Deep Reinforcement Learning: Policy Gradients and Q-Learning

John Schulman

OpenAI

Bay Area Deep Learning School
September 24, 2016

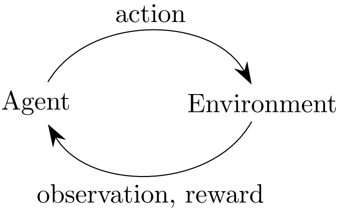
Introduction and Overview

Aim of This Talk

- What is deep RL, and should I use it?
- Overview of the leading techniques in deep reinforcement learning
  - Policy gradient methods
  - Q-learning and SARSA
  - What are their pros and cons?

What is Reinforcement Learning?

- Branch of machine learning concerned with taking sequences of actions
- Usually described in terms of agent interacting with a previously unknown environment, trying to maximize cumulative reward



What Is Deep Reinforcement Learning?

Reinforcement learning using neural networks to approximate functions

- Policies (select next action)
- Value functions (measure goodness of states or state-action pairs)
- Models (predict next states and rewards)

Motor Control and Robotics

Robotics:

- Observations: camera images, joint angles
- Actions: joint torques
- Rewards: stay balanced, navigate to target locations, serve and protect humans

Business Operations

Inventory Management

- Observations: current inventory levels
- Actions: number of units of each item to purchase
- Rewards: profit

In Other ML Problems

- Hard Attention[1]
  - Observation: current image window
  - Action: where to look
  - Reward: classification
- Sequential/structured prediction, e.g., machine translation[2]
  - Observations: words in source language
  - Actions: emit word in target language
  - Rewards: sentence-level metric, e.g. BLEU score

---

[1] V. Mnih et al. "Recurrent models of visual attention". In: *Advances in Neural Information Processing Systems*. 2014, pp. 2204–2212.

[2] H. Daumé Iii, J. Langford, and D. Marcu. "Search-based structured prediction". In: *Machine learning* 75.3 (2009), pp. 297–325; S. Ross, G. J. Gordon, and D. Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning." In: *AISTATS*. vol. 1. 2. 2011, p. 6; M. Ranzato et al. "Sequence level training with recurrent neural networks". In: *arXiv preprint arXiv:1511.06732* (2015).

How Does RL Relate to Other ML Problems?

How Does RL Relate to Other ML Problems?

Supervised learning:

- Environment samples input-output pair $(x_t, y_t) \sim \rho$
- Agent predicts $\hat{y}_t = f(x_t)$
- Agent receives loss $\ell(y_t, \hat{y}_t)$
- *Environment asks agent a question, and then tells her the right answer*

How Does RL Relate to Other ML Problems?

Contextual bandits:

- Environment samples input $x_t \sim \rho$
- Agent takes action $\hat{y}_t = f(x_t)$
- Agent receives cost $c_t \sim P(c_t \mid x_t, \hat{y}_t)$ where P is an unknown probability distribution
- *Environment asks agent a question, and gives her a noisy score on her answer*
- Application: personalized recommendations

How Does RL Relate to Other ML Problems?

Reinforcement learning:

- Environment samples input $x_t \sim P(x_t \mid x_{t-1}, y_{t-1})$
  - Input depends on your previous actions!
- Agent takes action $\hat{y}_t = f(x_t)$
- Agent receives cost $c_t \sim P(c_t \mid x_t, \hat{y}_t)$ where P a probability distribution unknown to the agent.

How Does RL Relate to Other Machine Learning Problems?

Summary of differences between RL and supervised learning:

- You don't have full access to the function you're trying to optimize—must query it through interaction.
- Interacting with a *stateful* world: input x_t depend on your previous actions

Should I Use Deep RL On My Practical Problem?

- Might be overkill
- Other methods worth investigating first
  - Derivative-free optimization (simulated annealing, cross entropy method, SPSA)
  - Is it a contextual bandit problem?
  - Non-deep RL methods developed by Operations Research community[3]

[3] W. B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Vol. 703. John Wiley & Sons, 2007.

Recent Success Stories in Deep RL

- ATARI using deep Q-learning[4], policy gradients[5], DAGGER[6]
- Superhuman Go using supervised learning + policy gradients + Monte Carlo tree search + value functions[7]
- Robotic manipulation using guided policy search[8]
- Robotic locomotion using policy gradients[9]
- 3D games using policy gradients[10]

---

[4] V. Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[5] J. Schulman et al. "Trust Region Policy Optimization". In: *arXiv preprint arXiv:1502.05477* (2015).

[6] X. Guo et al. "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning". In: *Advances in Neural Information Processing Systems*. 2014, pp. 3338–3346.

[7] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[8] S. Levine et al. "End-to-end training of deep visuomotor policies". In: *arXiv preprint arXiv:1504.00702* (2015).

[9] J. Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[10] V. Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *arXiv preprint arXiv:1602.01783* (2016).

Markov Decision Processes

Definition

- Markov Decision Process (MDP) defined by $(\mathcal{S}, \mathcal{A}, P)$, where
  - $\mathcal{S}$: **state space**
  - $\mathcal{A}$: **action space**
  - $P(r, s' \mid s, a)$: transition $+$ reward probability distribution
- Extra objects defined depending on problem setting
  - μ: Initial state distribution
- Optimization problem: maximize expected cumulative reward

Episodic Setting

- In each episode, the initial state is sampled from μ, and the agent acts until the *terminal state* is reached. For example:
  - Taxi robot reaches its destination (termination $=$ good)
  - Waiter robot finishes a shift (fixed time)
  - Walking robot falls over (termination $=$ bad)
- Goal: maximize expected reward per episode

Policies

- Deterministic policies: $a = \pi(s)$
- Stochastic policies: $a \sim \pi(a \mid s)$

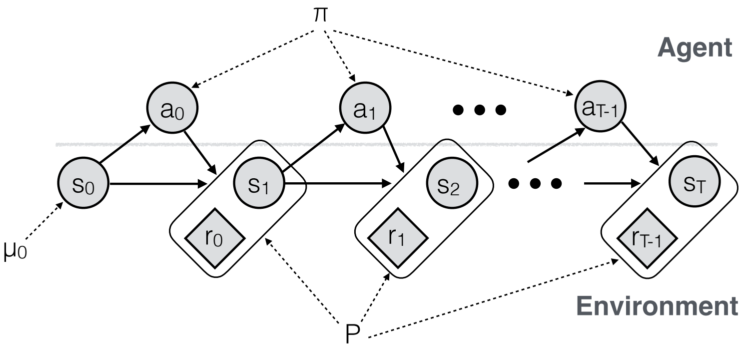
Episodic Setting

$$
\begin{aligned}
s_0 &\sim \mu(s_0) \\
a_0 &\sim \pi(a_0 \mid s_0) \\
s_1, r_0 &\sim P(s_1, r_0 \mid s_0, a_0) \\
a_1 &\sim \pi(a_1 \mid s_1) \\
s_2, r_1 &\sim P(s_2, r_1 \mid s_1, a_1) \\
&\dots \\
a_{T-1} &\sim \pi(a_{T-1} \mid s_{T-1}) \\
s_T, r_{T-1} &\sim P(s_T \mid s_{T-1}, a_{T-1})
\end{aligned}
$$

Objective:

$$
\begin{aligned}
&\text{maximize}\, \eta(\pi), \;\text{where} \\
&\quad \eta(\pi) = E[r_0 + r_1 + \cdots + r_{T-1} \mid \pi]
\end{aligned}
$$

Episodic Setting



Objective:

$$\text{maximize}\, \eta(\pi), \;\; \text{where}$$
$$\eta(\pi) = E[r_0 + r_1 + \cdots + r_{T-1} \mid \pi]$$

Parameterized Policies

- A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$
  - Deterministic: $a = \pi(s, \theta)$
  - Stochastic: $\pi(a \mid s, \theta)$
- Analogous to classification or regression with input s, output a.
  - Discrete action space: network outputs vector of probabilities
  - Continuous action space: network outputs mean and diagonal covariance of Gaussian

Policy Gradient Methods

Policy Gradient Methods: Overview

Problem:

$$\text{maximize } E[R \mid \pi_\theta]$$

Intuitions: collect a bunch of trajectories, and ...

1. Make the good trajectories more probable
2. Make the good actions more probable
3. Push the actions towards good actions (DPG[11], SVG[12])

---

[11] D. Silver et al. "Deterministic policy gradient algorithms". In: *ICML*. 2014.

[12] N. Heess et al. "Learning continuous control policies by stochastic value gradients". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2926–2934.

Score Function Gradient Estimator

- Consider an expectation $E_{x \sim p(x \mid \theta)}[f(x)]$. Want to compute gradient wrt θ

$$
\begin{aligned}
\nabla_\theta E_x[f(x)] &= \nabla_\theta \int \mathrm{d}x \; p(x \mid \theta) f(x) \\
&= \int \mathrm{d}x \; \nabla_\theta p(x \mid \theta) f(x) \\
&= \int \mathrm{d}x \; p(x \mid \theta) \frac{\nabla_\theta p(x \mid \theta)}{p(x \mid \theta)} f(x) \\
&= \int \mathrm{d}x \; p(x \mid \theta) \nabla_\theta \log p(x \mid \theta) f(x) \\
&= E_x[f(x) \nabla_\theta \log p(x \mid \theta)].
\end{aligned}
$$

- Last expression gives us an unbiased gradient estimator. Just sample $x_i \sim p(x \mid \theta)$, and compute $\hat{g}_i = f(x_i) \nabla_\theta \log p(x_i \mid \theta)$.
- Need to be able to compute and differentiate density $p(x \mid \theta)$ wrt θ

Derivation via Importance Sampling

Alternative Derivation Using Importance Sampling[13]

$$\mathbb{E}_{x\sim\theta}\left[f(x)\right] = \mathbb{E}_{x\sim\theta_{\text{old}}}\left[\frac{p(x \mid \theta)}{p(x \mid \theta_{\text{old}})} f(x)\right]$$

$$\nabla_\theta \mathbb{E}_{x\sim\theta}\left[f(x)\right] = \mathbb{E}_{x\sim\theta_{\text{old}}}\left[\frac{\nabla_\theta p(x \mid \theta)}{p(x \mid \theta_{\text{old}})} f(x)\right]$$

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{x\sim\theta}\left[f(x)\right]\Big|_{\theta=\theta_{\text{old}}} &= \mathbb{E}_{x\sim\theta_{\text{old}}}\left[\frac{\nabla_\theta p(x \mid \theta)\big|_{\theta=\theta_{\text{old}}}}{p(x \mid \theta_{\text{old}})} f(x)\right] \\
&= \mathbb{E}_{x\sim\theta_{\text{old}}}\left[\nabla_\theta \log p(x \mid \theta)\big|_{\theta=\theta_{\text{old}}} f(x)\right]
\end{aligned}$$

[13] T. Jie and P. Abbeel. "On a connection between importance sampling and the likelihood ratio policy gradient". In: *Advances in Neural Information Processing Systems*. 2010, pp. 1000–1008.

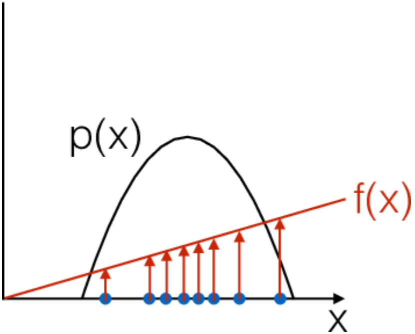
Score Function Gradient Estimator: Intuition

$$\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i \mid \theta)$$

- Let's say that $f(x)$ measures how good the sample x is.
- Moving in the direction $\hat{g}_i$ pushes up the logprob of the sample, in proportion to how good it is
- *Valid even if $f(x)$ is discontinuous, and unknown, or sample space (containing x) is a discrete set*

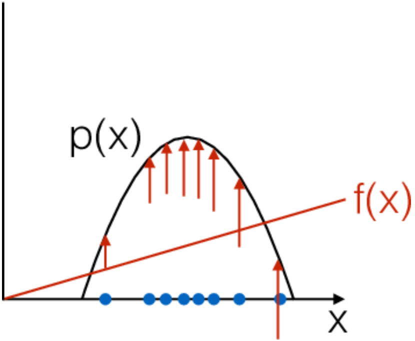
Score Function Gradient Estimator: Intuition

$$\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i \mid \theta)$$



Score Function Gradient Estimator: Intuition

$$\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i \mid \theta)$$



Score Function Gradient Estimator for Policies

- Now random variable x is a whole trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau[\nabla_\theta \log p(\tau \mid \theta) R(\tau)]$$

- Just need to write out $p(\tau \mid \theta)$:

$$p(\tau \mid \theta) = \mu(s_0) \prod_{t=0}^{T-1} [\pi(a_t \mid s_t, \theta) P(s_{t+1}, r_t \mid s_t, a_t)]$$

$$\log p(\tau \mid \theta) = \log \mu(s_0) + \sum_{t=0}^{T-1} [\log \pi(a_t \mid s_t, \theta) + \log P(s_{t+1}, r_t \mid s_t, a_t)]$$

$$\nabla_\theta \log p(\tau \mid \theta) = \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t \mid s_t, \theta)$$

$$\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau \left[R \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t \mid s_t, \theta) \right]$$

- Interpretation: using good trajectories (high R) as supervised examples in classification / regression

Policy Gradient: Use Temporal Structure

- Previous slide:

$$\nabla_\theta \mathbb{E}_\tau [R] = \mathbb{E}_\tau \left[\left(\sum_{t=0}^{T-1} r_t \right) \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \right) \right]$$

- We can repeat the same argument to derive the gradient estimator for a single reward term $r_{t'}$.

$$\nabla_\theta \mathbb{E} [r_{t'}] = \mathbb{E} \left[r_{t'} \sum_{t=0}^{t} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \right]$$

- Sum this formula over t, we obtain

$$\begin{aligned} \nabla_\theta \mathbb{E} [R] &= \mathbb{E} \left[\sum_{t=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \right] \\ &= \mathbb{E} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \sum_{t'=t}^{T-1} r_{t'} \right] \end{aligned}$$

Policy Gradient: Introduce Baseline

- Further reduce variance by introducing a *baseline* $b(s)$

$$\nabla_\theta \mathbb{E}_\tau [R] = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \right]$$

- For any choice of b, gradient estimator is unbiased.
- Near optimal choice is expected return, $b(s_t) \approx \mathbb{E}\left[r_t + r_{t+1} + r_{t+2} + \cdots + r_{T-1}\right]$
- Interpretation: increase logprob of action a_t proportionally to how much returns $\sum_{t=t'}^{T-1} r_{t'}$ are better than expected

Discounts for Variance Reduction

- Introduce discount factor γ, which ignores delayed effects between actions and rewards

$$\nabla_\theta \mathbb{E}_\tau [R] \approx \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - b(s_t) \right) \right]$$

- Now, we want
$b(s_t) \approx \mathbb{E} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1-t} r_{T-1} \right]$
- Write gradient estimator more generally as

$$\nabla_\theta \mathbb{E}_\tau [R] \approx \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \hat{A}_t \right]$$

$\hat{A}_t$ is the *advantage estimate*

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b

for iteration$=1, 2, \ldots$ **do**

Collect a set of trajectories by executing the current policy

At each timestep in each trajectory, compute

the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and

the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.

Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,

summed over all trajectories and timesteps.

Update the policy, using a policy gradient estimate $\hat{g}$,

which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$

end for

Extension: Step Sizes and Trust Regions

Why are step sizes a big deal in RL?

- Supervised learning
  - Step too far $\rightarrow$ next update will fix it
- Reinforcement learning
  - Step too far $\rightarrow$ bad policy
  - Next batch: collected under bad policy
  - Can't recover, collapse in performance!

Extension: Step Sizes and Trust Regions

- Trust Region Policy Optimization: limit KL divergence between action distribution of pre-update and post-update policy[14]

$$\mathbb{E}_s \left[D^{\mathrm{KL}}(\pi_{\mathrm{old}}(\cdot \mid s) \parallel \pi(\cdot \mid s)) \right] \leq \delta$$

- Closely elated to previous natural policy gradient methods[15]

---

[14] J. Schulman et al. "Trust Region Policy Optimization". In: *arXiv preprint arXiv:1502.05477* (2015).

[15] S. Kakade. "A Natural Policy Gradient." In: *NIPS*. vol. 14. 2001, pp. 1531–1538; J. A. Bagnell and J. Schneider. "Covariant policy search". In: IJCAI. 2003; J. Peters and S. Schaal. "Natural actor-critic". In: *Neurocomputing* 71.7 (2008), pp. 1180–1190.

Extension: Further Variance Reduction

- Use value functions for more variance reduction (at the cost of bias): actor-critic methods[16]
- Reparameterization trick: instead of increasing the probability of the good actions, push the actions towards (hopefully) better actions[17]

[16] J. Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015); V. Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *arXiv preprint arXiv:1602.01783* (2016).

[17] D. Silver et al. "Deterministic policy gradient algorithms". In: *ICML*. 2014; N. Heess et al. "Learning continuous control policies by stochastic value gradients". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2926–2934.

Interlude

Q-Function Learning Methods

Value Functions

- Definitions:

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \mid s_0 = s, a_0 = a\right]$$

Called Q-function or state-action-value function

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \mid s_0 = s\right] \\ &= \mathbb{E}_{a\sim\pi}\left[Q^\pi(s,a)\right] \end{aligned}$$

Called state-value function

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Called advantage function

- This section considers methods that explicitly store Q-functions instead of policies π, and updates them using *Bellman* equations

Bellman Equations for Q^π

- Bellman equation for Q^π

$$
\begin{aligned}
Q^\pi(s_0, a_0) &= \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)} [r_0 + \gamma V^\pi(s_1)] \\
&= \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)} [r_0 + \gamma \mathbb{E}_{a_1 \sim \pi} [Q^\pi(s_1, a_1)]]
\end{aligned}
$$

- We can write out Q^π with k-step empirical returns

$$
\begin{aligned}
Q^\pi(s_0, a_0) &= \mathbb{E}_{s_1, a_1 \mid s_0, a_0} [r_0 + \gamma V^\pi(s_1, a_1)] \\
&= \mathbb{E}_{s_1, a_1, s_2, a_2 \mid s_0, a_0} \left[r_0 + \gamma r_1 + \gamma^2 Q^\pi(s_2, a_2)\right] \\
&= \mathbb{E}_{s_1, a_1 \ldots, s_k, a_k \mid s_0, a_0} \left[r_0 + \gamma r_1 + \cdots + \gamma^{k-1} r_{k-1} + \gamma^k Q^\pi(s
\end{aligned}
$$

Bellman Backups

- From previous slide:

$$Q^\pi(s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)}\left[r_0 + \gamma \mathbb{E}_{a_1 \sim \pi}\left[Q^\pi(s_1, a_1)\right]\right]$$

- Define the Bellman backup operator (operating on Q-functions) as follows

$$[\mathcal{B}^\pi Q](s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)}\left[r_0 + \gamma \mathbb{E}_{a_1 \sim \pi}\left[Q(s_1, a_1)\right]\right]$$

- Then Q^π is a *fixed point* of this operator

$$\mathcal{B}^\pi Q^\pi = Q^\pi$$

- Furthermore, if we apply $\mathcal{B}^\pi$ repeatedly to any initial Q, the series converges to Q^π

$$Q,\ \mathcal{B}^\pi Q,\ (\mathcal{B}^\pi)^2 Q,\ (\mathcal{B}^\pi)^3 Q,\ \cdots \to Q^\pi$$

Introducing Q^*

- Let π^* denote an optimal policy
- Define $Q^* = Q^{\pi^*}$, which also satisfies $Q^*(s,a) = \max_\pi Q^\pi(s,a)$
- π^* is deterministic and satisfies $\pi^*(s) = \arg\max_a Q^*(s,a)$
- Thus, Bellman equation

$$Q^\pi(s_0,a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0,a_0)}\left[r_0 + \gamma \mathbb{E}_{a_1 \sim \pi}\left[Q^\pi(s_1,a_1)\right]\right]$$

becomes

$$Q^*(s_0,a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0,a_0)}\left[r_0 + \gamma \max_{a_1} Q^\pi(s_1,a_1)\right]$$

Bellman Operator for Q^*

- Define a corresponding Bellman backup operator

$$[BQ](s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)} \left[r_0 + \gamma \max_{a_1} Q(s_1, a_1) \right]$$

- Q^* is a fixed point of B:

$$BQ^* = Q^*$$

- If we apply B repeatedly to any initial Q, the series converges to Q^*

$$Q, \; BQ, \; B^2 Q, \; \cdots \to Q^*$$

Classic Algorithms for Solving MDPs

- Value iteration:
  - Initialize Q
  - Do $Q \leftarrow BQ$ until convergence
- Policy iteration:
  - Initialize π
  - Repeat:
    - Compute Q^π
    - $\pi \leftarrow \mathcal{G}Q^\pi$ ("greedy policy" for Q^π)
      where $[\mathcal{G}Q^\pi](s) = \arg\max_a Q^\pi(s, a)$
- To compute Q^π in policy iteration, we can solve linear equations exactly, or more commonly, do k Bellman backups $Q \leftarrow B^\pi Q$.

Sampling Based Algorithms

- Recall backup formulas for Q^π and Q^*

$$[BQ](s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)}\left[r_0 + \gamma \max_{a_1} Q(s_1, a_1)\right]$$

$$[B^\pi Q](s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)}\left[r_0 + \gamma \mathbb{E}_{a_1 \sim \pi}\left[Q(s_1, a_1)\right]\right]$$

- We can compute unbiased estimator of RHS of both equations using a single sample. Does not matter what policy was used to select actions!

$$[\widehat{BQ}](s_0, a_0) = r_0 + \gamma \max_{a_1} Q(s_1, a_1)$$

$$[\widehat{B^\pi Q}](s_0, a_0) = r_0 + \gamma \mathbb{E}_{a_1 \sim \pi}\left[Q(s_1, a_1)\right]$$

- Backups still converge to Q^π, Q^* with this noise[18]

[18] T. Jaakkola, M. I. Jordan, and S. P. Singh. "On the convergence of stochastic iterative dynamic programming algorithms". In: *Neural computation* 6.6 (1994), pp. 1185–1201; D. P. Bertsekas. *Dynamic programming and optimal control*. Vol. 2. 2. Athena Scientific, 2012.

Neural-Fitted Algorithms

- Parameterize Q-function with a neural network Q_θ
- Instead of $Q \leftarrow \widehat{BQ}$, do

$$\underset{\theta}{\text{minimize}} \sum_t \| Q_\theta(s_t, a_t) - \widehat{BQ}(s_t, a_t) \|^2 \tag{1}$$

- One version[19]

Algorithm 2 Neural-Fitted Q-Iteration (NFQ)

Initialize $\theta^{(0)}$.
for $n = 1, 2, \ldots$ **do**
  Sample trajectory using policy $\pi^{(n)}$.
  $\theta^{(n)} = \text{minimize}_\theta \sum_t (R_t + \gamma \max_{a'} Q_{\theta^{(n)}}(s_t, a') - Q_\theta(s_t, a_t))^2$
end for

---

[19] M. Riedmiller. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *Machine Learning: ECML 2005*. Springer, 2005, pp. 317–328.

Online Algorithms

- The deep Q-network algorithm, introduced by[20], is an online algorithm for neural fitted value iteration
  - Uses a replay pool—a rolling history used as data distribution
  - Uses a "target network" to represent the old Q-function, which we are doing backups on $Q_\theta \leftarrow BQ_{\text{target}}$
- Many extensions have been proposed since then[21]
- SARSA, which approximates B^π rather than B and is closer to policy iteration than value iteration, is found to work as well or better than DQN in some settings[22]

---

[20] V. Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[21] Z. Wang, N. de Freitas, and M. Lanctot. "Dueling Network Architectures for Deep Reinforcement Learning". In: *arXiv preprint arXiv:1511.06581* (2015); H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double Q-learning". In: *CoRR, abs/1509.06461* (2015); T. Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015); M. Hausknecht and P. Stone. "Deep recurrent Q-learning for partially observable MDPs". In: *arXiv preprint arXiv:1507.06527* (2015).

[22] V. Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *arXiv preprint arXiv:1602.01783* (2016).

Conclusion

Summary of the Current State of Affairs

- Policy gradient methods
  - Vanilla policy gradient (including A3C)
  - Natural policy gradient and trust region methods (including TRPO)
- Q-function methods
  - DQN and relatives: like value iteration, approximates B
  - SARSA: also found to perform well
- Comparison: Q-function methods are more sample efficient *when they work* but don't work as generally as policy gradient methods
  - Policy gradient methods easier to debug and understand

Summary of the Current State of Affairs

		Simple & Scalable	Data Efficient
Vanilla PG	OK	Good	Bad
Natural PG	Good	Bad	OK
Q-Learning	Bad	Good	OK

Still room for improvement!

Fin

Thank you. Questions?