Value Functions 2 and Deep Q-Learning

Lecture 3

Peter Henderson

COS 435 / ECE 433

Thanks to helpful slides/notes by Ben Van Roy, Emma Brunskill, Ben Eysenbach, and Csaba Szepesvári.

Today's Agenda

1. Logistics: Attendance check quiz, look at the blackboard for the attendance check code (go to the Canvas quiz).
2. Logistics: Projects, we will post team matching and project ideas next week. Start forming teams. We will ask folks to submit a lightweight project proposal and form teams by March 13th.
3. Assignment 1 will be posted tonight, due 2 weeks from today.
4. Review: Value Functions and Bellman Equations
5. Exercise: Computing Value Functions
6. Review: Learning Value Functions
7. Discussion: Limitations of DQN

Review: Value Functions

Review: Value Functions and Bellman Equations

- What is $Q(s, a)$ and $V(s)$? How do these depend on the policy?

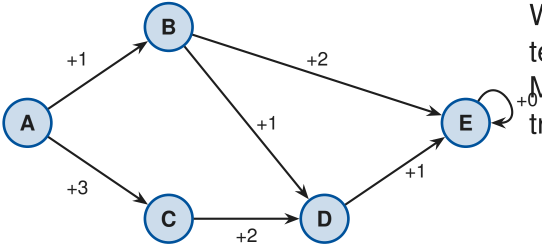
$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right]$$
$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s \right]$$

- Bellman equations:

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{p(s'|s,a)\pi(a'|s')}[Q^\pi(s', a')] \qquad \text{(expectation eq.)}$$
$$Q^*(s, a) = r(s, a) + \gamma \mathbb{E}_{p(s'|s,a)}[\max_{a'} Q^*(s', a')] \qquad \text{(optimality eq.)}$$

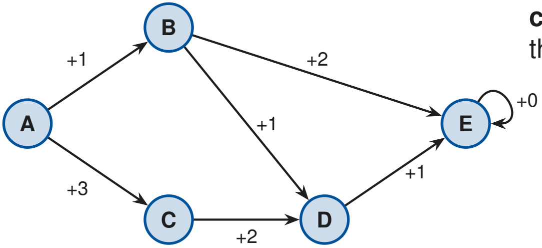
Exercise: Computing Value Functions

Exercise: Computing Value Functions (MDP 1)



What are the discounted values (leave in terms of discount factor γ) for each state in MDP 1 assuming no actions and uniform transition probabilities?

Exercise: With Actions (MDP 1)

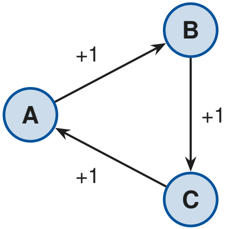


Now assume that actions allow the agent to **choose** between the outgoing edges. What are the state-action values?

1. What is $Q(B, \text{down})$?
2. What is $Q(B, \text{right})$?
3. What is $Q(A, \text{right})$?

   (Hint: requires knowing actions for B!)

Exercise: Circle MDP (MDP 2)



What are the discounted values (in terms of γ) for each state in MDP 2?

Review: Learning Value Functions

Learning Value Functions from Data

So far: value iteration and policy iteration require a **model** (T, R). But in practice, we often don't have a model.

Today's methods learn value functions directly from data (transitions $\{(s, a, r, s')\}$). These are **model-free** methods.

Key approaches:

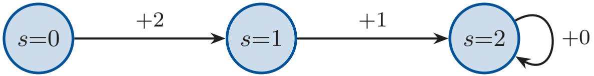
1. **Monte Carlo**: Use full trajectory returns (unbiased, high variance)
2. **SARSA**: Temporal difference, on-policy
3. **Expected SARSA**: Lower variance, can be off-policy
4. **Q-Learning**: TD, off-policy, learns Q^* directly

Monte Carlo Estimation

We are given as input trajectory tuples $\{(s_0, a_0, r_0, s_1, a_1, r_1, \ldots)\}$ and want to fill in the entries in our Q value table.

The **Monte Carlo** approach: look at the state-action pairs that you have visited, and see what the future returns were afterwards.

As an example, say you had the trajectory below, where there is a single action $a = 0$:



We do a full rollout from $s=0$ and collect the trajectory:

	s_0	a_0	r_0	s_1	a_1	r_1	s_2	$\cdots$
τ_1	0	0	2	1	0	1	2	$\cdots$

Exercise: Monte Carlo Q-Value Estimation

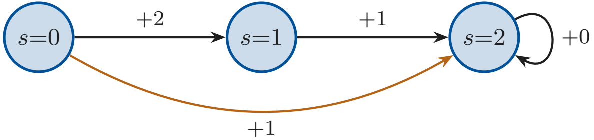
To compute $Q(s{=}0, a{=}0)$, we simply count up the future rewards after that state-action pair:

$$\gamma^0 \cdot 2 + \gamma^1 \cdot 1 + \gamma^2 \cdot 0 + \gamma^3 \cdot 0 + \cdots$$

Questions: What are $Q(s{=}0, a{=}0)$, $Q(s{=}1, a{=}0)$, and $Q(s{=}2, a{=}0)$?

MC Estimation with Stochastic Dynamics

Now say we have **two trajectories**. They both start with the same $(s{=}0, a{=}0)$ pair, but because the dynamics are stochastic the next state can be different:



Sampled trajectories:

	s_0	a_0	r_0	s_1	a_1	r_1	s_2	a_2	r_2	$\cdots$
τ_1	0	0	2	1	0	1	2	0	0	$\cdots$
τ_2	0	0	1	2	0	0	2	0	0	$\cdots$

Exercise: MC with Stochastic Dynamics

MC approach: Average the returns across trajectories for each state-action pair.

Questions: What are $Q(s{=}2, a{=}0)$, $Q(s{=}1, a{=}0)$, and $Q(s{=}0, a{=}0)$?

SARSA: On-Policy TD Control

SARSA estimates Q^π — the Q-values of the behavior policy (on policy).
Given transition (s, a, r, s', a') where $a' \sim \pi(\cdot|s')$:

SARSA Update

$$Q(s,a) \leftarrow (1-\alpha)\,Q(s,a) + \alpha\,(r(s,a) + \gamma Q(s',a'))$$

Name: **S**tate, **A**ction, **R**eward, **S**tate, **A**ction — the quintuple used in each update.
Gradient descent view: Prediction is $Q(s,a)$, target is $y = r(s,a) + \gamma Q(s',a')$:

$$\mathcal{L} = \frac{1}{2}(Q(s,a) - y)^2 \qquad \Rightarrow \qquad Q(s,a) \leftarrow Q(s,a) - \alpha(Q(s,a) - y)$$

Properties: On-policy (learns Q^π, not Q^*), single sample of a'.

Expected SARSA: Lower Variance, Off-Policy Capable

Idea: Instead of sampling a', take the expectation over next actions.

Expected SARSA Update

$$Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + \alpha \left(r(s,a) + \gamma \sum_{a'} \pi(a'|s') Q(s',a') \right)$$

Two key advantages over SARSA:

1. **Lower variance:** No randomness from sampling a' — we average over all actions
2. **Off-policy capable:** π in the expectation can differ from the data-collecting policy — we can estimate values of one policy using data from another

Q-Learning: Learning Q^* Directly

Q-learning is off-policy: it estimates Q^* even when transitions come from a suboptimal policy.

Q-Learning Update

$$Q(s,a) \leftarrow (1-\alpha)\,Q(s,a) + \alpha\left(r(s,a) + \gamma \max_{a'} Q(s',a')\right)$$

Key properties:

- **Input:** transitions $\{(s,a,r,s')\}$ from any behavior policy
- **Output:** $Q^*(s,a)$, the optimal value for each state-action pair
- **Off-policy:** will converge to Q^* even if data is from a suboptimal policy
- Uses $\max$ (Bellman optimality) instead of $\sum_{a'} \pi(\cdot)$ (expectation)

Q-Learning Algorithm

Q-Learning with ϵ-Greedy Exploration

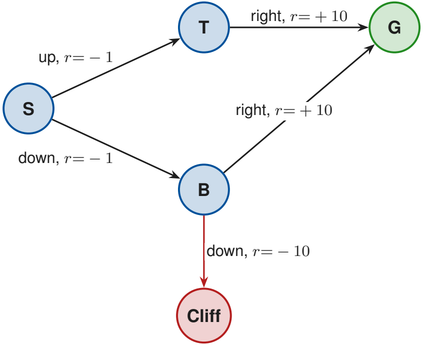
Initialize: $Q(s, a) = 0$ for all states and actions.
While not converged **do**:

1. $s \leftarrow$ ENV.RESET()
2. **For** $t = 1, \ldots, T$ **do**:
   - $a = \arg\max_a Q(s, a)$ with probability 1 - ϵ and random action with probability ϵ
   - Take action a, observe r, s'
   - $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
   - $s \leftarrow s'$

Return $Q(s, a)$.

Exercise: Cliff Walk

Consider the following cliff walk MDP. All dynamics are **deterministic**. There are two paths from S to G: a **safe path** through T, or a **risky path** through B (near the cliff).



Exercise: Comparing Update Rules

Given: $\alpha = 0.5$, $\gamma = 1$, ε-greedy with $\varepsilon = 0.5$ (2 actions $\Rightarrow$ greedy w.p. 0.75, other w.p. 0.25).

Current Q-values:

State	Action 1	Action 2
$Q(S, \cdot)$	up: 0	down: 2
$Q(B, \cdot)$	right: 5	down: 1

Observed transition: $(s{=}S,\ a{=}\text{down},\ r{=}-1,\ s'{=}B)$.
The agent then picks $a'{=}$down (exploratory action) at state B — it falls off the cliff!

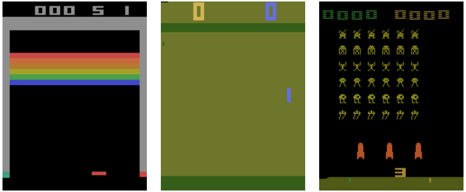
Compute the updated $Q(S, \textbf{down})$ under each method:

1. **SARSA**
2. **Expected SARSA**
3. **Q-Learning**

Break - 15 minutes

From Tabular to Deep Q-Learning

Discussion: Why Does Tabular Fail Here?



Atari 2600: Breakout, Pong, Space Invaders.

Discuss with your neighbor:

1. Why can't we use tabular Q-learning for Atari games?
2. What could we do instead?

Take 2 minutes to discuss.

Discussion: Designing a Neural Network for Q-Learning

Suppose we want to use a neural network for Q-learning on Atari.

Discuss with your neighbor:

1. What would the **input** to the neural network be?
2. What would the **output** be?
3. What would the **loss function** look like?

Take 2 minutes to discuss.

Gradient Descent Interpretation of Q-Learning

Key insight: Q-learning can be viewed as stochastic gradient descent.

The prediction is $Q(s,a)$, the label/target is $y = r(s,a) + \gamma \max_{a'} Q(s',a')$:

Loss Function

$$\mathcal{L} = \frac{1}{2}\left(Q(s,a) - y\right)^2 \qquad \text{where } y = r(s,a) + \gamma \max_{a'} Q(s',a')$$

Taking the derivative:

$$\frac{d\mathcal{L}}{dQ(s,a)} = Q(s,a) - y$$

$$Q(s,a) - \alpha \frac{d\mathcal{L}}{dQ(s,a)} = (1-\alpha)\, Q(s,a) + \alpha\, y$$

Note: the target y depends on Q but is treated as a constant.

From Tables to Neural Networks

Tabular Q-learning stores $Q(s,a)$ as a big table of size $|\mathcal{S}| \times |\mathcal{A}|$.

Deep Q-learning: Replace the table with a neural network $Q_\theta(s,a)$:

- We want $Q_\theta(s,a) = Q^*(s,a)$
- Instead of learning table entries, we learn **parameters** θ
- Architecture: state is input, output is $Q(s,a)$ for all actions a

Trained via the same loss, using θ_i to denote weights at iteration i:

$$\theta_{i+1} \leftarrow \arg\min_{\theta_{i+1}} \frac{1}{2} \Bigg(\underbrace{Q_{\theta_{i+1}}(s,a)}_{\text{prediction}} - \underbrace{\Big(r(s,a) + \gamma \max_{a'} Q_{\theta_i}(s',a') \Big)}_{\text{target / label}} \Bigg)^2$$

Important: We only update Q at the current time step — the target uses the *old* Q.

The Deadly Triad

Combining these three things can cause divergence:

1. **Function approximation** — neural net instead of table
2. **Bootstrapping** — target depends on current Q estimate
3. **Off-policy learning** — data from different policy than we're evaluating

Sutton & Barto (2018, Ch. 11.10)

"The potential for off-policy learning remains tantalizing, the best way to achieve it still a mystery."

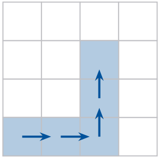
DQN (Mnih et al., 2015) introduced two key tricks to tame this instability:

- **Experience replay**
- **Target networks**

DQN and Neural Network Function Approximation

Problem: Consecutive transitions are highly correlated.

Example: An agent navigating a grid collects transitions along its path:



Training batch (consecutive):

$(s_1, \rightarrow, r_1, s_2)$
$(s_2, \rightarrow, r_2, s_3)$
$(s_3, \uparrow, r_3, s_4)$
$(s_4, \uparrow, r_4, s_5)$

All from the same small region and correlated with one another!

Discuss with your neighbor: Why is this a problem? What are the consequences? Think back to your machine learning class and assumptions for stochastic gradient descent and neural networks.

DQN Experience Replay

Problem: Consecutive transitions are highly correlated $\Rightarrow$ unstable SGD which assumes i.i.d. data.
Solution: Store transitions in a **replay buffer** $\mathcal{D}$ and sample random mini-batches.

Experience Replay

1. Store each transition (s, a, r, s') in buffer $\mathcal{D}$ (circular, fixed size)
2. Sample random mini-batch $\{(s_i, a_i, r_i, s'_i)\} \sim \mathcal{D}$
3. Compute gradient on mini-batch and update θ

Benefits:

- **Breaks correlations:** Random sampling decorrelates training data
- **Data efficiency:** Each transition reused in multiple updates
- **Stability:** Smooths over changes in data distribution as policy changes

DQN: Target Networks

Problem: The target $r + \gamma \max_{a'} Q_\theta(s', a')$ changes every time we update θ.

$\Rightarrow$ We're chasing a moving target — makes optimization unstable.

Solution: Use a separate **target network** Q_{θ^-} with frozen weights.

DQN Target

$y = r + \gamma \max_{a'} Q_{\theta^-}(s', a')$. Update $\theta^- \leftarrow \theta$ every C steps, or soft: $\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-$.

DQN Pseudocode

Deep Q-Network Algorithm

Initialize: replay buffer $\mathcal{D}$, Q_θ with random weights, $\theta^- \leftarrow \theta$
For each episode:

1. Initialize state s_0 (stack of 4 frames)
2. **For each step t:**
   - Select a_t via ϵ-greedy w.r.t. Q_θ
   - Execute a_t, observe r_t, s_{t+1}
   - Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
   - Sample mini-batch $\{(s_i, a_i, r_i, s'_i)\}$ from $\mathcal{D}$
   - Compute targets: $y_i = r_i + \gamma \max_{a'} Q_{\theta^-}(s'_i, a')$
   - Update θ by SGD on $\sum_i (Q_\theta(s_i, a_i) - y_i)^2$
   - Every C steps: $\theta^- \leftarrow \theta$

DQN: Architecture and Results

Architecture:

- Input: stack of 4 raw frames (pixels)
- 3 conv layers $\rightarrow$ 2 FC layers
- Output: $Q(s, a)$ for all 18 actions
- Reward: change in game score
- Same architecture across all 49 games!

Key results (Mnih et al., 2015):

- Superhuman on many atari games
- Learned from raw pixels

Discussion: Limitations of DQN

Quick Discussion:

1. What sorts of problems do you think might still exist in DQN?
2. What sorts of improvements do you think we can make?

Take 2–3 minutes to brainstorm with your neighbor.

DQN Improvements: Toward Rainbow

Double DQN: Fixing Overestimation Bias

Problem: $\max_{a'} Q(s', a')$ overestimates the true value because noise in Q gets amplified by the max operator (also recall bias-variance tradeoffs from Kearns and Singh, amplification of bias).

Double Q-Learning (van Hasselt, 2010): Decouple action *selection* from action *evaluation* using two different networks.

Double Q-Learning: Full Algorithm

Double Q-Learning (van Hasselt, 2010)

Initialize: $Q^A(s,a)$, $Q^B(s,a)$ for all s, a; initial state s
Repeat:

1. Choose a based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$; observe r, s'
2. Choose (e.g. random) either **UPDATE(A)** or **UPDATE(B)**:
   - **If UPDATE(A):** $a^* = \arg\max_{a'} Q^A(s', a')$
     $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\big[r + \gamma Q^B(s', a^*) - Q^A(s,a)\big]$
   - **Else if UPDATE(B):** $b^* = \arg\max_{a'} Q^B(s', a')$
     $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)\big[r + \gamma Q^A(s', b^*) - Q^B(s,a)\big]$
3. $s \leftarrow s'$

Until end.

Action *selection* uses one network; action *evaluation* uses the other $\Rightarrow$ reduces overestimation bias.

Prioritized Experience Replay

Problem: Uniform sampling from replay buffer wastes time on “easy” transitions.

Idea (Schaul et al., 2016): Sample transitions proportional to their **TD error**:

$$p_i \propto |\delta_i|^\alpha \quad \text{where} \quad \delta_i = r_i + \gamma \max_{a'} Q_{\theta^-}(s'_i, a') - Q_\theta(s_i, a_i)$$

Transitions where the agent is “most wrong” get replayed more often.

Importance sampling correction: To compensate for non-uniform sampling:

$$w_i = \left(\frac{1}{N \cdot p_i}\right)^\beta$$

Anneal $\beta \to 1$ over training to remove bias.

Prioritized Experience Replay

What idea from value iteration speedups does this remind you of?

Double DQN with Prioritized Experience Replay

Algorithm: Double DQN with Proportional Prioritization (Schaul et al., 2016)

Input: minibatch k, step-size η, replay period K, size N, exponents α, β, budget T
Initialize: replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$. Observe s_0, choose $a_0 \sim \pi_\theta(s_0)$.
For $t = 1$ **to** T:

1. Observe s_t, r_t, γ_t
2. Store $(s_{t-1}, a_{t-1}, r_t, \gamma_t, s_t)$ in $\mathcal{H}$ with priority $p_t = \max_{i<t} p_i$
3. **If** $t \equiv 0 \pmod{K}$:
   - **For** $j = 1$ **to** k:
     - Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
     - $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
     - $\delta_j = r_j + \gamma_j Q_{\theta^-}(s_j, a^*) - Q_\theta(s_{j-1}, a_{j-1})$, $a^* = \arg\max_a Q_\theta(s_j, a)$
     - $p_j \leftarrow |\delta_j|; \quad \Delta \leftarrow \Delta + w_j \delta_j \nabla_\theta Q_\theta(s_{j-1}, a_{j-1})$
   - $\theta \leftarrow \theta + \eta\Delta$, $\Delta \leftarrow 0$; periodically $\theta^- \leftarrow \theta$
4. Choose $a_t \sim \pi_\theta(s_t)$

Dueling Networks

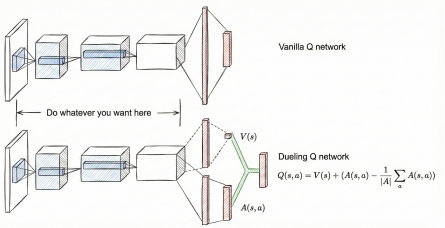
Observation: In many states, the *value of being in the state* matters more than which action you take. Can create a bit more stability by predicting both.

Dueling DQN (Wang et al., 2016): Decompose Q into value and advantage:

Dueling Architecture

$$Q_\theta(s,a) = V_\theta(s) + \left(A_\theta(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A_\theta(s,a')\right)$$

- $V_\theta(s)$: how good is this state?
- $A_\theta(s,a)$: how much better is a than average?



Top: standard DQN. Bottom: dueling architecture with separate V and A streams. (Wang et al., 2016)

Distributional DQN and Classification-Based Values

Idea: Standard DQN learns the *mean* return via MSE regression. **Distributional RL** learns the full *distribution* (e.g. C51, QR-DQN).

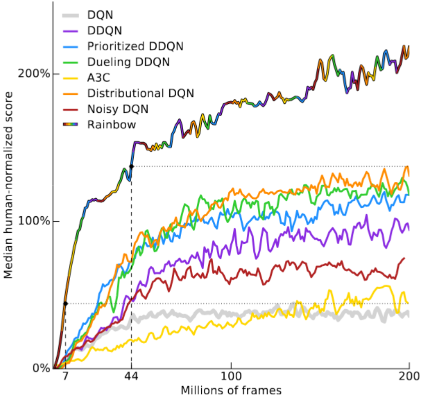
Why classification? Discretize the value range into quantiles and predict the distribution. More scalable, robust to noisy targets and non-stationarity, reduces overfitting; SOTA on Atari, multi-task RL, robotics, and more.

Rainbow: Combining All the Improvements

Rainbow (Hessel et al., 2018): Combine **six** DQN improvements:

1. **Double DQN** — reduce overestimation bias
2. **Prioritized Replay** — focus on surprising transitions
3. **Dueling Networks** — separate state value from action advantage
4. **Multi-step Return Targets** — Predict a few steps ahead.
5. **Distributional RL** — learn the full *distribution* of returns, not just the mean
6. **Noisy Networks** — learned exploration via stochastic network layers (replaces ϵ-greedy)

Rainbow: Learning Curves



Rainbow achieves >200% median human-normalized score in 44M frames. (Hessel et al., 2018)

Rainbow: Detailed Results

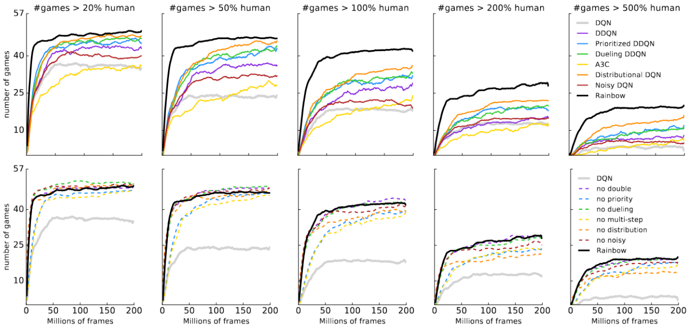


Figure 2: Each plot shows, for several agents, the number of games where they have achieved at least a given fraction of human performance, as a function of time. From left to right we consider the 20%, 50%, 100%, 200% and 500% thresholds. On the first row we compare Rainbow to the baselines. On the second row we compare Rainbow to its ablations.

Games achieving human performance thresholds. Top: Rainbow vs. baselines. Bottom: Ablations. (Hessel et al., 2018)

Under-Reported Trick: Classification Instead of Regression

Problem: MSE regression is unstable with noisy, non-stationary TD targets.
Solution: Discretize values into bins, predict a categorical distribution, use cross-entropy but add the bins to get the full value. Works for both value and action-value learning.

How It Works

1. Discretize $[V_{\min}, V_{\max}]$ into m bins $z_1, \ldots, z_m$
2. Network $\rightarrow$ softmax $\rightarrow$ probs $\hat{p}_i$ over bins
3. Recover: $Q = \sum_i \hat{p}_i \cdot z_i$

Why it helps:

- Handles noisy targets better
- Scales to larger networks
- Bounded gradients

Farebrother et al., “Stop Regressing: Training Value Functions via Classification,” 2024.

Discussion

Discussion: Readings?

Quick Discussion:

1. What was one thing you liked, one thing you didn't like, and one thing you're unsure about with respect to the readings?

Take 5 minutes to brainstorm with your neighbor.

(Almost) Q Learning, but for Continuous Actions

DDPG: Deep Deterministic Policy Gradient

Problem: Q-learning is not suitable for continuous action spaces. How can we choose $max_{a'} Q(s', a')$ if your actions are continuous? What do you think?

DDPG: Deep Deterministic Policy Gradient

Solution: Use another neural network to estimate the policy $\pi(s) \approx \arg\max_a Q(s,a)$! Next week, we'll talk more about policy gradient methods. But keep this in mind.

DDPG: Deep Deterministic Policy Gradient

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient: