Learning to control

D.A.Forsyth, UIUC
Topics

- Vocabulary
- MDPs, Value Iteration, Policy Iteration
- Simple reinforcement learning ideas
  - and problems!
Markov Decision Process

Assumption: agent gets to observe the state

[Drawing from Sutton and Barto, Reinforcement Learning: An Introduction, 1998]
Model

• At time 0, environment samples initial state
  • agent is in that state
• Then for t=0 till done
  • agent chooses action
  • environment samples new state conditioned on action, old state
  • environment samples reward conditioned on action, old state, new state
  • agent gets that reward and moves into new state

• Policy
  • what action to take in each state
    • this could be stochastic
• Maximise total discounted reward
Examples

- Cleaning robot
- Walking robot
- Pole balancing
- Games: tetris, backgammon
- Server management
- Shortest path problems
- Model for animals, people
Markov Decision Process (S, A, T, R, H)

Given

- \( S \): set of states
- \( A \): set of actions
- \( T \): \( S \times A \times S \times \{0, 1, \ldots, H\} \rightarrow [0, 1] \), \( T_t(s,a,s') = P(s_{t+1} = s' | s_t = s, a_t = a) \)
- \( R \): \( S \times A \times S \times \{0, 1, \ldots, H\} \rightarrow \mathbb{R} \), \( R_t(s,a,s') = \text{reward for } (s_{t+1} = s', s_t = s, a_t = a) \)
- \( H \): horizon over which the agent will act

Goal:

- Find \( \pi : S \times \{0, 1, \ldots, H\} \rightarrow A \) that maximizes expected sum of rewards, i.e.,

\[
\pi^* = \arg \max_{\pi} \mathbb{E}\left[ \sum_{t=0}^{H} R_t(S_t, A_t, S_{t+1}) | \pi \right]
\]

This is usually discounted by gamma
Canonical Example: Grid World

- The agent lives in a grid
- Walls block the agent’s path
- The agent’s actions do not always go as planned:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Big rewards come at the end

And this is true for the other three; 80% of the time you go where you intended, 10% at right angles one way 10% the other
Solving MDPs

- In an MDP, we want an optimal policy $\pi^*$: $S \times 0:H \rightarrow A$
  - A policy $\pi$ gives an action for each state for each time

- An optimal policy maximizes expected sum of rewards

- Contrast: In deterministic, want an optimal plan, or sequence of actions, from start to a goal
Outline

- Optimal Control
  - given an MDP \((S, A, T, R, \gamma, H)\)
  - find the optimal policy \(\pi^*\)

- Exact Methods:
  - Value Iteration
  - Policy Iteration
Value of a state

- value of a state
  - expected reward of proceeding optimally from that state
  - this satisfies a recurrence relation
  - if you know this, optimal policy is easy
    - take action with with best expected reward

Expected reward of action \( a \) in state \( s \)

\[
\sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right]
\]

Value of \( s' \)
the new state
The Bellman Equation

- Value of a state
  - expected reward of proceeding optimally from that state
  - this satisfies a recurrence relation

- The Bellman Equation

\[ V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \]

What you get by acting optimally from \( s' \)

Acting optimally from \( s \)
Value iteration

- Idea:
  - value of a state = expected reward of proceeding optimally from that state
  - if we knew the value of each state, choosing an action is easy
    - take the one with the best expected yield
- Idea:
  - we could estimate the value of a state
    - set the value of every state to something
    - now for a given state, compute the expected value of best action
      - replace value with that and continue
Value Iteration

Algorithm:

- Start with $V_0^*(s) = 0$ for all $s$.
- For $i=1, \ldots, H$
  
  Given $V_i^*$, calculate for all states $s \in S$:

  \[ V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_i^*(s')] \]

- This is called a value update or Bellman update/back-up

- $V_i^*(s)$ = the expected sum of rewards accumulated when starting from state $s$ and acting optimally for a horizon of $i$ steps
Value Iteration in Gridworld

noise = 0.2, \gamma = 0.9, two terminal states with R = +1 and -1
Value Iteration in Gridworld

noise = 0.2, $\gamma = 0.9$, two terminal states with $R = +1$ and $-1$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

VALUES AFTER 2 ITERATIONS
Value Iteration in Gridworld

noise = 0.2, $\gamma$ = 0.9, two terminal states with R = +1 and -1

<table>
<thead>
<tr>
<th></th>
<th>0.00</th>
<th>0.52</th>
<th>0.78</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.43</td>
<td></td>
<td>-1.00</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>

VALUES AFTER 3 ITERATIONS
Value Iteration in Gridworld

noise = 0.2, $\gamma$ = 0.9, two terminal states with R = +1 and -1

VALUES AFTER 4 ITERATIONS
Value Iteration in Gridworld

noise = 0.2, $\gamma = 0.9$, two terminal states with $R = +1$ and $-1$
Value Iteration in Gridworld
noise = 0.2, \( \gamma = 0.9 \), two terminal states with \( R = +1 \) and \(-1\)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.64</td>
<td></td>
<td>0.74</td>
<td>0.85</td>
</tr>
<tr>
<td>\text{gray}</td>
<td></td>
<td>\text{gray}</td>
<td></td>
</tr>
<tr>
<td>0.57</td>
<td>0.57</td>
<td></td>
<td>-1.00</td>
</tr>
<tr>
<td>0.49</td>
<td>0.43</td>
<td>0.48</td>
<td>0.28</td>
</tr>
</tbody>
</table>

VALUES AFTER 100 ITERATIONS
Value Iteration in Gridworld

noise = 0.2, \( \gamma = 0.9 \), two terminal states with \( R = +1 \) and -1

VALUES AFTER 1000 ITERATIONS
Exercise 1: Effect of discount, noise

(a) Prefer the close exit (+1), risking the cliff (-10)
(b) Prefer the close exit (+1), but avoiding the cliff (-10)
(c) Prefer the distant exit (+10), risking the cliff (-10)
(d) Prefer the distant exit (+10), avoiding the cliff (-10)

(1) $\gamma = 0.1$, noise = 0.5
(2) $\gamma = 0.99$, noise = 0
(3) $\gamma = 0.99$, noise = 0.5
(4) $\gamma = 0.1$, noise = 0
Exercise 1 Solution

(a) Prefer close exit (+1), risking the cliff (-10) --- $\gamma = 0.1$, noise = 0
Exercise 1 Solution

<table>
<thead>
<tr>
<th></th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.00</th>
<th>0.03</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td></td>
<td>0.05</td>
<td>0.03</td>
<td>0.51</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td></td>
<td>1.00</td>
<td></td>
<td>10.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.05</td>
<td>0.01</td>
<td>0.51</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.00</td>
<td>10.00</td>
<td>10.00</td>
<td>10.00</td>
<td>10.00</td>
<td></td>
</tr>
</tbody>
</table>

(b) Prefer close exit (+1), avoiding the cliff (-10) -- $\gamma = 0.1$, noise = 0.5
Exercise 1 Solution

(c) Prefer distant exit (+1), risking the cliff (-10) -- $\gamma = 0.99$, noise = 0
Exercise 1 Solution

(d) Prefer distant exit (+1), avoid the cliff (-10) -- $\gamma = 0.99$, noise = 0.5
Value Iteration Convergence

**Theorem.** Value iteration converges. At convergence, we have found the optimal value function $V^*$ for the discounted infinite horizon problem, which satisfies the Bellman equations

$$
\forall s \in S: \quad V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right]
$$

- Now we know how to act for infinite horizon with discounted rewards!
  - Run value iteration till convergence.
  - This produces $V^*$, which in turn tells us how to act, namely following:

$$
\pi^*(s) = \arg \max_{a \in A} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]
$$

- Note: the infinite horizon optimal policy is stationary, i.e., the optimal action at a state $s$ is the same action at all times. (Efficient to store!)
Policy Evaluation

- Recall value iteration iterates:

\[ V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_i^*(s')] \]

- Policy evaluation:

\[ V_{i+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_i^\pi(s')] \]

- At convergence:

\[ \forall s \quad V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi(s')] \]
Exercise 2

Consider a stochastic policy $\mu(a|s)$, where $\mu(a|s)$ is the probability of taking action $a$ when in state $s$. Which of the following is the correct value iteration update to perform policy evaluation for this stochastic policy?

1. $V_{i+1}^\mu(s) \leftarrow \max_a \sum_{s'} T(s, a, s')(R(s, a, s') + \gamma V_i^\mu(s'))$

2. $V_{i+1}^\mu(s) \leftarrow \sum_{s'} \sum_a \mu(a|s) T(s, a, s')(R(s, a, s') + \gamma V_i^\mu(s'))$

3. $V_{i+1}^\mu(s) \leftarrow \sum_a \mu(a|s) \max_{s'} T'(s, a, s')(R(s, a, s') + \gamma V_i^\mu(s'))$
Policy Iteration

Alternative approach:

- **Step 1: Policy evaluation**: calculate utilities for some fixed policy (not optimal utilities!) until convergence
- **Step 2: Policy improvement**: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
- Repeat steps until policy converges

This is **policy iteration**

- It’s still optimal!
- Can converge faster under some conditions
Policy Evaluation Revisited

- **Idea 1:** modify Bellman updates

\[ V_0^{\pi}(s) = 0 \]
\[ V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^{\pi}(s')] \]

- **Idea 2:** it’s just a linear system, solve with Matlab (or whatever), variables: \( V^\pi(s) \), constants: \( T, R \)

\[ \forall s \quad V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')] \]
Policy Iteration Guarantees

Policy Iteration iterates over:

- Policy evaluation: with fixed current policy $\pi$, find values with simplified Bellman updates:
  - Iterate until values converge
  $$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[ R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s') \right]$$

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead
  $$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i^{\pi_k}(s') \right]$$

**Theorem.** Policy iteration is guaranteed to converge and at convergence, the current policy and its value function are the optimal policy and the optimal value function.

Proof sketch:
(1) **Guarantee to converge:** In every step the policy improves. This means that a given policy can be encountered at most once. This means that after we have iterated as many times as there are different policies, i.e., $(\text{number actions})^{(\text{number states})}$, we must be done and hence have converged.

(2) **Optimal at convergence:** by definition of convergence, at convergence $\pi_{k+1}(s) = \pi_k(s)$ for all states $s$. This means $\forall s \ V^{\pi_k}(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i^{\pi_k}(s') \right]$
Hence $V^{\pi_k}$ satisfies the Bellman equation, which means $V^{\pi_k}$ is equal to the optimal value function $V^\star$. 
But it’s not really all over…

• What if:
  • there are lots of states?
  • we don’t know T?
  • we don’t know R?
Policy iteration

• Idea:
  • evaluate some policy
  • then make it better
The optimal policy $\pi^*$

We want to find optimal policy $\pi^*$ that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability...)?
Maximize the **expected sum of rewards**!

Formally: $\pi^* = \arg\max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi \right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$
Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths) \( s_0, a_0, r_0, s_1, a_1, r_1, \ldots \)

How good is a state?
The **value function** at state \( s \), is the expected cumulative reward from following the policy from state \( s \):

\[
V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi \right]
\]

How good is a state-action pair?
The **Q-value function** at state \( s \) and action \( a \), is the expected cumulative reward from taking action \( a \) in state \( s \) and then following the policy:

\[
Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]
\]
Bellman equation

The optimal Q-value function $Q^*$ is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s, a) = \max_\pi \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]$$

$Q^*$ satisfies the following Bellman equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

**Intuition:** if the optimal state-action values for the next time-step $Q^*(s', a')$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s', a')$

The optimal policy $\pi^*$ corresponds to taking the best action in any state as specified by $Q^*$
Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

\[ Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right] \]

\( Q_i \) will converge to \( Q^* \) as \( i \to \infty \)

What’s the problem with this?
Not scalable. Must compute \( Q(s, a) \) for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Solution: use a function approximator to estimate \( Q(s, a) \). E.g. a neural network!
Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

\[ Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right] \]

Forward Pass

Loss function: \( L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[ (y_i - Q(s, a; \theta_i))^2 \right] \)

where \( y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \mid s, a \right] \)

Backward Pass

Gradient update (with respect to Q-function parameters \( \theta \)):

\[ \nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right] \nabla_{\theta_i} Q(s, a; \theta_i) \]
Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:
- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using experience replay
- Continually update a replay memory table of transitions \((s_t, a_t, r_t, s_{t+1})\) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

Each transition can also contribute to multiple weight updates => greater data efficiency
Policy Gradients

What is a problem with Q-learning? The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

But the policy can be much simpler: just close your hand
Can we learn a policy directly, e.g. finding the best policy from a collection of policies?
Policy Gradients

Formally, let’s define a class of parametrized policies: \( \Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\} \)

For each policy, define its value:
\[
J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]
\]

We want to find the optimal policy \( \theta^* = \arg \max_\theta J(\theta) \)

How can we do this? 
Gradient ascent on policy parameters!
REINFORCE algorithm

Mathematically, we can write:

\[ J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \]

\[ = \int_{\tau} r(\tau)p(\tau; \theta) d\tau \]

Where \( r(\tau) \) is the reward of a trajectory \( \tau = (s_0, a_0, r_0, s_1, \ldots) \)
**REINFORCE algorithm**

Expected reward:  
\[ J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \]
\[ = \int \tau r(\tau)p(\tau; \theta) d\tau \]

Now let's differentiate this:  
\[ \nabla_\theta J(\theta) = \int \nabla_\theta p(\tau; \theta) r(\tau) d\tau \]

However, we can use a nice trick:  
\[ \nabla_\theta p(\tau; \theta) = p(\tau; \theta) \frac{\nabla_\theta p(\tau; \theta)}{p(\tau; \theta)} = p(\tau; \theta) \nabla_\theta \log p(\tau; \theta) \]

If we inject this back:
\[ \nabla_\theta J(\theta) = \int \nabla_\theta \log p(\tau; \theta) p(\tau; \theta) d\tau \]
\[ = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_\theta \log p(\tau; \theta)] \]

Intractable! Gradient of an expectation is problematic when p depends on \( \theta \)

Can estimate with Monte Carlo sampling
REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau (r(\tau) \nabla_\theta \log p(\tau; \theta)) p(\tau; \theta) d\tau$$

$$= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_\theta \log p(\tau; \theta)]$$

Can we compute those quantities without knowing the transition probabilities?

We have: $$p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Thus: $$\log p(\tau; \theta) = \sum_{t \geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$$

And when differentiating: $$\nabla_\theta \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Doesn't depend on transition probabilities!

Therefore when sampling a trajectory \(\tau\), we can estimate \(J(\theta)\) with

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$
Intuition

Gradient estimator: \[ \nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t) \]

Interpretation:
- If \( r(\tau) \) is high, push up the probabilities of the actions seen
- If \( r(\tau) \) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. **But in expectation, it averages out!**

However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?
Variance reduction

Gradient estimator:

\[ \nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t) \]

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

\[ \nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t | s_t) \]

Second idea: Use discount factor \( \gamma \) to ignore delayed effects

\[ \nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t | s_t) \]
Variance reduction: Baseline

**Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

**What is important then?** Whether a reward is better or worse than what you expect to get.

**Idea:** Introduce a baseline function dependent on the state. Concretely, estimator is now:

\[ \nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \]
How to choose the baseline?

\[ \nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t) \]

A simple baseline: constant moving average of rewards experienced so far from all trajectories

Variance reduction techniques seen so far are typically used in “Vanilla REINFORCE”
How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the expected value of what we should get from that state.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action $a_t$ in a state $s_t$ if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. On the contrary, we are unhappy with an action if it’s small.

Using this, we get the estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi^\theta}(s_t, a_t) - V^{\pi^\theta}(s_t)) \nabla_\theta \log \pi^\theta(a_t|s_t)$
Actor-Critic Algorithm

**Problem:** we don’t know Q and V. Can we learn them?

**Yes,** using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust.
- Also alleviates the task of the critic as it only has to learn the values of (state, action) pairs generated by the policy.
- Can also incorporate Q-learning tricks e.g. experience replay.
- **Remark:** we can define by the **advantage function** how much an action was better than expected

\[ A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \]
Why so many RL algorithms?

- Different tradeoffs
  - Sample efficiency
  - Stability & ease of use
- Different assumptions
  - Stochastic or deterministic?
  - Continuous or discrete?
  - Episodic or infinite horizon?
- Different things are easy or hard in different settings
  - Easier to represent the policy?
  - Easier to represent the model?
Blog post entitled: “Why deep reinforcement learning doesn’t work”

https://www.alexirpan.com/2018/02/14/rl-hard.html