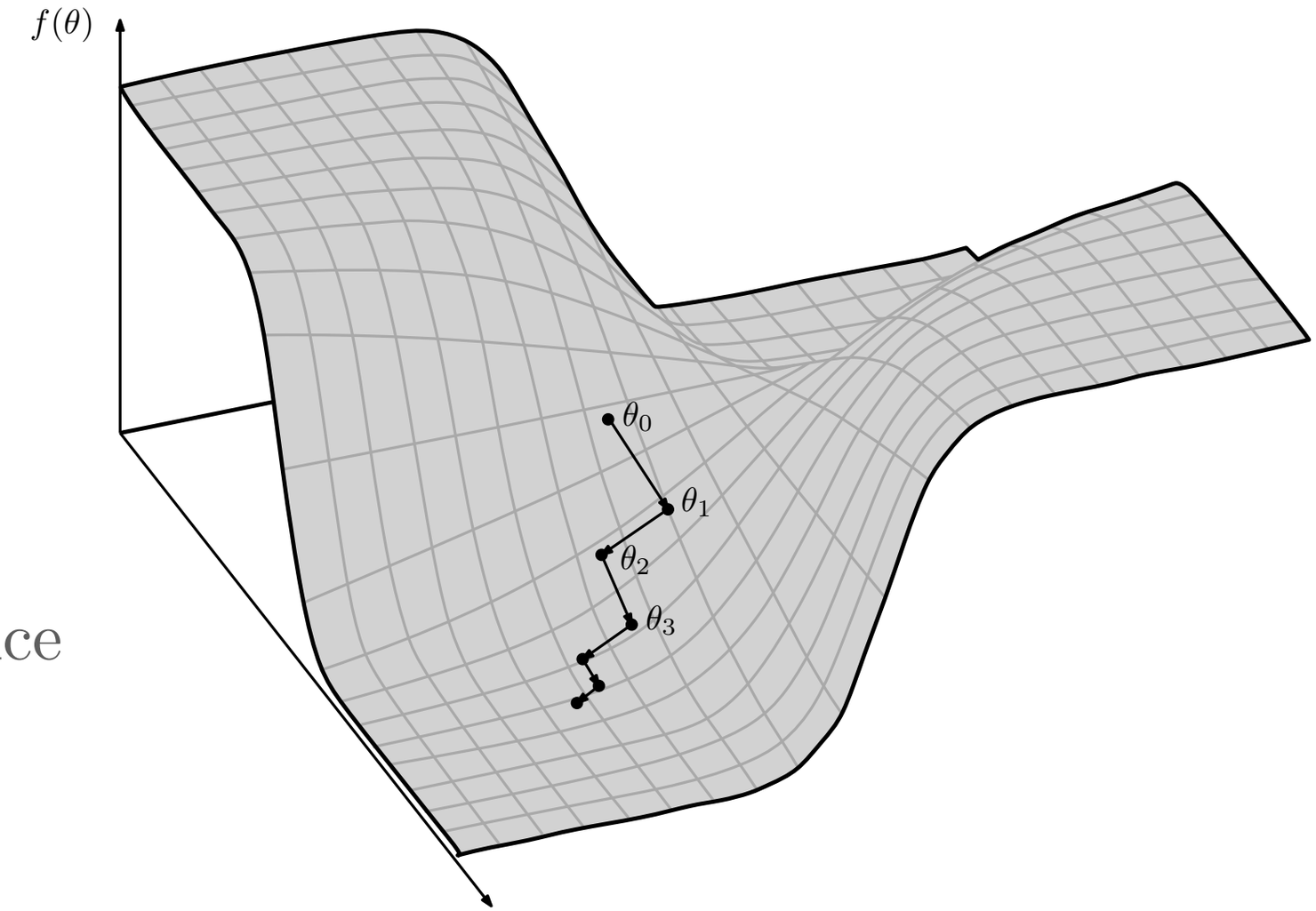
Gradient descent

- Start somewhere
- Compute gradient
- Take a step in the opposite direction



Gradient descent

- Start somewhere
- Compute gradient
- Take a step in the opposite direction
- Repeat until convergence



Expectations?

- What if f is an expectation?

$$f(\theta) = \mathbb{E}[F(\mathrm{x}, \theta)]$$

Gradient

$$\nabla_\theta f(\theta) = \mathbb{E}[\nabla_\theta F(\mathrm{x}, \theta)]$$

- Gradient descent then comes

$$\theta_{t+1} = \theta_t - \alpha \mathbb{E}[\nabla_\theta F(\mathrm{x}, \theta_t)]$$

Unknown?

Stochastic approximation!

Stochastic gradient descent

- We want to compute the solution to

$$\mathbb{E}[-\nabla_\theta F(\mathrm{x}, \theta)] = 0$$

- We apply our stochastic approximation trick to get

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta F(x_t, \theta_t)$$

Sample

Gradient descent

Starting with prediction...

Computing J^π

- Ideally, we want to compute J such that

$$J = \arg\min_\theta \mathbb{E}_\mu \left[(J^\pi(\mathrm{x}) - J_\theta(\mathrm{x}))^2 \right]$$

Stochastic gradient descent

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t)(J^\pi(x_t) - J_{\theta_t}(x_t))$$

… problem?

Computing J^π

- Ideally, we want to compute J such that

$$J = \arg\min_\theta \mathbb{E}_\mu \left[(J^\pi(\mathrm{x}) - J_\theta(\mathrm{x}))^2 \right]$$

Stochastic gradient descent

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t)(J^\pi(x_t) - J_{\theta_t}(x_t))$$

We don't know $J^\pi(x_t)$

Computing J^π

- Ideally, we want to compute J such that

$$J = \arg\min_\theta \mathbb{E}_\mu \left[(J^\pi(\mathrm{x}) - J_\theta(\mathrm{x}))^2 \right]$$

Stochastic gradient descent

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t)(\text{target} - J_{\theta_t}(x_t))$$

Replacement for $J^\pi(x_t)$

Monte Carlo target

- Given the stochastic gradient descent update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t)(\text{target} - J_{\theta_t}(x_t))$$

... we can use a Monte Carlo target:

$$\text{target} = \sum_{n=0}^{N} \gamma^n c_{t+n}$$

... yielding

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_\theta(x_t) \left(\sum_{n=0}^{N} \gamma^n c_{t+n} - J_\theta(x_t) \right)$$

Monte Carlo Policy Evaluation

- Given a trajectory obtained with policy π

$$\tau = \{x_t, c_t, x_{t+1}, c_{t+1}, \ldots, c_{t+N}, x_{t+N+1}\}$$

- Compute loss

$$L(\tau) = \sum_{n=0}^{N} \gamma^n c_{t+n}$$

- Update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t)(L(\tau) - J_{\theta_t}(x_t))$$

Monte Carlo Policy Evaluation

- Compare...
  - ... the exact update from the previous lecture:

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t \left(L(\tau) - J_t(x_t)\right)$$

  - ... the update with function approximation:

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t)(L(\tau) - J_{\theta_t}(x_t))$$

Gradient term

TD(0) target

- Given the stochastic gradient descent update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t)(\text{target} - J_{\theta_t}(x_t))$$

... we can use a TD(0) target:

$$\text{target} = c_t + \gamma J_{\theta_t}(x_{t+1})$$

Uses current estimate (*bootstraps*)

... yielding

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t)\left(c_t + \gamma J_{\theta_t}(x_{t+1}) - J_{\theta_t}(x_t)\right)$$

No longer stochastic gradient descent

TD(0) with function approximation

- Given a sample (x_t, c_t, x_{t+1}), where the action was selected from π
- Update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t) \left(c_t + \gamma J_{\theta_t}(x_{t+1}) - J_{\theta_t}(x_t)\right)$$

TD(0) with function approximation

- Compare...
  - ... the exact update from the previous lecture:

$$J_{t+1}(x_t) = J_t(x_t) + \alpha_t \left[c_t + \gamma J_t(x_{t+1}) - J_t(x_t)\right]$$

  - ... the update with function approximation:

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t) \left(c_t + \gamma J_{\theta_t}(x_{t+1}) - J_{\theta_t}(x_t)\right)$$

Gradient term

N-step TD target

- Given the stochastic gradient descent update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t)(\text{target} - J_{\theta_t}(x_t))$$

... we can use a N-step TD target:

$$\text{target} = \sum_{n=0}^{N} \gamma^n c_{t+n} + \gamma^{N+1} J_{\theta_t}(x_{t+N+1})$$

... yielding

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t) \left(\sum_{n=0}^{N} \gamma^n c_{t+n} + \gamma^{N+1} J_{\theta_t}(x_{t+N+1}) - J_{\theta_t}(x_t) \right)$$

N-step TD target

N-step TD learning with function approximation

- Given a sample $(x_t, c_t, c_{t+1}, \ldots, c_{t+N}, x_{t+N+1})$ obtained with policy π
- Compute

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t) \left(\sum_{n=0}^{N} \gamma^n c_{t+n} + \gamma^{N+1} J_{\theta_t}(x_{t+N+1}) - J_{\theta_t}(x_t) \right)$$

TD(λ) target

- Given the stochastic gradient descent update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta J_{\theta_t}(x_t)(\text{target} - J_{\theta_t}(x_t))$$

...we can use a TD(λ) target:

$$\text{target} = \sum_{n=0}^{\infty} (\lambda\gamma)^n (c_{t+n} + \gamma J_{\theta_t}(x_{t+n+1}) - J_{\theta_t}(x_{t+n}))$$

... which, using eligibility traces, yields:

$$z_{t+1} = \lambda\gamma z_t + \nabla_\theta J_{\theta_t}(x_t)$$
$$\theta_{t+1} = \theta_t + \alpha_t z_{t+1}[c_t + \gamma J_{\theta_t}(x_{t+1}) - J_{\theta_t}(x_t)]$$

TD(λ) target

TD(λ) with function approximation

- Given a sample (x_t, c_t, x_{t+1}), where the action was selected from π
- Compute

$$z_{t+1} = \lambda\gamma z_t + \nabla_\theta J_{\theta_t}(x_t)$$
$$\theta_{t+1} = \theta_t + \alpha_t z_{t+1}[c_t + \gamma J_{\theta_t}(x_{t+1}) - J_{\theta_t}(x_t)]$$

What about control?

Computing Q^π

- Ideally, we want to compute Q such that

$$Q = \arg\min_\theta \mathbb{E}_\mu \left[(Q^\pi(\mathrm{x}, \mathrm{a}) - Q_\theta(\mathrm{x}, \mathrm{a}))^2 \right]$$

Stochastic gradient descent

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_{\theta_t}(x_t, a_t)(Q^\pi(x_t, a_t) - Q_{\theta_t}(x_t, a_t))$$

We don't know $Q^\pi(x_t, a_t)$

Computing Q^π

- Ideally, we want to compute Q such that

$$Q = \arg\min_\theta \mathbb{E}_\mu \left[(Q^\pi(\mathrm{x}, \mathrm{a}) - Q_\theta(\mathrm{x}, \mathrm{a}))^2 \right]$$

Stochastic gradient descent

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_{\theta_t}(x_t, a_t)(\text{target} - Q_{\theta_t}(x_t, a_t))$$

Replacement for $Q^\pi(x_t, a_t)$

Monte Carlo target

- Given the stochastic gradient descent update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_{\theta_t}(x_t, a_t)(\text{target} - Q_{\theta_t}(x_t, a_t))$$

... we can use a Monte Carlo target:

$$\text{target} = \sum_{n=0}^{N} \gamma^n c_{t+n}$$

... yielding

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_\theta(x_t, a_t) \left(\sum_{n=0}^{N} \gamma^n c_{t+n} - Q_\theta(x_t, a_t) \right)$$

Monte Carlo policy evaluation

- Given a trajectory obtained with policy π

$$\tau = \{x_t, a_t, c_t, x_{t+1}, a_{t+1}, c_{t+1}, \ldots, a_{t+N}, c_{t+N}, x_{t+N+1}\}$$

- Compute loss

$$L(\tau) = \sum_{n=0}^{N} \gamma^n c_{t+n}$$

- Update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_{\theta_t}(x_t, a_t)(L(\tau) - Q_{\theta_t}(x_t, a_t))$$

Temporal difference target

- Given the stochastic gradient descent update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_{\theta_t}(x_t, a_t)(\text{target} - Q_{\theta_t}(x_t, a_t))$$

... we can use a temporal difference target:

$$\text{target} = c_t + \gamma Q_{\theta_t}(x_{t+1}, a_{t+1})$$

Uses current estimate (*bootstraps*)

... yielding

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_\theta(x_t, a_t)\left(c_t + \gamma Q_{\theta_t}(x_{t+1}, a_{t+1}) - Q_\theta(x_t, a_t)\right)$$

No longer stochastic gradient descent

SARSA with function approximation

- Given a sample $(x_t, a_t, c_t, x_{t+1}, a_{t+1})$,
- Update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_\theta(x_t, a_t) \left(c_t + \gamma Q_{\theta_t}(x_{t+1}, a_{t+1}) - Q_\theta(x_t, a_t)\right)$$

Generalized policy iteration

- Both Monte Carlo policy evaluation and SARSA can be used as part of **generalized policy iteration**:
  - We start with random policy π_0
  - We evaluate the policy (compute Q^{π_0})
  - We compute an improved policy π_1
  - **Repeat**

Computing Q^*

- Ideally, we want to compute Q such that

$$Q = \arg\min_{\theta} \mathbb{E}_{\mu} \left[(Q^*(\mathrm{x}, \mathrm{a}) - Q_\theta(\mathrm{x}, \mathrm{a}))^2 \right]$$

Stochastic gradient descent

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_{\theta_t}(x_t, a_t)(Q^*(x_t, a_t) - Q_{\theta_t}(x_t, a_t))$$

We don't know $Q^*(x_t, a_t)$

Computing Q^π

- Ideally, we want to compute Q such that

$$Q = \arg\min_\theta \mathbb{E}_\mu \left[(Q^*(\mathrm{x}, \mathrm{a}) - Q_\theta(\mathrm{x}, \mathrm{a}))^2 \right]$$

Stochastic gradient descent

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_{\theta_t}(x_t, a_t)(\text{target} - Q_{\theta_t}(x_t, a_t))$$

Replacement for $Q^*(x_t, a_t)$

Temporal difference target

- Given the stochastic gradient descent update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_{\theta_t}(x_t, a_t)(\text{target} - Q_{\theta_t}(x_t, a_t))$$

... we can use a temporal difference target:

$$\text{target} = c_t + \gamma \min_{a' \in \mathcal{A}} Q_{\theta_t}(x_{t+1}, a')$$

Uses current estimate (*bootstraps*)

... yielding

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_\theta(x_t, a_t) \left(c_t + \gamma \min_{a' \in \mathcal{A}} Q_{\theta_t}(x_{t+1}, a') - Q_\theta(x_t, a_t) \right)$$

No longer stochastic gradient descent

Q-learning with function approx.

- Given a sample (x_t, a_t, c_t, x_{t+1}),
- Update

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta Q_\theta(x_t, a_t) \left(c_t + \gamma \min_{a' \in \mathcal{A}} Q_{\theta_t}(x_{t+1}, a') - Q_\theta(x_t, a_t) \right)$$

Do these work?

Monte Carlo methods

- Monte Carlo methods **are** stochastic gradient descent methods
- Under mild conditions, they do converge
  - May be stuck in **local minima**

Temporal difference methods

- What can we say about:
  - ... TD(λ)?
  - ... SARSA?
  - ... Q-learning?

Linear approximation

- Each state/state-action pair is described as a vector of "features"

$$\phi(x) \qquad \text{or} \qquad \phi(x,a)$$

- J and Q are represented as combinations of features

$$J^\pi(x) \approx J_\theta(x) = \phi^\top(x)\theta \qquad Q^*(x,a) \approx Q_\theta(x,a) = \phi^\top(x,a)\theta$$

- We have that

$$\nabla_\theta J_\theta(x) = \phi(x)$$
$$\nabla_\theta Q_\theta(x,a) = \phi(x,a)$$

TD(λ)

TD(λ) with <u>linear</u> function approximation

- Given a sample (x_t, c_t, x_{t+1}), where the action was selected from π

- Compute

$$z_{t+1} = \lambda\gamma z_t + \phi(x_t)$$

$$\theta_{t+1} = \theta_t + \alpha_t z_{t+1}[c_t + \gamma J_{\theta_t}(x_{t+1}) - J_{\theta_t}(x_t)]$$

TD(λ)

Theorem: For all $0 \leq \lambda \leq 1$, TD(λ) with linear function approximation converges, as long as:

- Policy π induces an ergodic Markov chain
- The stationary distribution μ is such that the matrix

$$\mathbb{E}_{\mathrm{x} \sim \mu}[\phi(\mathrm{x})\phi^{\top}(\mathrm{x})]$$

is non-singular

SARSA

SARSA with linear function approximation

- Given a sample $(x_t, a_t, c_t, x_{t+1}, a_{t+1})$,

- Update

$$\theta_{t+1} = \theta_t + \alpha_t \phi(x_t, a_t) \left(c_t + \gamma Q_{\theta_t}(x_{t+1}, a_{t+1}) - Q_\theta(x_t, a_t)\right)$$

SARSA

Theorem: SARSA with linear function approximation converges as long as

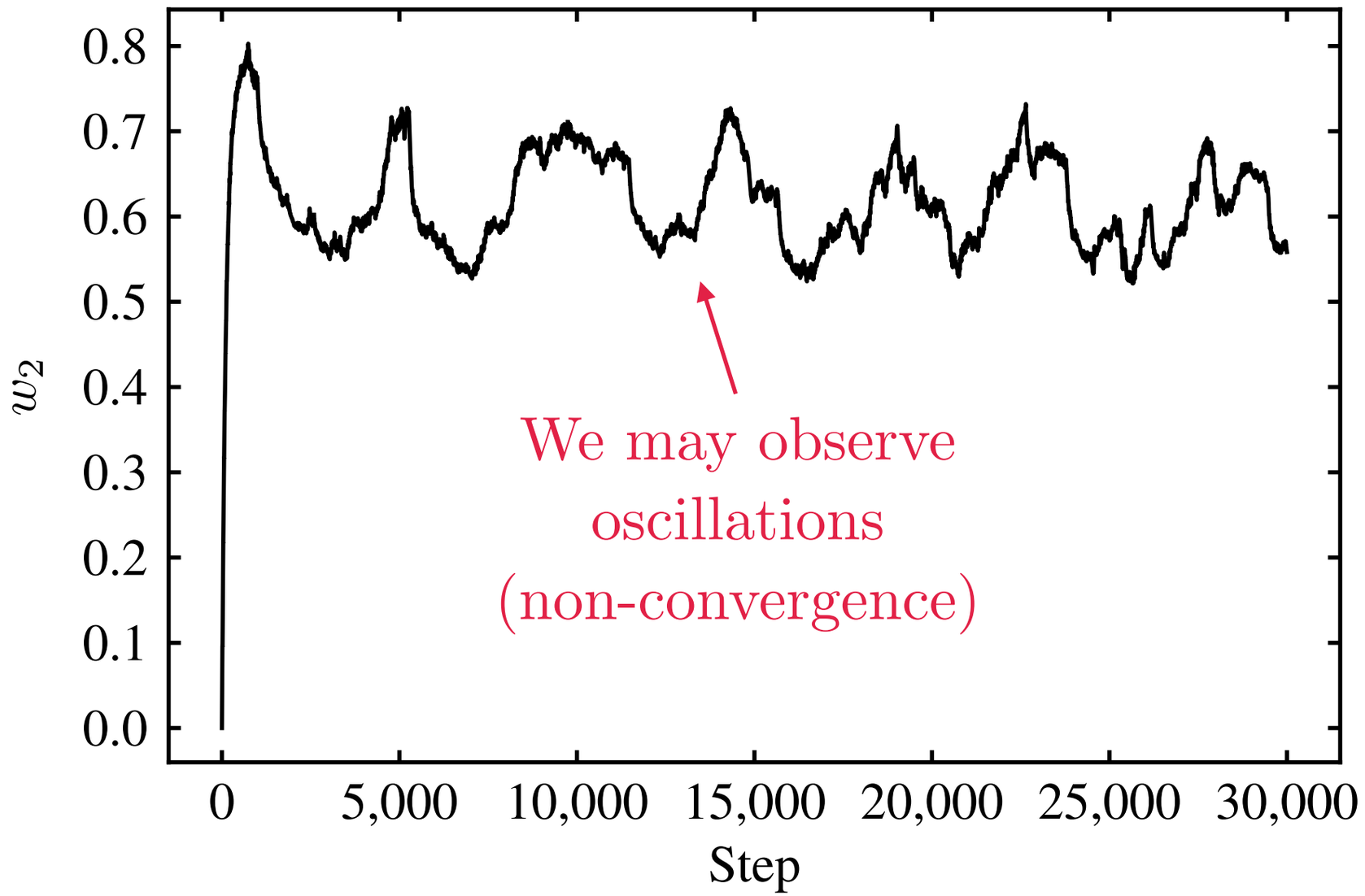
- Policy π induces an ergodic Markov chain
- The stationary distribution μ is such that the matrix

$$\mathbb{E}_{\mathrm{x},\mathrm{a}\sim\mu}[\phi(\mathrm{x},\mathrm{a})\phi^{\top}(\mathrm{x},\mathrm{a})]$$

is non-singular

SARSA

- What if we use an ϵ-greedy policy with decaying ϵ?



Q-learning

Q-learning with linear function approximation

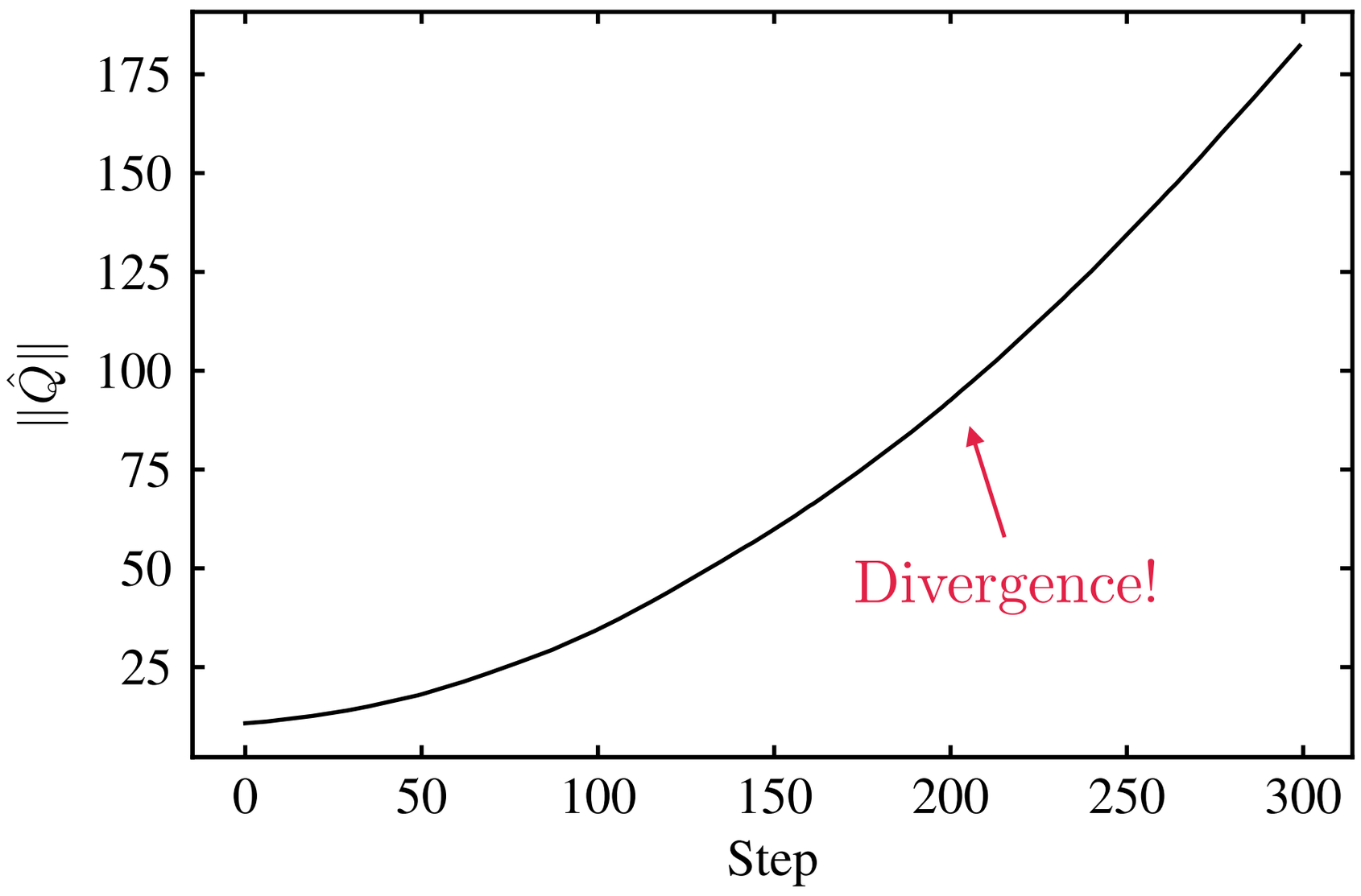
- Given a sample (x_t, a_t, c_t, x_{t+1}),

- Update

$$\theta_{t+1} = \theta_t + \alpha_t \phi(x_t, a_t) \left(c_t + \gamma \min_{a' \in \mathcal{A}} Q_{\theta_t}(x_{t+1}, a') - Q_\theta(x_t, a_t) \right)$$

Q-learning

Theorem?



Why?

- 3 elements that—together—may lead to divergence:
  - Function approximation ⟵ Exact methods do not diverge...
  - Bootstrapping ⟵ All TD-based methods bootstrap
  - Off-policy updates ⟵ Only Q-learning is off-policy

Summarizing...

- TD(λ) does retain its convergence guarantees
- SARSA is more stable than Q-learning, as long as the learning policy changes smoothly with Q_t
- Q-learning with function approximation does not retain its convergence guarantees

C:\>batch

Batch RL

Batch RL

- All methods seen so far are *online*
  - Agent collects samples and performs immediate updates
  - Samples are not used again

Batch RL

- Alternatively, agent could collect a big batch of data and use it to learn *offline*
  - Hopefully, we'll make better use of data...
  - Maybe learning will be more stable?
  - Maybe we can use supervised learning?

Can we turn a **reinforcement learning problem** into a **supervised learning problem**?

Turning RL into SL

Prediction:

- Given a batch of transitions obtained with policy π,

$$\mathcal{D} = \{(x_n, c_n, x'_n), n = 1, \ldots, N\}$$

$$x'_n \sim \mathbf{P}_\pi(x_n)$$

Turning RL into SL

Prediction:

- Given a batch of transitions obtained with policy π,

$$\mathcal{D} = \{(x_n, c_n, x'_n), n = 1, \ldots, N\}$$

... can we learn J^π using supervised learning?

Turning RL into SL

- We have all the ingredients to do it!
- Recall that we are looking for J such that

$$J = \arg\min_{\theta} \mathbb{E}_{\mu}\left[(J^{\pi}(\mathrm{x}) - J_{\theta}(\mathrm{x}))^2\right]$$

$J_\theta(x)$ is the output

$J^\pi(x)$ is the desired output

x is the input

Turning RL into SL

- We can turn learning J^π into a **regression problem**
  - **Task:** Learn J^π
  - **Experience:** We build a "supervised" dataset from $\mathcal{D}$

$$\mathcal{D}_{\text{supervised}} = \{(x_n, \text{target}_n), n = 1, \ldots, N\}$$

  - **Performance:** We use the loss

$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} (\text{target}_n - J_\theta(x_n))^2$$

Turning RL into SL

- How do we build the supervised dataset?
  - Given a transition (x_n, c_n, x'_n) from $\mathcal{D}$ we set

$$\text{target}_n = c_n + \gamma J(x'_n)$$

TD target (*bootstraps*)

Current estimate

Turning RL into SL

- How do we build the supervised dataset?
  - Given a transition (x_n, c_n, x'_n) from $\mathcal{D}$ we set

$$\text{target}_n = c_n + \gamma J(x'_n)$$

- We get a dataset

$$\mathcal{D}_{\text{supervised}} = \{(x_n, c_n + \gamma J(x'_n)), n = 1, \ldots, N\}$$

Fitted Value Iteration

- Initialize $k = 0,\ J_0 = 0$
- Repeat
  - Build dataset

$$\mathcal{D}_{k+1} = \{(x_n, c_n + \gamma J_k(x'_n)), n = 1, \ldots, N\}$$

  - Compute J_{k+1} by minimizing

$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} (c_n + \gamma J_k(x'_n) - J_\theta(x_n))^2$$

Turning RL into SL

Control:

- Given a batch of transitions

$$\mathcal{D} = \{(x_n, a_n, c_n, x'_n), n = 1, \ldots, N\}$$

$$x'_n \sim \mathbf{P}_{a_n}(x_n)$$

Turning RL into SL

Control:

- Given a batch of transitions

$$\mathcal{D} = \{(x_n, a_n, c_n, x'_n), n = 1, \ldots, N\}$$

... can we learn Q^* using supervised learning?

Turning RL into SL

- We can turn learning Q^* into a **regression problem**
  - **Task:** Learn Q^*
  - **Experience:** We build a "supervised" dataset from $\mathcal{D}$

$$\mathcal{D}_{\text{supervised}} = \{(x_n, a_n, \text{target}_n), n = 1, \ldots, N\}$$

  - **Performance:** We use the loss

$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} (\text{target}_n - Q_\theta(x_n, a_n))^2$$

Turning RL into SL

- How do we build the supervised dataset?
  - Given a transition (x_n, a_n, c_n, x'_n) from $\mathcal{D}$ we set

$$\text{target}_n = c_n + \gamma \min_{a' \in \mathcal{A}} Q(x'_n, a')$$

QL target (*bootstraps*)

Current estimate

Turning RL into SL

- How do we build the supervised dataset?
  - Given a transition (x_n, a_n, c_n, x'_n) from $\mathcal{D}$ we set

$$\text{target}_n = c_n + \gamma \min_{a' \in \mathcal{A}} Q(x'_n, a')$$

- We get a dataset

$$\mathcal{D}_{\text{supervised}} = \{(x_n, a_n, c_n + \gamma \min_{a'} Q(x'_n, a')), n = 1, \ldots, N\}$$

Fitted Q-Iteration

- Initialize $k = 0$, $Q_0 = 0$
- Repeat
  - Build dataset

$$\mathcal{D}_{k+1} = \{(x_n, a_n, c_n + \gamma \min_{a'} Q_k(x'_n, a')), n = 1, \ldots, N\}$$

  - Compute Q_{k+1} by minimizing

$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} (c_n + \gamma \min_{a'} Q_k(x'_n, a') - Q_\theta(x_n, a_n))^2$$

... however...

- **Problem 1:** Supervised learning algorithms typically assume that the datapoints are **independent**
  - The data in FVI and FQI comes from trajectories (not independent)...
- **Problem 2:** The update for FQI is similar to that of QL
  - It is still **off-policy** and still **bootstraps**
  - **It may still diverge!**

Two clever ideas

- **Problem 1:** Supervised learning algorithms typically assume that the datapoints are **independent**
  - We don't use the whole dataset, but just some (random) transitions (avoid trajectories)
- **Problem 2:** The update for FQI is similar to that of QL
  - We build the targets from an **independent estimator** (avoid bootstrapping)

Fitted Q-iteration with independent target estimator

- Initialize $k = 0$, $Q_0 = 0$, $Q_{\text{target}} = 0$
- Repeat
  - Build dataset

$$\mathcal{D}_{k+1} = \{(x_n, a_n, c_n + \gamma \min_{a'} Q_{\text{target}}(x'_n, a')), n = 1, \ldots, N\}$$

  - Compute Q_{k+1} by minimizing

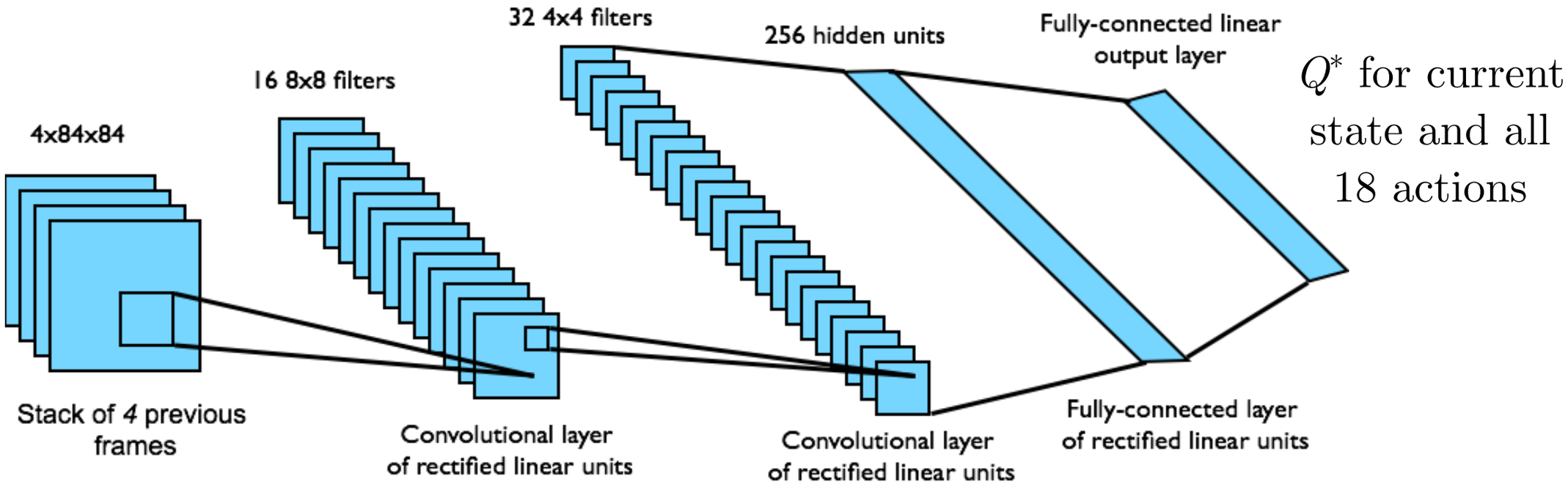
$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} (c_n + \gamma \min_{a'} Q_{\text{target}}(x'_n, a') - Q_\theta(x_n, a_n))^2$$

- Every now and then, set $Q_{\text{target}} = Q_k$

Deep Q-networks

- For example, if Q_θ is a CNN, we get the DQN algorithm

Pixels of last 4 frames (state)

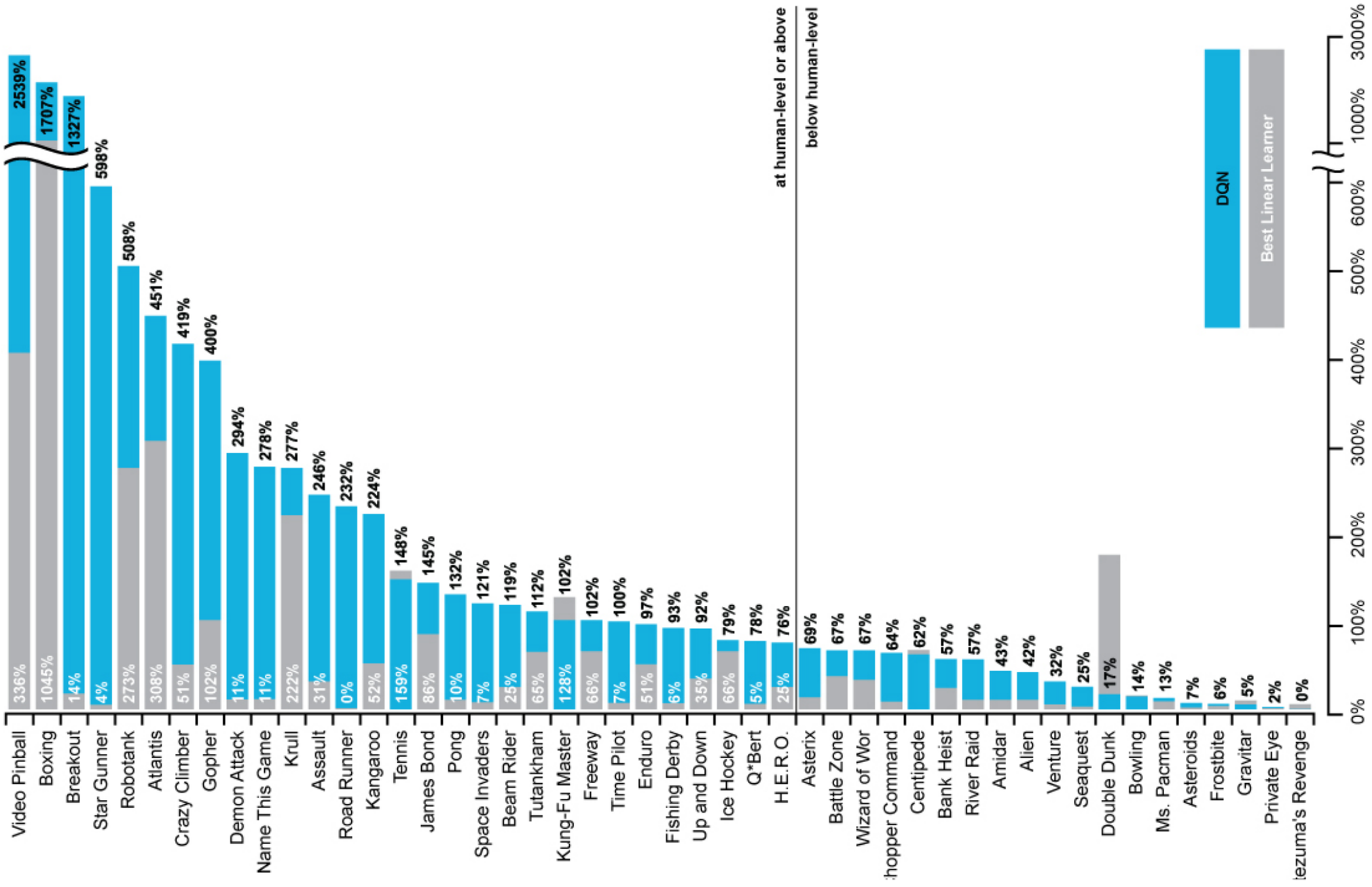


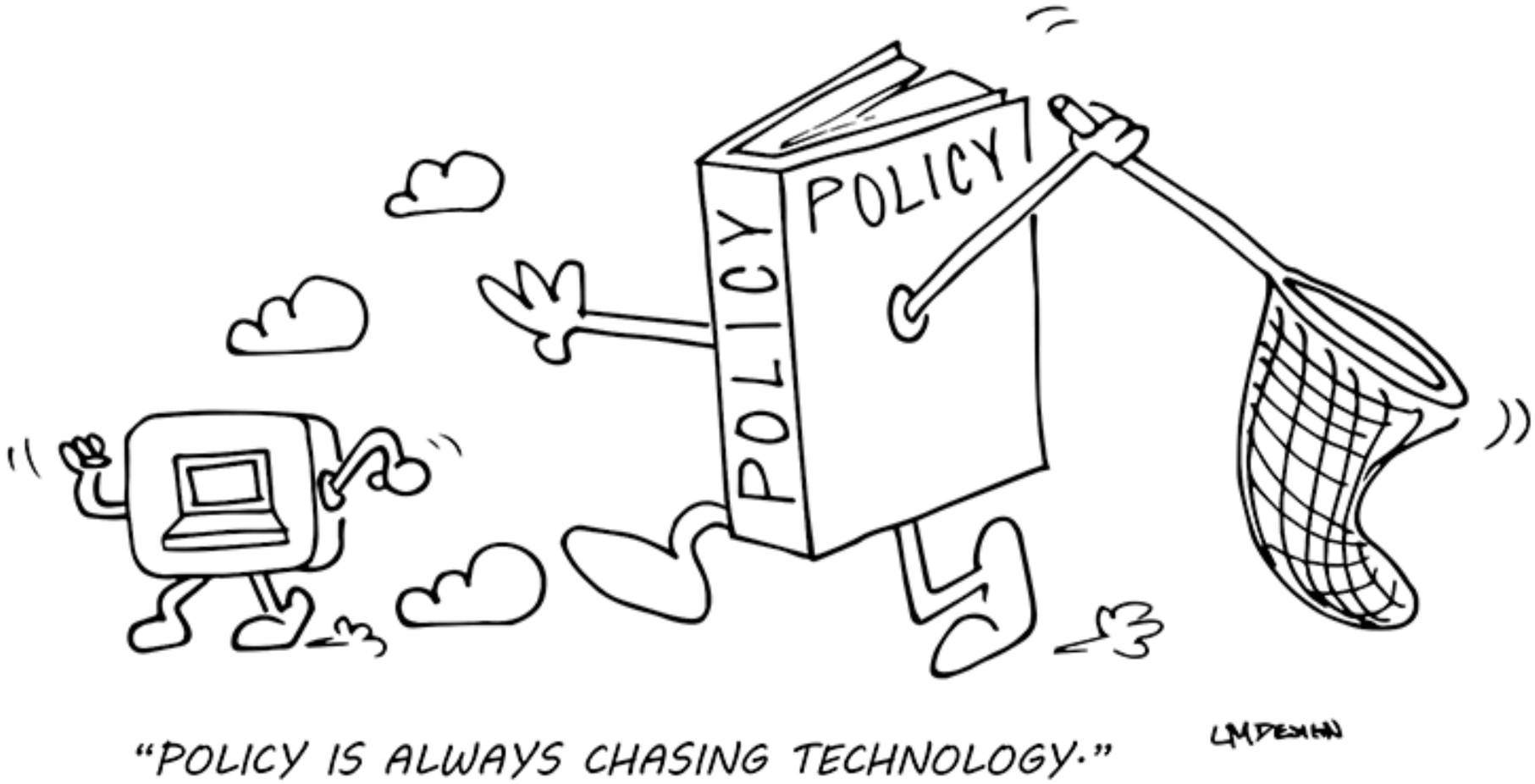
Q^* for current state and all 18 actions

Deep Q-networks

- DQN learned how to play Atari 2600 games from pixels

Deep Q-networks





Policy-based methods

Example

- In some MDP, the Q-values in state x are

a_1	a_2
5	6

- Which is the optimal action?
  - a_1: minimum cost

Example

- In some MDP, the Q-values in state x are

a_1	a_2
5	6

- Consider the two approximations

a_1	a_2
6	5.5

a_1	a_2
0	1

Which one has smaller error?

Example

- In some MDP, the Q-values in state x are

a_1	a_2
5	6

- Consider the two approximations

a_1	a_2
6	5.5

a_1	a_2
0	1

Which one is better?

Why policy-based RL?

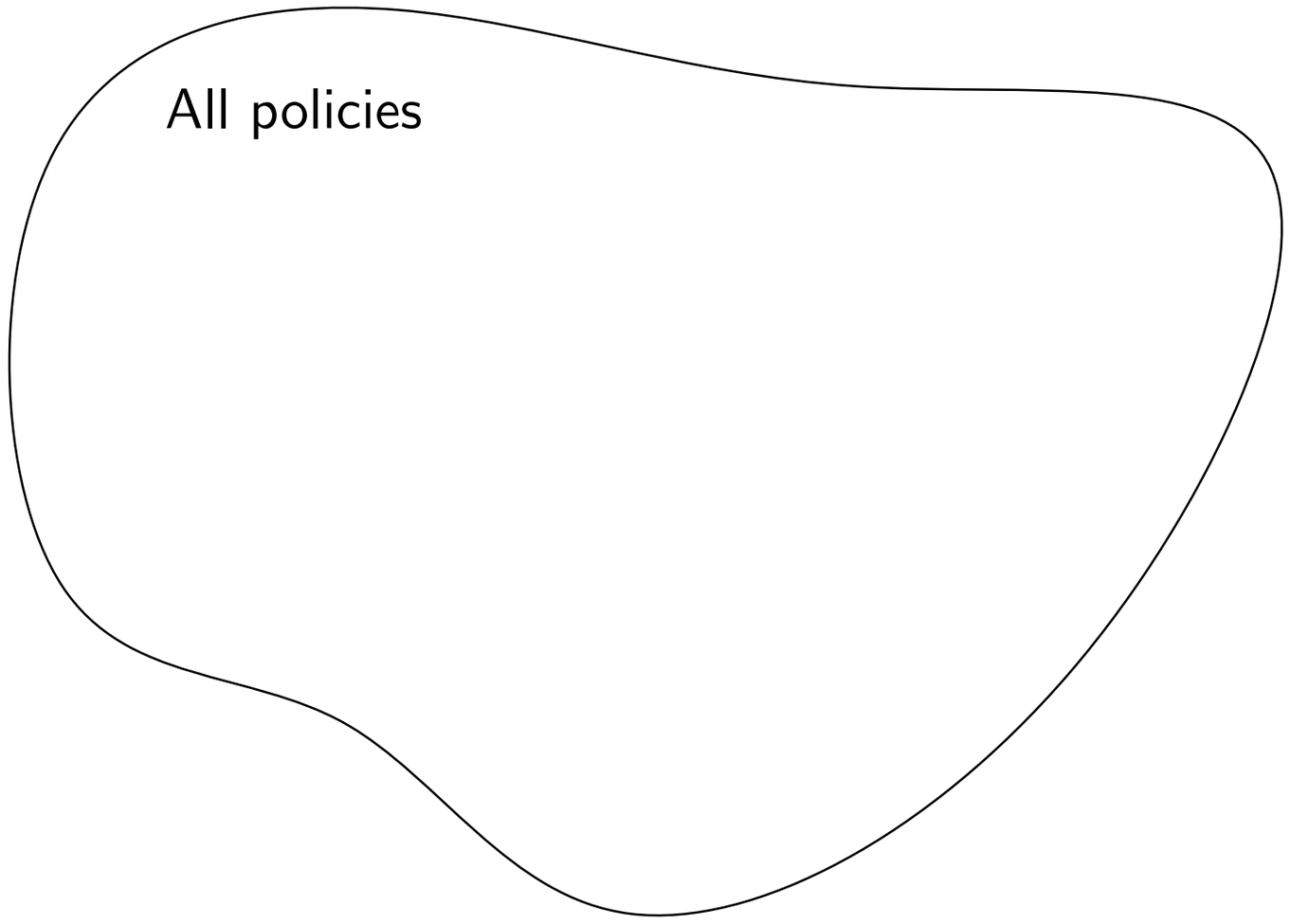
- Value-based methods are not guaranteed to converge with approximations
- Quality of the result depends on the quality of approximation
- Even if the approximation is good, the resulting policy may be bad

What is policy-based RL?

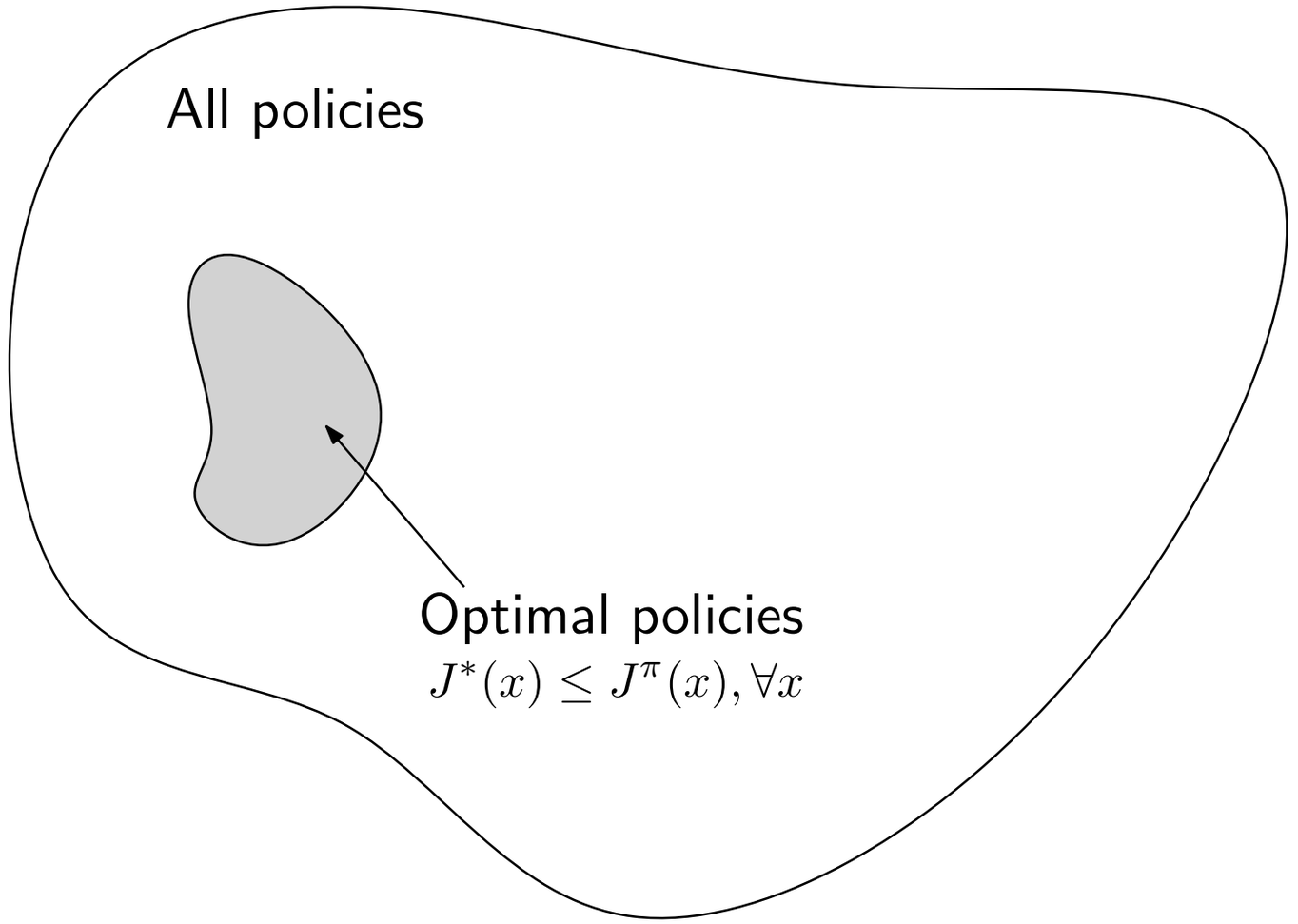
- Instead of selecting an approximation for the value, we select an approximation for the policy
- Policy-based methods select the best policy within the approximation

What is the “best policy”?

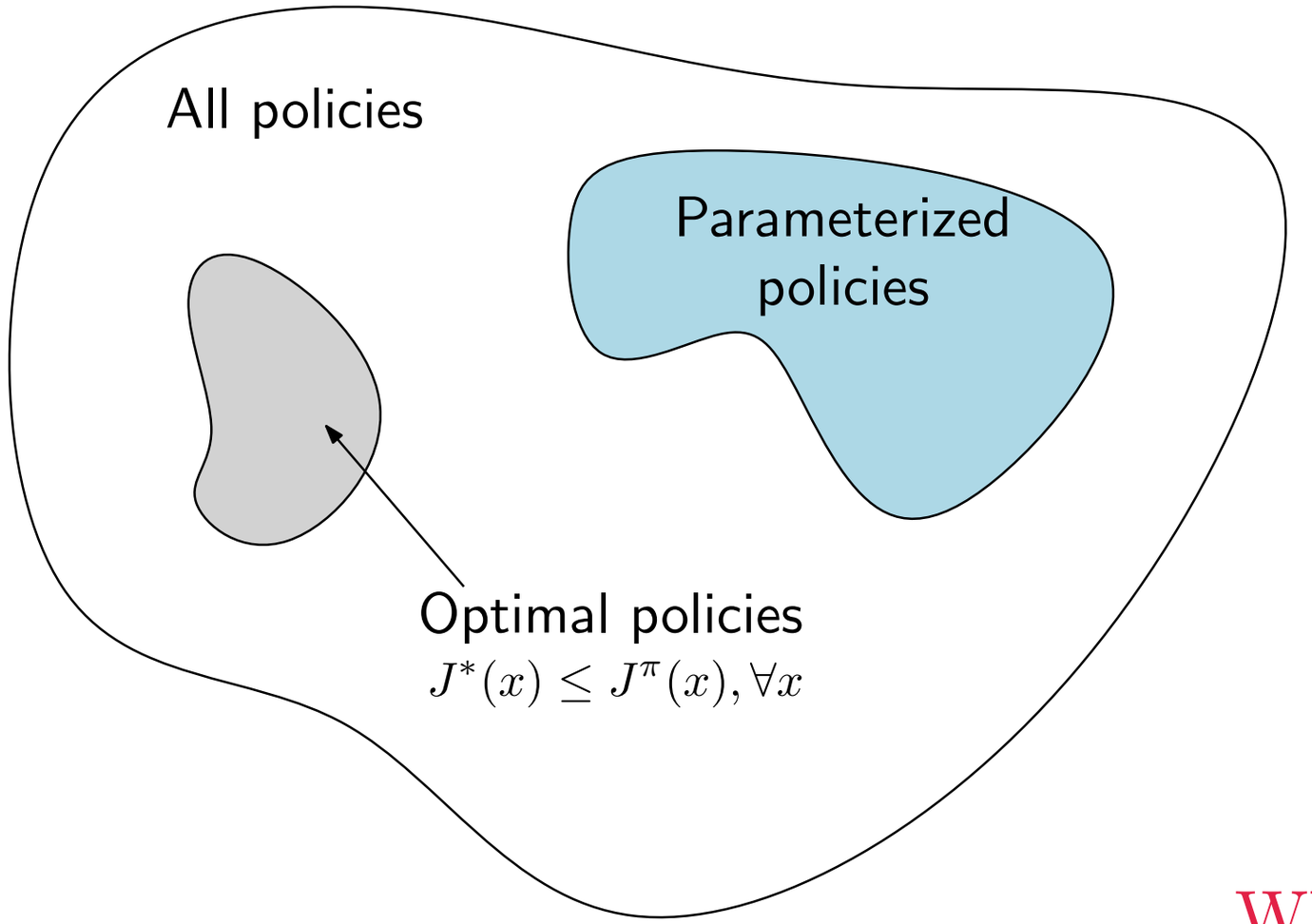
Revisiting optimality



Revisiting optimality



Revisiting optimality



What is "best"?

Example

- Suppose, that, for two policies π_1 and π_2,

$$\mathbf{J}^{\pi_1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \mathbf{J}^{\pi_2} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- Which one is "best"?

Initial distribution

- We consider the initial distribution μ_0 and set

$$V(\pi) = \mathbb{E}_{\mu_0}[J^\pi(\mathrm{x}_0)]$$
$$= \sum_{x \in \mathcal{X}} \mu_0(x) J^\pi(x)$$

Single number

Initial distribution

- We consider the initial distribution μ_0 and set

$$\begin{aligned} V(\pi) &= \mathbb{E}_{\mu_0}[J^\pi(\mathrm{x}_0)] \\ &= \sum_{x \in \mathcal{X}} \mu_0(x) J^\pi(x) \end{aligned}$$

- A policy π_1 is better than π_2 if $V(\pi_1) < V(\pi_2)$

Policy improvement

- Several approaches:
  - Hill climbing
  - Evolutionary algorithms
  - ...

BBO
Black box optimization

- We focus on **gradient descent:**
  - More efficient
  - Exploits MDP structure better

Policy gradient

- We consider a family of parameterized policies $\{\pi_w, w \in \mathbb{R}^P\}$
- w is the parameter of the policy
- Example:

$$\pi_w(a \mid x) = \frac{e^{w^\top \phi(x,a)}}{\sum_{a' \in \mathcal{A}} e^{w^\top \phi(x,a')}}$$

  ... many other possibilities

- V is a function of w

How to compute the gradient?

Policy gradient

- Assume that we know $\nabla_w \pi_w$
- Then,

$$\nabla_w V(w) = \nabla_w \mu_0 J^{\pi_w}$$

Does not depend on θ

Policy gradient

- Assume that we know $\nabla_w \pi_w$
- Then,

$$\nabla_w V(w) = \mu_0 \boxed{\nabla_w J^{\pi_w}}$$

Let us consider this term

Policy gradient

- For any policy π,

$$J^\pi(x) = \mathbb{E}_{\mathrm{a}\sim\pi(x)}[Q^\pi(x, \mathrm{a})]$$

or, equivalently,

$$J^\pi(x) = \sum_{a\in\mathcal{A}} \pi(a \mid x) Q^\pi(x, a)$$

Policy gradient

- Computing the gradient of J^{π_w}, we get

$$\nabla_w J^{\pi_w}(x) = \nabla_w \left[\sum_{a \in \mathcal{A}} \pi_w(a \mid x) Q^{\pi_w}(x, a) \right]$$

These two terms depend on w

Policy gradient

- Computing the gradient of J^{π_w}, we get

$$\nabla_w J^{\pi_w}(x) = \nabla_w \left[\sum_{a \in \mathcal{A}} \pi_w(a \mid x) Q^{\pi_w}(x, a) \right]$$

- Then (derivative of the product),

$$\nabla_w J^{\pi_w}(x) = \sum_{a \in \mathcal{A}} \nabla_w \pi_w(a \mid x) Q^{\pi_w}(x, a) + \sum_{a \in \mathcal{A}} \pi_w(a \mid x) \nabla_w Q^{\pi_w}(x, a)$$

Policy gradient

- Since

$$Q^{\pi_w}(x,a) = c(x,a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) J^{\pi_w}(x')$$

- Then,

$$\nabla_w Q^{\pi_w}(x,a) = \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_a(x' \mid x) \nabla_w J^{\pi_w}(x')$$

Policy gradient

- Putting everything together

$$\nabla_w J^{\pi_w}(x) = \sum_{a \in \mathcal{A}} \nabla_w \pi_w(a \mid x) Q^{\pi_w}(x, a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_{\pi_w}(x' \mid x) \nabla_w J^{\pi_w}(x')$$

- This is a recursive equation in $\nabla_w J^{\pi_w}$

- Unfolding the recursion, we get

$$\nabla_w J^{\pi_w}(x) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \sum_{a \in \mathcal{A}} \nabla_w \pi_w(a \mid \mathrm{x}_t) Q^{\pi_w}(\mathrm{x}_t, a) \mid \mathrm{x}_0 = x \right]$$

Policy gradient

- Putting everything together

$$\nabla_w J^{\pi_w}(x) = \sum_{a \in \mathcal{A}} \nabla_w \pi_w(a \mid x) Q^{\pi_w}(x, a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_{\pi_w}(x' \mid x) \nabla_w J^{\pi_w}(x')$$

- This is a recursive equation in $\nabla_w J^{\pi_w}$

- Unfolding the recursion, we get

$$\nabla_w V(w) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \sum_{a \in \mathcal{A}} \nabla_w \pi_w(a \mid \mathrm{x}_t) Q^{\pi_w}(\mathrm{x}_t, a) \mid \mathrm{x}_0 \sim \mu_0 \right]$$

$$\nabla_w \pi_w(a \mid x) = \pi_w(a \mid x) \frac{\nabla_w \pi_w(a \mid x)}{\pi_w(a \mid x)}$$

Policy gradient

- Putting everything together

$$\nabla_w J^{\pi_w}(x) = \sum_{a \in \mathcal{A}} \nabla_w \pi_w(a \mid x) Q^{\pi_w}(x, a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_{\pi_w}(x' \mid x) \nabla_w J^{\pi_w}(x')$$

- This is a recursive equation in $\nabla_w J^{\pi_w}$

- Unfolding the recursion, we get

$$\nabla_w V(w) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \sum_{a \in \mathcal{A}} \nabla_w \pi_w(a \mid \mathrm{x}_t) Q^{\pi_w}(\mathrm{x}_t, a) \mid \mathrm{x}_0 \sim \mu_0 \right]$$

$$\nabla_w \pi_w(a \mid x) = \pi_w(a \mid x) \nabla_w \log \pi_w(a \mid x)$$

Policy gradient

- Putting everything together

$$\nabla_w J^{\pi_w}(x) = \sum_{a \in \mathcal{A}} \nabla_w \pi_w(a \mid x) Q^{\pi_w}(x, a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_{\pi_w}(x' \mid x) \nabla_w J^{\pi_w}(x')$$

- This is a recursive equation in $\nabla_w J^{\pi_w}$

- Unfolding the recursion, we get

$$\nabla_w V(w) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \nabla_w \log \pi_w(\mathrm{a}_t \mid \mathrm{x}_t) Q^{\pi_w}(\mathrm{x}_t, \mathrm{a}_t) \mid \mathrm{x}_0 \sim \mu_0 \right]$$

Discounted sum of $\nabla_w \log \pi_w(a \mid x) Q^{\pi_w}(x, a)$ along a trajectory obtained with π_w

Policy gradient

- Putting everything together

$$\nabla_w J^{\pi_w}(x) = \sum_{a \in \mathcal{A}} \nabla_w \pi_w(a \mid x) Q^{\pi_w}(x, a) + \gamma \sum_{x' \in \mathcal{X}} \mathbf{P}_{\pi_w}(x' \mid x) \nabla_w J^{\pi_w}(x')$$

- This is a recursive equation in $\nabla_w J^{\pi_w}$

- Unfolding the recursion, we get

$$\nabla_w V(w) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \nabla_w \log \pi_w(\mathrm{a}_t \mid \mathrm{x}_t) Q^{\pi_w}(\mathrm{x}_t, \mathrm{a}_t) \mid \mathrm{x}_0 \sim \mu_0 \right]$$

We can sample this!

Proto-PG algorithm

- Given a trajectory obtained with policy π_{w_t}

$$\tau = \{x_t, a_t, c_t, x_{t+1}, a_{t+1}, c_{t+1}, \ldots, a_{t+N}, c_{t+N}, x_{t+N+1}\}$$

- Compute

$$\hat{\nabla}_w V(w_t) \approx \sum_{n=0}^{N} \gamma^n \nabla_w \log \pi_{w_t}(x_{t+n}, a_{t+n}) Q^{\pi_{w_t}}(x_{t+n}, a_{t+n})$$

Where does this come from?

- Update

$$w_{t+1} = w_t - \alpha_t \hat{\nabla}_w V(w_t)$$

REINFORCE algorithm

- Given a trajectory obtained with policy π_{w_t}

$$\tau = \{x_t, a_t, c_t, x_{t+1}, a_{t+1}, c_{t+1}, \ldots, a_{t+N}, c_{t+N}, x_{t+N+1}\}$$

- We approximate

$$Q^{\pi_{w_t}}(x_{t+n}, a_{t+n}) = \sum_{\tau=0}^{N-n} \gamma^\tau c_{t+n+\tau}$$

Monte Carlo estimate

REINFORCE algorithm

- Given a trajectory obtained with policy π_{w_t}

$$\tau = \{x_t, a_t, c_t, x_{t+1}, a_{t+1}, c_{t+1}, \ldots, a_{t+N}, c_{t+N}, x_{t+N+1}\}$$

- Compute

$$\hat{\nabla}_w V(w_t) \approx \sum_{n=0}^{N} \nabla_w \log \pi_{w_t}(x_{t+n}, a_{t+n}) \sum_{m=n}^{N} \gamma^m c_{t+m}$$

- Update

$$w_{t+1} = w_t - \alpha_t \hat{\nabla}_w V(w_t)$$

REINFORCE Algorithm

- The REINFORCE algorithm was the first policy gradient algorithm (1992)
- Gradient estimates make use of long trajectories
- Just like MC methods, REINFORCE suffers from large variance
- Every time the policy is updated, new trajectories must be obtained (poor data efficiency)

Addressing variance, part 1

- When computing

$$\nabla_w V(w) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \nabla_w \log \pi_w(\mathrm{a}_t \mid \mathrm{x}_t) Q^{\pi_w}(\mathrm{x}_t, \mathrm{a}_t) \mid \mathrm{x}_0 \sim \mu_0 \right]$$

we can add a baseline $b(\mathrm{x}_t)$, since

$$\mathbb{E}_\pi \left[\nabla_w \log \pi_w(\mathrm{a}_t \mid \mathrm{x}_t) b(\mathrm{x}_t) \right] = 0$$

Why?

Addressing variance, part 1

- The best baseline to use is J^{π_w}
- We get

$$\nabla_w V(w) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \nabla_w \log \pi_w(\mathrm{a}_t \mid \mathrm{x}_t) \left(Q^{\pi_w}(\mathrm{x}_t, \mathrm{a}_t) - J^{\pi_w}(\mathrm{x}_t) \right) \mid \mathrm{x}_0 \sim \mu_0 \right]$$

Advantage function

$$\mathrm{Adv}^{\pi_w}(\mathrm{x}_t, \mathrm{a}_t)$$

Addressing variance, part 1

- The best baseline to use is J^{π_w}
- We get

$$\nabla_w V(w) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \nabla_w \log \pi_w(\mathrm{a}_t \mid \mathrm{x}_t) \mathrm{Adv}^{\pi_w}(\mathrm{x}_t, \mathrm{a}_t) \mid \mathrm{x}_0 \sim \mu_0 \right]$$

Addressing variance, part 2

- When computing

$$\nabla_w V(w) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \nabla_w \log \pi_w(\mathrm{a}_t \mid \mathrm{x}_t) \mathrm{Adv}^{\pi_w}(\mathrm{x}_t, \mathrm{a}_t) \mid \mathrm{x}_0 \sim \mu_0 \right]$$

we can use low-variance estimates for $\mathrm{Adv}^{\pi_w}(\mathrm{x}_t, \mathrm{a}_t)$

$$\sum_{n=0}^{N} \gamma^n c_{t+n} - J^{\pi_w}(x_t)$$

$$Q^{\pi_w}(x_t, a_t) - \mathbb{E}_{\mathrm{a} \sim \pi_w}[Q^{\pi_w}(x_t, \mathrm{a})]$$

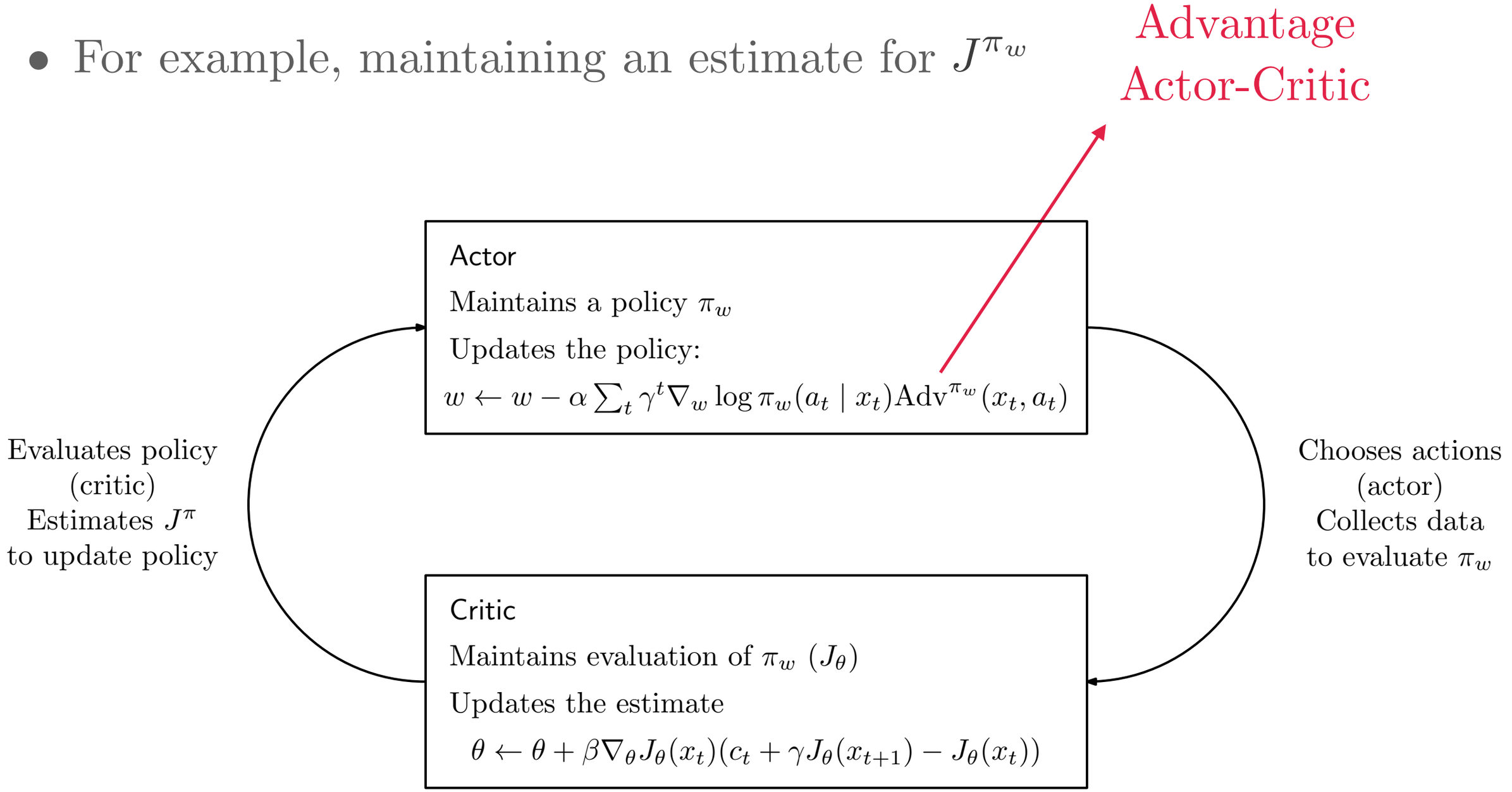
$$c_t + \gamma J^{\pi_w}(x_{t+1}) - J^{\pi_w}(x_t)$$

Addressing variance, part 2

- Previous approaches require keeping an estimate of J^{π_w} or Q^{π_w}
- We break the algorithm in two parts:
  - One part maintains/runs the policy (actor)
  - One part computes J^{π_w} or Q^{π_w} (critic)

Actor-critic architecture

- For example, maintaining an estimate for J^{π_w}



Actor-critic methods

- Many variations exist:
  - Critic using batch learning
  - Actor using adjusted/natural gradients
  - Multiple time scales
  - Synchronous/asynchronous updates
  - ...

Addressing data efficiency

- Can we reuse trajectories from previous policies to estimate Adv^{π_w}?
  - If the policy does not change a lot, we may...

Addressing data efficiency

- We have that

$$V(w) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_t \mid \mathrm{x}_0 \sim \mu_0\right]$$

and, for π_w and $\pi_{w'}$ close,

$$V(w') = V(w) + \mathbb{E}_{\pi_w}\left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi_{w'}(\mathrm{a}_t \mid \mathrm{x}_t)}{\pi_w(\mathrm{a}_t \mid \mathrm{x}_t)} \mathrm{Adv}^{\pi_w}(\mathrm{x}_t, \mathrm{a}_t) \mid \mathrm{x}_0 \sim \mu_0\right]$$

We can minimize this...

... by minimizing this

Addressing data efficiency

- When minimizing

$$L(w') = \mathbb{E}_{\pi_w}\left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi_{w'}(\mathrm{a}_t \mid \mathrm{x}_t)}{\pi_w(\mathrm{a}_t \mid \mathrm{x}_t)} \mathrm{Adv}^{\pi_w}(\mathrm{x}_t, \mathrm{a}_t) \mid \mathrm{x}_0 \sim \mu_0\right]$$

... we can use trajectories from π_w (better data efficiency)

- We can keep π_w and $\pi_{w'}$ close by bounding

$$r_t(w') = \frac{\pi_{w'}(a_t \mid x_t)}{\pi_w(a_t \mid x_t)}$$

A new policy gradient...

- Given a trajectory obtained with some policy π_{old}

$$\tau = \{x_t, a_t, c_t, x_{t+1}, a_{t+1}, c_{t+1}, \ldots, a_{t+N}, c_{t+N}, x_{t+N+1}\}$$

- We get a new policy gradient update

$$w_{t+1} = w_t - \alpha_t \nabla_w L(w_t)$$

with

$$\nabla_w L(w_t) = \sum_{n=0}^{N} \gamma^n \nabla_w \log \pi_w(a_{t+n} \mid x_{t+n}) r_{t+n}(w_t) \text{Adv}^{\pi_{\text{old}}}(x_{t+n}, a_{t+n})$$

How do we bound this?

Bounding $r_t(w)$

- To bound the value of $r_t(w)$, we replace

$$r_{t+n}(w_t)\text{Adv}^{\pi_{\text{old}}}(x_{t+n}, a_{t+n})$$

by

$$\min\{r_{t+n}(w_t)\text{Adv}^{\pi_{\text{old}}}(x_{t+n}, a_{t+n}), \text{clip}(r_{t+n}(w_t), 1-\epsilon, 1+\epsilon)\text{Adv}^{\pi_{\text{old}}}(x_{t+n}, a_{t+n})\}$$

We clip the value of
$r_{t+n}(w_t)$ to $[1-\epsilon, 1+\epsilon]$

Proximal Policy Optimization (PPO)

- The resulting algorithm is called **proximal policy optimization** (PPO)
- Is often used within an actor-critic architecture
- It is a state-of-the-art reinforcement learning algorithm
- Used in many robotic tasks