Reinforcement Learning: Review

CS 330
Reminders

Today: Project proposals due

Monday next week: Homework 2 due, Homework 3 out
Why Reinforcement Learning?

Isolated action that doesn’t affect the future?

Common applications

- robotics
- language & dialog
- autonomous driving
- business operations
- finance
  (most deployed ML systems)
+ a key aspect of intelligence
The Plan

Reinforcement learning problem

Policy gradients

Q-learning
The Plan

Reinforcement learning problem

Policy gradients

Q-learning
object classification

supervised learning

iid data

large labeled, curated dataset

well-defined notions of success

object manipulation

sequential decision making

action affects next state

how to collect data?
what are the labels?

what does success mean?
Terminology & notation

- $s_t$ – state
- $o_t$ – observation

Slide adapted from Sergey Levine
Imitation Learning vs Reinforcement Learning?

$\pi_{\theta}(a_t|o_t)$
Reward functions

which action is better or worse?

$r(s, a)$: reward function

tells us which states and actions are better

$s, a, r(s, a)$, and $p(s'|s, a)$ define Markov decision process

high reward

low reward

Slide adapted from Sergey Levine
The goal of reinforcement learning

Slide adapted from Sergey Levine
The goal of reinforcement learning

- \( \pi_\theta(a|o) \)
- \( p(s'|s, a) \)
- Markov property independent of \( s_{t-1} \)
The goal of reinforcement learning

\[ \pi_\theta(a|s) \]

\[ p(s'|s, a) \]

\[ \pi_\theta(s_{t+1}|s_t, a_t) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t) \]

\[ \pi_\theta(\tau) \]

Markov property independent of \( s_{t-1} \)

\[ \theta^* = \arg \max_\theta \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \]

Slide adapted from Sergey Levine
What is a reinforcement learning task?

**Supervised learning**

A task:

\[ \mathcal{T}_i \triangleq \{ p_i(x), p_i(y|x), \mathcal{L}_i \} \]

data generating distributions, loss

**Reinforcement learning**

A task:

\[ \mathcal{T}_i \triangleq \{ \mathcal{S}_i, \mathcal{A}_i, p_i(s_1), p_i(s'|s, a), r_i(s, a) \} \]

action space, dynamics

state space

initial state distribution

reward

a Markov decision process

much more than the semantic meaning of task!
Examples Task Distributions

A task: \[ T_i \triangleq \{ S_i, A_i, p_i(s_1), p_i(s'|s, a), r_i(s, a) \} \]

Character animation:

- across maneuvers \( r_i(s, a) \) vary
- across garments & initial states \( p_i(s_1), p_i(s'|s, a) \) vary

Multi-robot RL:

\[ S_i, A_i, p_i(s_1), p_i(s'|s, a) \] vary
The Plan

Reinforcement learning problem

Policy gradients

Q-learning
The anatomy of a reinforcement learning algorithm

- generate samples (i.e. run the policy)
- fit a model to estimate return
  - compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)
  - fit $Q_\phi(s, a)$ (actor-critic, Q-learning)
  - estimate $p(s'|s, a)$ (model-based)
- improve the policy
  - $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)
  - $\pi(s) = \text{arg max } Q_\phi(s, a)$ (Q-learning)
  - optimize $\pi_\theta(a|s)$ (model-based)
Evaluating the objective

\[ \theta^* = \arg \max_\theta E_{\tau \sim \pi_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \]

\[ J(\theta) \]

\[ J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t}) \]

sum over samples from \( \pi_\theta \)
Direct policy differentiation

\[ \theta^* = \arg \max_{\theta} E_{\tau \sim \pi_{\theta}(\tau)} \left[ \sum_t r(s_t, a_t) \right] \]

\[ J(\theta) = \sum_{t=1}^{T} r(s_t, a_t) \]

\[ J(\theta) = E_{\tau \sim \pi_{\theta}(\tau)}[r(\tau)] = \int \pi_{\theta}(\tau)r(\tau)d\tau \]

\[ \nabla_\theta J(\theta) = \int \nabla_\theta \pi_{\theta}(\tau)r(\tau)d\tau = \int \pi_{\theta}(\tau) \nabla_\theta \log \pi_{\theta}(\tau)r(\tau)d\tau = E_{\tau \sim \pi_{\theta}(\tau)}[\nabla_\theta \log \pi_{\theta}(\tau)r(\tau)] \]

A convenient identity:

\[ \pi_{\theta}(\tau) \nabla_\theta \log \pi_{\theta}(\tau) = \pi_{\theta}(\tau) \frac{\nabla_\theta \pi_{\theta}(\tau)}{\pi_{\theta}(\tau)} = \nabla_\theta \pi_{\theta}(\tau) \]
Direct policy differentiation

$$\theta^* = \arg \max_{\theta} J(\theta)$$

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

$$\pi_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1) \prod_{t=1}^T \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$$

$$\log \pi_\theta(\tau) = \log p(s_1) + \sum_{t=1}^T \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)$$

$$\nabla_\theta \log p(s_1) + \sum_{t=1}^T \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)$$

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[ \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left( \sum_{t=1}^T r(s_t, a_t) \right) \right]$$

Slide adapted from Sergey Levine
Evaluating the policy gradient

recall: \( J(\theta) = E_{\tau \sim \pi_{\theta}(\tau)} \left[ \sum_t r(s_t, a_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t}) \)

\[ \nabla_\theta J(\theta) = E_{\tau \sim \pi_{\theta}(\tau)} \left[ \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left( \sum_{t=1}^T r(s_t, a_t) \right) \right] \]

\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left( \sum_{t=1}^T r(s_{i,t}, a_{i,t}) \right) \]

\[ \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \]

REINFORCE algorithm:
1. sample \( \{\tau^i\} \) from \( \pi_\theta(a_t|s_t) \) (run the policy)
2. \( \nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(a^i_t|s^i_t) \right) \left( \sum_t r(s^i_t, a^i_t) \right) \)
3. \( \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \)

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

Slide adapted from Sergey Levine
Comparison to maximum likelihood

Policy gradient: \[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left( \sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right) \]

Maximum likelihood: \[ \nabla_\theta J_{ML}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \]
What did we just do?

\[
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left( \sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right)
\]

\[
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau_i) r(\tau_i)
\]

maximum likelihood: \[
\nabla_\theta J_{ML}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau_i)
\]

good stuff is made more likely
bad stuff is made less likely
simply formalizes the notion of “trial and error”!

REINFORCE algorithm:
1. sample \( \{\tau_i\} \) from \( \pi_\theta(a_t|s_t) \) (run it on the robot)
2. \( \nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(a^i_t|s^i_t) \right) \left( \sum_t r(s^i_t, a^i_t) \right) \)
3. \( \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \)

Slide adapted from Sergey Levine
Policy Gradients

Policy gradient: \( \nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[ \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left( \sum_{t=1}^{T} r(s_t, a_t) \right) \right] \)

**Pros:**
- Simple
- Easy to combine with existing multi-task & meta-learning algorithms

**Cons:**
- Produces a high-variance gradient
  - Can be mitigated with baselines (used by all algorithms in practice), trust regions
- Requires on-policy data
  - Cannot reuse existing experience to estimate the gradient!
    - Importance weights can help, but also high variance
On-policy  vs  Off-policy

- Data comes from the current policy
- Compatible with all RL algorithms
- Can’t reuse data from previous policies

- Data comes from any policy
- Works with specific RL algorithms
- Much more sample efficient, can re-use old data
Small note

policy gradient: \[ \nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_{\theta} \log \pi_\theta(a_{i,t} | s_{i,t}) \right) \left( \sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right) \]

\[ \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_\theta(a_{i,t} | s_{i,t}) \left( \sum_{t' = t}^{T} r(a_{i,t'}, s_{i,t'}) \right) \]

\[ \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_\theta(a_{i,t} | s_{i,t}) \left( \sum_{t' = t+1}^{T} r(a_{i,t'}, s_{i,t'}) \right) \]

Reward “to go”
The Plan

Reinforcement learning problem

Policy gradients

Q-learning
The anatomy of a reinforcement learning algorithm

1. **generate samples (i.e. run the policy)**
2. **fit a model to estimate return**
   - compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)
   - fit $Q_{\phi}(s, a)$ (actor-critic, Q-learning)
   - estimate $p(s'|s, a)$ (model-based)
3. **improve the policy**
   - $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$ (policy gradient)
   - $\pi(s) = \arg \max Q_{\phi}(s, a)$ (Q-learning)
   - optimize $\pi_{\theta}(a|s)$ (model-based)

\[
\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t}|s_{i,t}) \left( \sum_{t'=t}^{T} r(a_{i,t'}, s_{i,t'}) \right)
\]
Improving the policy gradient

\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \left( \sum_{t'=t}^{T} r(a_{i,t'}, s_{i,t'}) \right) \]

Reward “to go”

\( \hat{Q}_{i,t} \)  
\( \hat{Q}_{i,t} \): estimate of expected reward if we take action \( a_{i,t} \) in state \( s_{i,t} \) can we get a better estimate?

\[ Q(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta} [r(s_{t'}, a_{t'}) | s_t, a_t] : \text{true expected reward-to-go} \]

\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) Q(s_{i,t}, a_{i,t}) \]

Slide adapted from Sergey Levine
State & state-action value functions

\[ Q(s_t, a_t) = \sum_{t=0}^{T} \mathbb{E}_{\pi(d)}[r(s_{t+1}, a_{t+1}) | s_t, a_t] : \text{total reward from taking } a_t \text{ in } s_t \]

\[ V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi(a_t | s_t)}[Q^\pi(s_t, a_t)] : \text{total reward from } s_t \]

\[ A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) : \text{how much better } a_t \text{ is} \]

\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) A^\pi(s_{i,t}, a_{i,t}) \]

fit \( Q^\pi, V^\pi, \) or \( A^\pi \)

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

\[ \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \]

Slide adapted from Sergey Levine
Value-Based RL

Value function: \( V^\pi(s_t) = ? \)

Q function: \( Q^\pi(s_t, a_t) = ? \)

Advantage function: \( A^\pi(s_t, a_t) = ? \)

Reward = 1 if I can play it in a month, 0 otherwise

Current \( \pi(a_1|s) = 1 \)
Multi-Step Prediction

\[
\hat{Q}_{i,t} \approx \left( \sum_{t'=t}^{T} r(a_{i,t'}, s_{i,t'}) \right)
\]

\[
\hat{Q}_{i,t} \approx \sum_{t'=t}^{T} E_{\pi_{\theta}} [r(s_{t', a_{t'}})|s_t, a_t]
\]

- How do you update your predictions about winning the game?
- What happens if you don’t finish the game?
- Do you always wait till the end?
How can we use all of this to fit a better estimator?

**Goal:** fit $V^\pi$

ideal target: $y_{i,t} = \sum_{t'=t}^{T} E_{\pi_\theta} [r(s_{i,t'}, a_{i,t'}) | s_{i,t}] \approx r(s_{i,t}, a_{i,t}) + \sum_{t'=t+1}^{T} E_{\pi_{\phi}}[\pi(s_{i,t}, a_{i,t}) | s_{i,t+1}] V^\pi_{\phi}(s_{i,t+1})$

Monte Carlo target: $y_{i,t} = \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$

directly use previous fitted value function!

training data: $\{ (s_{i,t}, r(s_{i,t}, a_{i,t}) + V^\pi_{\phi}(s_{i,t+1})) \}$

$y_{i,t}$

supervised regression: $L(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}^\pi_{\phi}(s_i) - y_i \right\|^2$

sometimes referred to as a “bootstrapped” estimate

Slide adapted from Sergey Levine
Policy evaluation examples

TD-Gammon, Gerald Tesauro 1992

Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal consensus in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-8, 6-5, to TD-Gammon’s preference, 13-8, 24-23. TD-Gammon’s analysis is given in Table 2.

reward: game outcome
value function $\hat{V}_\phi^\pi(s_t)$:
expected outcome given board state

AlphaGo, Silver et al. 2016

Figure 1. An illustration of the multilayer perceptron architecture used in TD-Gammon’s neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].

reward: game outcome
value function $\hat{V}_\phi^\pi(s_t)$:
expected outcome given board state
REINFORCE algorithm:
1. sample \( \{\tau^i\} \) from \( \pi_\theta(a_t|s_t) \) (run the policy)
2. \( \nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \right) \left( \sum_t r(s_t^i, a_t^i) \right) \)
3. \( \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \)

online actor-critic algorithm:
1. take action \( a \sim \pi_\theta(a|s) \), get \( (s, a, s', r) \)
2. update \( \hat{V}_\phi^\pi \) using target \( r + \gamma \hat{V}_\phi^\pi(s') \)
3. evaluate \( \hat{A}^\pi(s, a) = r(s, a) + \gamma \hat{V}_\phi^\pi(s') - \hat{V}_\phi^\pi(s) \)
4. \( \nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a|s) \hat{A}^\pi(s, a) \)
5. \( \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \)

fit a model to estimate return
generate samples (i.e. run the policy)

improve the policy

Slide adapted from Sergey Levine
This was just the prediction part...
Improving the Policy

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

how good is an action compared to the policy?

\[
\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \text{ (policy gradient)}
\]

\[
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left( \sum_{t'=t}^{T} r(a_{i,t'}, s_{i,t'}) \right)
\]

\[
\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a|s) \hat{A}^\pi(s, a)
\]

fit $V^\pi(s_t)$

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

$\pi \leftarrow \pi'$
Value-Based RL

Value function: $V^\pi(s_t) =$ ?
Q function: $Q^\pi(s_t, a_t) =$ ?
Advantage function: $A^\pi(s_t, a_t) =$ ?

Reward = 1 if I can play it in a month, 0 otherwise

How can we improve the policy?

Current $\pi(a_1|s) = 1$
Improving the Policy

$A^\pi(s_t, a_t)$: how much better is $a_t$ than the average action according to $\pi$

$\arg \max_{a_t} A^\pi(s_t, a_t)$: best action from $s_t$, if we then follow $\pi$

$\pi'(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \arg \max_{a_t} A^\pi(s_t, a_t) \\ 0 & \text{otherwise} \end{cases}$

at least as good as any $a_t \sim \pi(a_t|s_t)$

regardless of what $\pi(a_t|s_t)$ is!
Policy Iteration

policy iteration algorithm:
1. evaluate $A^\pi(s, a)$
2. set $\pi \leftarrow \pi'$

$\pi'(a_t|s_t) = \begin{cases} 
1 & \text{if } a_t = \arg\max_{a_t} A^\pi(s_t, a_t) \\
0 & \text{otherwise}
\end{cases}$

as before: $A^\pi(s, a) = r(s, a) + \gamma E[V^\pi(s')] - V^\pi(s)$

Slide adapted from Sergey Levine
Value Iteration

policy iteration algorithm:
1. evaluate $Q^\pi(s, a)$
2. set $\pi \leftarrow \pi'$

$$
\pi'(a_t|s_t) = \begin{cases} 
1 & \text{if } a_t = \arg\max_a Q^\pi(s, a) \\
0 & \text{otherwise}
\end{cases}
$$

$$
A^\pi(s, a) = r(s, a) + \gamma E[V^\pi(s')] - V^\pi(s)
$$

$$
\arg\max_{a_t} A^\pi(s_t, a_t) = \arg\max_a Q^\pi(s_t, a_t)
$$

$$
Q^\pi(s, a) = r(s, a) + \gamma E[V^\pi(s')] \text{ (a bit simpler)}
$$

approximates the new value!

arg $\max_a Q(s, a) \rightarrow$ policy

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

$$
V^\pi(s) \leftarrow \max_a Q^\pi(s, a)
$$

skip the policy and compute values directly!

value iteration algorithm:
1. set $Q(s, a) \leftarrow r(s, a) + \gamma E[V(s')]$
2. set $V(s) \leftarrow \max_a Q(s, a)$

Slide adapted from Sergey Levine
Q learning

\[ \pi'(a_t|s_t) = \begin{cases} 
1 & \text{if } a_t = \arg \max_{a_t} Q^\pi(s, a) \\
0 & \text{otherwise}
\end{cases} \]

doesn’t require simulation of actions!

value iteration algorithm:

1. set \( Q(s, a) \leftarrow r(s, a) + \gamma E[V(s')] \)
2. set \( V(s) \leftarrow \max_a Q(s, a) \)

fitted Q iteration algorithm:

1. set \( y_i \leftarrow r(s_i, a_i) + \gamma E[V_\phi(s'_i)] \) approximate \( E[V(s'_i)] \approx \max_{a'_i} Q_\phi(s'_i, a'_i) \)
2. set \( \phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2 \) doesn’t require simulation of actions!

\[ Q^\pi(s, a) \leftarrow r(s, a) + \gamma E_{s' \sim p(s'|s, a)}[V^\pi(s')] \]
Value-Based RL

Value function: $V^\pi(s_t) = ?$

Q function: $Q^\pi(s_t, a_t) = ?$

Q* function: $Q^*(s_t, a_t) = ?$

Value* function: $V^*(s_t) = ?$

Reward = 1 if I can play it in a month, 0 otherwise

Current $\pi(a_1|s) = 1$
Fitted Q-iteration Algorithm

full fitted Q-iteration algorithm:

1. collect dataset \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy

\[ y_i \leftarrow r(s_i, a_i) + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i) \]

2. set \( y_i \leftarrow r(s_i, a_i) + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i) \)

3. set \( \phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2 \)

Result: get a policy \( \pi(a|s) \) from \( \arg \max_a Q_\phi(s, a) \)

Important notes:

We can **reuse data** from previous policies!

an **off-policy** algorithm using replay buffers

This is **not** a **gradient descent** algorithm!

Can be readily extended to **multi-task/goal-conditioned RL**

Slide adapted from Sergey Levine
Example: Q-learning Applied to Robotics

1. collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using some policy

2. set $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$

3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

Continuous action space?

Simple optimization algorithm ->

Cross Entropy Method (CEM)

---

1. Start with the normal distribution $N(\mu, \sigma^2)$.

2. Evaluate some parameters from this distribution and select the best (in grey).

3. Compute the mean and std.dev. of the best, add some noise and go to 1.
QT-Ops: Q-learning at Scale

In-memory buffers:
- off-policy \((s, a, s', r')\)
- on-policy \((s, a, s', r)\)
- labeled \((s, a, Q_T(s, a))\)

Bellman updaters:
compute \(Q_T(s, a) = r + \max_{a'} Q_\theta(s', a')\)

Training jobs:
\[
\min_{\theta} ||Q_\theta(s, a) - Q_T(s, a)||^2
\]

minimize \(\sum_i (Q(s_i, a_i) - [r(s_i, a_i) + \max_{a'_i} Q(s'_i, a'_i)])^2\)

Slide adapted from D. Kalashnikov

QT-Ops: Kalashnikov et al. '18, Google Brain
QT-Opt: Setup and Results

7 robots collected 580k grasps

Unseen test objects

96% test success rate!
Q-learning

Bellman equation:  
\[ Q^*(s_t, a_t) = \mathbb{E}_{s' \sim p(\cdot|s, a)} \left[ r(s, a) + \gamma \max_{a'} Q^*(s', a') \right] \]

**Pros:**
+ More sample efficient than on-policy methods
+ Can incorporate off-policy data (including a fully offline setting)
+ Can updates the policy even without seeing the reward
+ Relatively easy to parallelize

**Cons:**
- Lots of “tricks” to make it work
- Potentially could be harder to learn than just a policy
The Plan

Reinforcement learning problem

Policy gradients

Q-learning
Additional RL Resources

Stanford CS234: Reinforcement Learning
UCL Course from David Silver: Reinforcement Learning
Berkeley CS285: Deep Reinforcement Learning