Lecture 6: Model-Free Control

Joseph Modayil

Outline

Model-Free Reinforcement Learning

- Last lecture:
  - Model-free prediction
  - *Estimate* the value function of an *unknown* MDP
- This lecture:
  - Model-free control
  - *Optimise* the value function of an *unknown* MDP

Uses of Model-Free Control

Some example problems that can be modelled as MDPs

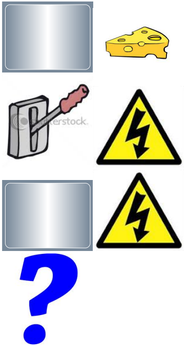
- Elevator
- Parallel Parking
- Ship Steering
- Bioreactor
- Helicopter
- Aeroplane Logistics
- Robocup Soccer
- Portfolio management
- Protein Folding
- Robot walking
- Atari video games
- Game of Go

For most of these problems, either:

- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, except by samples

Model-free control can solve these problems

Rat Example

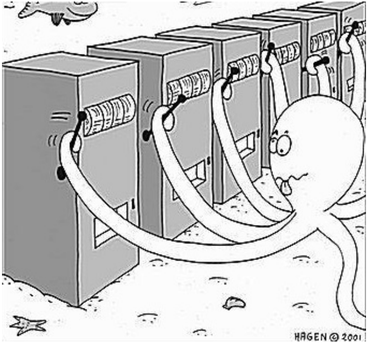


Exploration vs. Exploitation

- Online decision-making involves a fundamental choice:
  - Exploitation: Maximize return given current knowledge
  - Exploration: Increase knowledge
- The best long-term strategy may involve short-term sacrifices
- Gather enough information to make the best overall decisions

The Multi-Armed Bandit

- A multi-armed bandit is a tuple $\langle \mathcal{A}, \mathcal{R} \rangle$
- $\mathcal{A}$ is a known set of actions (or "arms")
- $\mathcal{R}^a(r) = \mathbb{P}[R_{t+1} = r | A_t = a]$ is an unknown probability distribution over rewards
- At each step t the agent selects an action $A_t \in \mathcal{A}$
- The environment generates a reward $R_{t+1} \sim \mathcal{R}^{A_t}$
- The goal is to maximize cumulative reward $\sum_{i=1}^{t} R_i$
- Repeated 'game against nature'



Action values

- The true *action value* for action a is the expected reward

$$q(a) = \mathbb{E}\left[R_{t+1} | A_t = a\right]$$

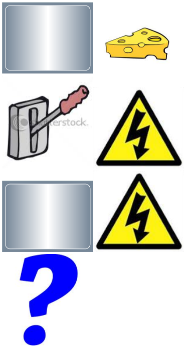
- We consider algorithms that estimate $Q_t(a) \approx q(a)$
- The *count* $N_t(a)$ is number of times we selected action a
- Monte-Carlo estimates:

$$Q_{t+1}(a) = \frac{1}{N_{t+1}(a)} \sum_{i=1}^{t} R_{i+1} \mathbf{1}(A_i = a)$$

- The *greedy* algorithm selects action with highest value

$$a^g_{t+1} = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q_{t+1}(a)$$

Rat Example

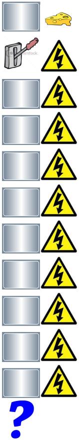


- Cheese: $R = +1$
- Shock: $R = -1$
- We can estimate action values:

$$Q_3(\text{button}) = 0$$
$$Q_3(\text{lever}) = -1$$

- When should we stop being greedy?

Rat Example



- Cheese: $R = +1$
- Shock: $R = -1$
- We can estimate action values:

$$Q_3(\text{button}) = \mathbf{-0.8}$$
$$Q_3(\text{lever}) = -1$$

- When should we stop being greedy?

Exploration

- We need to explore to learn about the values of all actions
- What is a good way to explore?
- One common solution: ϵ-greedy
  - Select greedy action (exploit) w.p. $1 - \epsilon$
  - Select random action (explore) w.p. ϵ
- Used in Atari
- Is this enough?
- How to pick ϵ?

Bandit Methods

- We have barely scratched the surface of bandit problems
- Many practical algorithms with theoretical guarantees.
- Extensions of bandit results to MDPs is open research
- The exploration-exploitation dilemma is natural in control
- We often still use ϵ-greedy exploration.

Function Approximation in Brief

- We have seen function approximation in earlier lectures
- We will introduce some common terminology
- We will see some recurring formulas that connect our losses, updates, and algorithms.

Common Kinds of Function Approximation

- Tabular: No generalization across states. The environmental state is used by the learning algorithm. Every real-valued function of state can be represented exactly.

$$f(s) \approx \hat{f}(s;\theta) = \theta[s] \quad \longrightarrow \quad \nabla_\theta \hat{f}(s;\theta_k) = e_s \ \{\text{a unit vector}\}$$

- Linear: A function f over the state space is approximated as a linear function of a feature vector $x : \mathcal{S} \to \Re^n$

$$f(s) \approx \hat{f}(s;\theta) := \theta^\top x(s) \quad \longrightarrow \quad \nabla_\theta \hat{f}(s;\theta_k) = x(s)$$

  State Aggregation: Special case where $x(s)$ is a unit vector

- Differentiable: We have a space of differentiable functions parameterized by $\theta \in \Re^n$.

$$f(s) \approx \hat{f}(s;\theta) \quad \longrightarrow \quad \nabla_\theta \hat{f}(s;\theta_k)$$

Converting Between Losses and a Parameter Update

- Consider a loss between a target y_t and an estimate $\hat{f}(x_t; \theta_k)$.

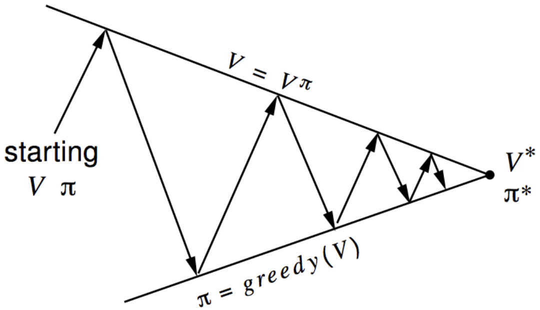
$$L(\{t\}, \theta_k) = \frac{1}{2}(y_t - \hat{f}(x_t; \theta_k))^2$$

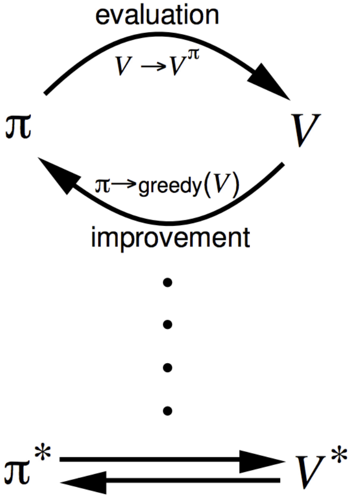
- We can update the parameter vector θ_k (with learning rate α).

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\{t\}, \theta_k)$$
$$\theta_{k+1} = \theta_k + \alpha(y_t - \hat{f}(x_t; \theta_k)) \nabla_\theta \hat{f}(x_t; \theta_k)$$

- We can derive a parameter update from a loss.
- We can associate a loss to a (conventional) parameter update.

Generalized Policy Iteration (Refresher)





Policy evaluation Estimate V^π

e.g. Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

e.g. Greedy policy improvement

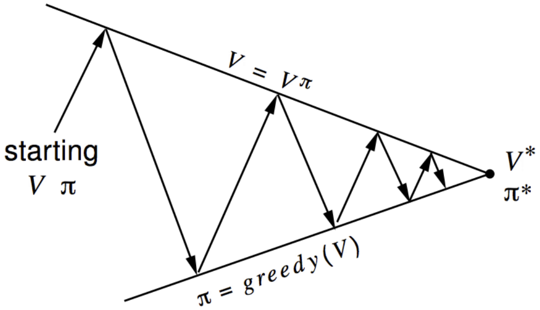
Monte Carlo

- Recall, Monte Carlo estimate from state S_t is

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

- $\mathbb{E}[G_t] = V^\pi$
- So, we can average multiple estimates to get V^π

Policy Iteration With Monte-Carlo Evaluation



Policy evaluation Monte-Carlo policy evaluation, $V = V^{\pi}$?

Policy improvement Greedy policy improvement?

Model-Free Policy Iteration Using Action-Value Function

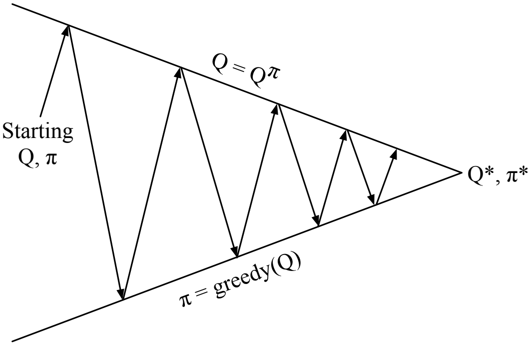
- Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \mathcal{R}_s^A + \mathcal{P}_{ss'}^A V(s')$$

- Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s, a)$$

Generalised Policy Iteration with Action-Value Function



Policy evaluation Monte-Carlo policy evaluation, $Q = Q^\pi$

Policy improvement Greedy policy improvement?

ϵ-Greedy Policy Improvement

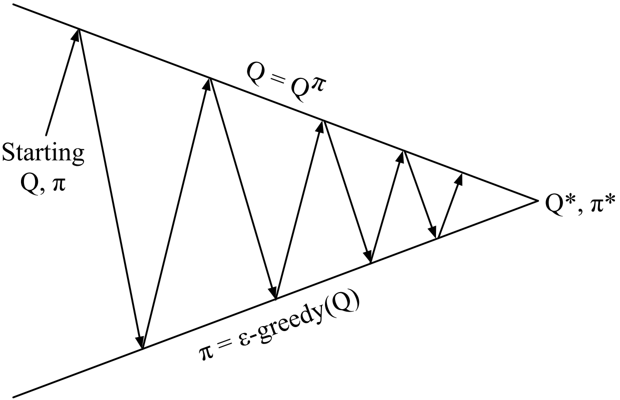
Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to Q^π is not worse, $V^{\pi'}(s) \geq V^\pi(s)$

$$
\begin{aligned}
Q^\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(s, a) Q^\pi(s, a) \\
&= \epsilon/m \sum_{a \in \mathcal{A}} Q^\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} Q^\pi(s, a) \\
&\geq \epsilon/m \sum_{a \in \mathcal{A}} Q^\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(s, a) - \epsilon/m}{1 - \epsilon} Q^\pi(s, a) \\
&= \sum_{a \in \mathcal{A}} \pi(s, a) Q^\pi(s, a) = V^\pi(s)
\end{aligned}
$$

Then from policy improvement theorem, $V^{\pi'}(s) \geq Q^\pi(s, \pi'(s)) \geq V^\pi(s)$

Monte-Carlo Policy Iteration

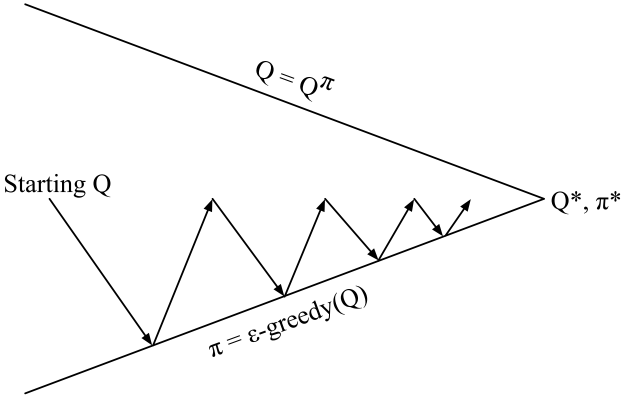


Policy evaluation Monte-Carlo policy evaluation, $Q = Q^\pi$

Policy improvement ϵ-greedy policy improvement

Here π^* is best ϵ-greedy policy

Monte-Carlo Generalized Policy Iteration



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx Q^\pi$

Policy improvement ϵ-greedy policy improvement

GLIE

Definition

Greedy in the Limit with Infinite Exploration (GLIE)

- Consider a tabular problem (no func. approx.)
- All state-action pairs are explored infinitely many times,

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- The policy converges on a greedy policy,

$$\lim_{k \to \infty} \pi_k(s, a) = \mathbf{1}(a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\ Q_k(s, a'))$$

- For example, ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

GLIE Every-Visit Monte-Carlo Control

- Sample kth episode using π: $\{S_1, A_1, R_2, ..., S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

$$
\begin{aligned}
N(S_t, A_t) &\leftarrow N(S_t, A_t) + 1 \\
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \left(G_t - Q(S_t, A_t)\right)
\end{aligned}
$$

- Improve policy based on new action-value function

$$
\begin{aligned}
\epsilon &\leftarrow 1/k \\
\pi &\leftarrow \epsilon\text{-greedy}(Q)
\end{aligned}
$$

Theorem

GLIE Monte-Carlo control converges to the optimal action-value function, $Q(s, a) \rightarrow Q^*(s, a)$

GLIE Every-Visit Monte-Carlo Control

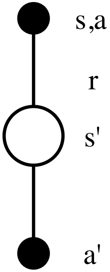
$$
\begin{aligned}
N(S_t, A_t) &\leftarrow N(S_t, A_t) + 1 \\
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \left(G_t - Q(S_t, A_t)\right) \\
\epsilon &\leftarrow 1/k \\
\pi &\leftarrow \epsilon\text{-greedy}(Q)
\end{aligned}
$$

- Any practical issues with this algorithm?
- What is the loss for function approximation?

MC vs. TD Control

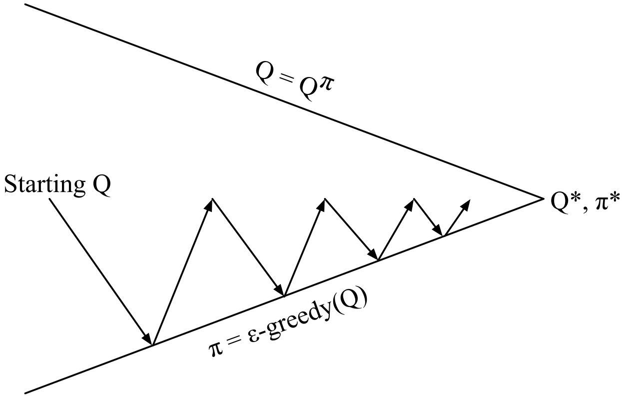
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Can learn from incomplete sequences
- Natural idea: use TD instead of MC for control
  - Apply TD to $Q(s, a)$
  - Use ϵ-greedy policy improvement
  - Update every time-step

Updating Action-Value Functions with Sarsa



$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R + \gamma Q(s',a') - Q(s,a) \right)$$

Sarsa



Every time-step:

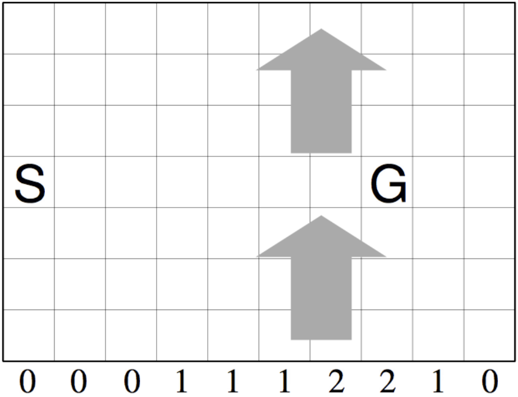
Policy evaluation Sarsa, $Q \approx Q^{\pi}$

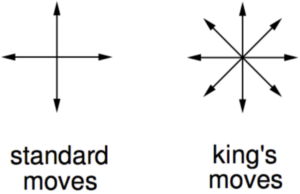
Policy improvement ϵ-greedy policy improvement

Tabular Sarsa

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g., ε-greedy)
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    until s is terminal

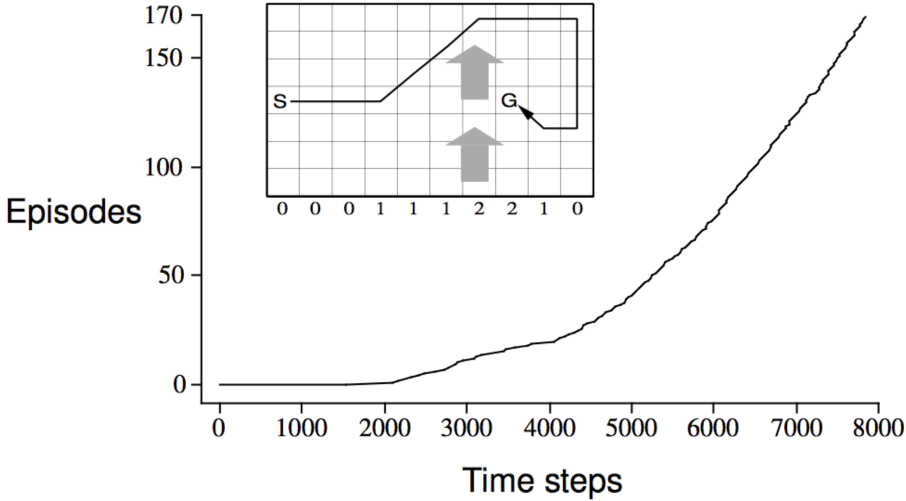
Windy Gridworld Example





- Reward = -1 per time-step until reaching goal
- Undiscounted

Sarsa on the Windy Gridworld



n-Step Sarsa

- Consider the following n-step returns for $n = 1, 2, \ldots, \infty$:

$$
\begin{array}{lll}
n = 1 & \textit{Sarsa}(0) & G_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}) \\
n = 2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}) \\
\vdots & & \vdots \\
n = \infty & MC & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T
\end{array}
$$

- Define the n-step Q-return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

- n-step Sarsa updates $Q(s, a)$ towards the n-step Q-return

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(G_t^{(n)} - Q(S_t, A_t) \right)$$

Backward View Sarsa(λ)

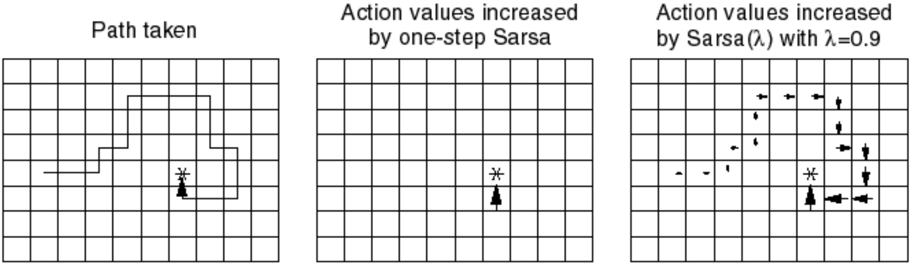
- Just like TD(λ), we can use eligibility traces in an online algorithm (tabular here)
- But Sarsa(λ) has one eligibility trace for each state-action pair

$$
\begin{aligned}
E_0(s, a) &= 0 \\
E_t(s, a) &= \gamma \lambda E_{t-1}(s, a) + \mathbf{1}(S_t = s, A_t = a) \;, \; \forall t > 0
\end{aligned}
$$

- $Q(s, a)$ is updated for every state s and action a
- In proportion to TD-error δ_t and eligibility trace $E_t(s, a)$

$$
\begin{aligned}
\delta_t &= R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \\
\forall s, a : Q(s, a) &\leftarrow Q(s, a) + \alpha \delta_t E_t(s, a)
\end{aligned}
$$

Sarsa(λ) Gridworld Example



On and Off-Policy Learning

- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π
- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from μ

Off-Policy Learning

- Evaluate target policy $\pi(s,a)$ to compute $V^{\pi}(s)$ or $Q^{\pi}(s,a)$
- While following behaviour policy $\mu(s,a)$

$$\{S_1, A_1, R_2, ..., S_T\} \sim \mu$$

- Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t-1}$
- Learn about *optimal* policy while following *exploratory* policy
- Learn about *multiple* policies while following *one* policy

Importance Sampling

- Estimate the expectation of a different distribution

$$
\begin{aligned}
\mathbb{E}_{x \sim d}[f(x)] &= \sum d(x) f(x) \\
&= \sum d'(x) \frac{d(x)}{d'(x)} f(x) \\
&= \mathbb{E}_{x \sim d'}\left[\frac{d(x)}{d'(x)} f(x)\right]
\end{aligned}
$$

Importance Sampling for Off-Policy Monte-Carlo

- Use returns generated from μ to evaluate π
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(S_t, A_t)}{\mu(S_t, A_t)} \frac{\pi(S_{t+1}, A_{t+1})}{\mu(S_{t+1}, A_{t+1})} \dots \frac{\pi(S_T, A_T)}{\mu(S_T, A_T)} G_t$$

- Update value towards *corrected* return

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\color{red}{G_t^{\pi/\mu}} - V(S_t) \right)$$

- Importance sampling can dramatically increase variance

Importance Sampling for Off-Policy TD Updates

- Use TD targets generated from μ to evaluate π
- Weight TD target $r + \gamma V(s')$ by importance sampling
- Only need a single importance sampling correction

$$
\begin{aligned}
V(S_t) \leftarrow & \; V(S_t) + \\
& \alpha \left(\frac{\pi(S_t, A_t)}{\mu(S_t, A_t)} \left(R_{t+1} + \gamma V(S_{t+1})\right) - V(S_t) \right)
\end{aligned}
$$

- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- No importance sampling is required
- Next action may be chosen using behaviour policy $A_{t+1} \sim \mu(S_{t+1}, \cdot)$
- But we consider probabilities under $\pi(S_t, \cdot)$
- Update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

- Called Expected Sarsa (when $\mu = \pi$) or Generalized Q-learning

Off-Policy Control with Q-Learning

- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \underset{a'}{\operatorname{argmax}}\ Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned}
&R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) \\
=&R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\ Q(S_{t+1}, a)) \\
=&R_{t+1} + \gamma \max_a Q(S_{t+1}, a)
\end{aligned}$$

Q-Learning Control Algorithm

Theorem

Q-learning control converges to the optimal action-value function, $Q(s, a) \rightarrow Q^*(s, a)$*, as long as we take each action in each state infinitely often.*

Note: no need for greedy behaviour!

Q-Learning Algorithm for Off-Policy Control

For $t = 0, 1, 2, \ldots$

Take action A_t according to $\pi_t(S_t)$, observe R_{t+1}, S_{t+1}

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma_t \max_a Q_t(S_{t+1}, a) - Q_t(S_t, A_t) \right)$$

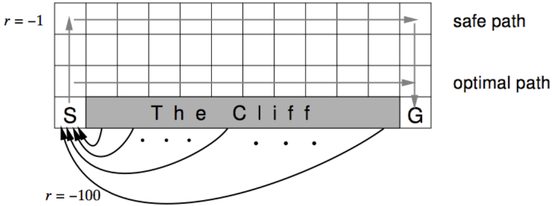
Note:

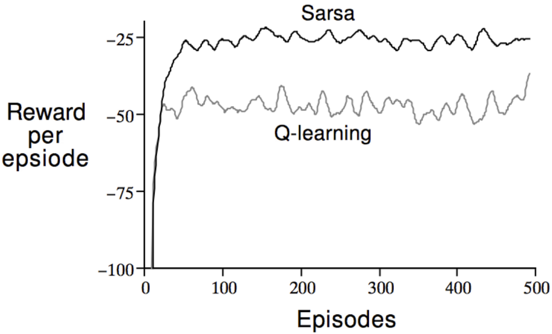
1. π_t can be the ϵ-greedy policy induced by Q_t.
2. Q-learning update for a differentiable $Q_t(s, a) = \hat{q}(s, a; \theta_t)$

$$\theta_{t+1} = \theta_t + \alpha \left(R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a; \theta_t) - \hat{q}(S_t, A_t; \theta_t) \right) \nabla_\theta \hat{q}(S_t, A_t; \theta_t)$$

3. More stable updates described in later slides.

Cliff Walking Example





Deep Networks as Function Approximators

- The strongest theoretical RL results have been achieved with tabular representations.
- The strongest empirical performance has been achieved with deep networks.
- To improve the reliability of training deep networks, two modifications to the classical online RL training loop have been found to be crucial in practice for robust learning.
  - Experience replay - Learning with a mini-batch of transitions drawn at random from recent history.
  - Stationary targets - Bootstrapping from a frozen value function estimate.

Experience Replay

- In classical Q-learning, the value function estimate is updated with each experienced transition. The classical method enables the learning agent to adapt its behaviour quickly in response to new information.
- With deep networks, it is better to update the value function using a minibatch drawn from recent history. Drawing a minibatch from recent history gives a lower variance estimate of the gradient.
- Sequential samples in time are often strongly correlated, and only using such samples can lead the network to overfit to the recent past.

Stationary Targets

- In classical RL, the TD-error δ is often computed using the current best estimate θ_t.

$$\delta = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t) - Q(S_t, A_t; \theta_t)$$
$$\theta_{t+1} = \theta_t + \alpha \delta \nabla_\theta Q(S_t, A_t; \theta_t)$$

- With deep nets, it is better to keep an older value function estimate θ^-, and use that for the bootstrap estimate.

$$\delta = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta^-) - Q(S_t, A_t; \theta_t)$$
$$\theta_{t+1} = \theta_t + \alpha \delta \nabla_\theta Q(S_t, A_t; \theta_t)$$

- The parameter θ^- is set to the most recent θ_t infrequently, perhaps once every few thousand samples.
- Infrequent changes of the target makes the algorithm closer to a supervised learning task, and improves empirical stability.

Q-learning overestimation

- Classical Q-learning has additional problems
- Recall

$$\max_a Q_t(S_{t+1}, a) = Q_t(S_{t+1}, \operatorname*{argmax}_a Q_t(S_{t+1}, a))$$

- Q-learning uses same values to select and to evaluate
- ... but values are approximate
- Therefore:
  - more likely to select overestimated values
  - less likely to select underestimated values
- This causes upward bias

Double Q-learning

- Q-learning uses same values to select and to evaluate

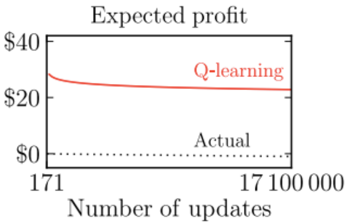
$$R_{t+1} + \gamma Q_t(S_{t+1}, \underset{a}{\operatorname{argmax}}\ Q_t(S_{t+1}, a))$$

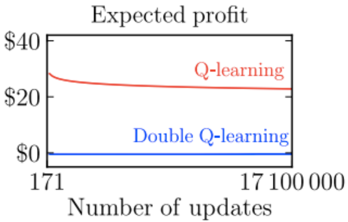
- Solution: decouple selection from evaluation
- Double Q-learning:

$$R_{t+1} + \gamma Q'_t(S_{t+1}, \underset{a}{\operatorname{argmax}}\ Q_t(S_{t+1}, a))$$
$$R_{t+1} + \gamma Q_t(S_{t+1}, \underset{a}{\operatorname{argmax}}\ Q'_t(S_{t+1}, a))$$

- Then update one at random for each experience
- For deep nets, the stationary target gives a second Q function.

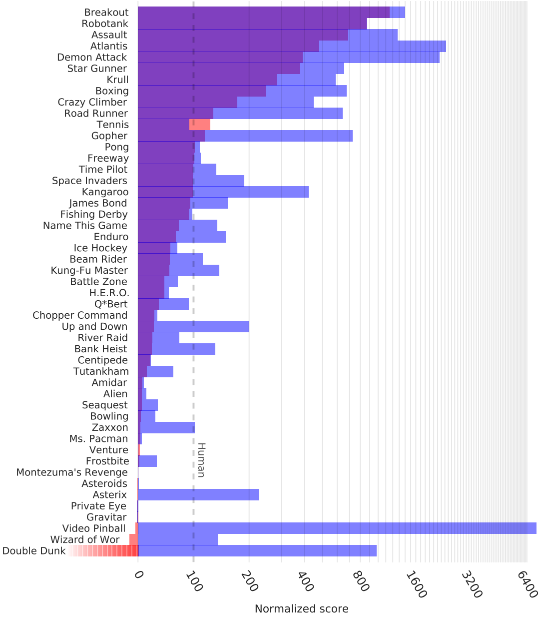
Q-learning overestimates



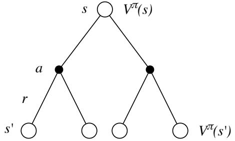


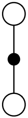
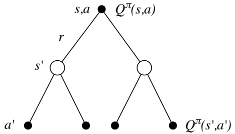
Double DQN on Atari

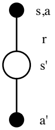
DQN
Double DQN

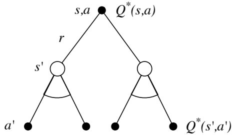


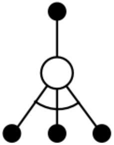
Relationship Between DP and TD

<table>
<tr><th></th><th>Full Backup (DP)</th><th>Sample Backup (TD)</th></tr>
<tr><td>Bellman Expectation
Equation for $V^\pi(s)$</td><td>

Iterative Policy Evaluation</td><td>
TD Learning</td></tr>
<tr><td>Bellman Expectation
Equation for $Q^\pi(s, a)$</td><td>

Q-Policy Iteration</td><td>

Sarsa</td></tr>
<tr><td>Bellman Optimality
Equation for $Q^*(s, a)$</td><td>

Q-Value Iteration</td><td>
Q-Learning</td></tr>
</table>

Questions?