Reinforcement Learning
Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley
Recap: policy gradients

REINFORCE algorithm:

1. sample \( \{\tau^i\} \) from \( \pi_\theta(a_t|s_t) \) (run the policy)
2. \( \nabla_\theta J(\theta) \approx \sum_i \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \left( \sum_{t'=t}^{T} r(s_{t'}, a_{t'}^i) \right) \right) \)
3. \( \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \)

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

"reward to go"

\[
\hat{Q}_t^\pi(x_t, u_t) = \sum_{t'=t}^{T} r(x_{t'}, u_{t'})
\]
Improving the policy gradient

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

"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?

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

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

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

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

\[ b_t = \frac{1}{N} \sum_{i} Q(s_{i,t}, a_{i,t}) \quad \text{average what?} \]

\[ V(s_t) = E_{a_t \sim \pi_\theta(a_t | s_t)}[Q(s_t, a_t)] \]
State & state-action value functions

\[
Q(s_t, a_t) = \sum_{t'=t}^{T} \mathbb{E}_{\tau \sim \pi_\theta} \left[ r(s_{t'}, a_{t'}) + \gamma \max_{a_{t'}} Q(s_{t'}, a_{t'}) \right] \quad \text{total reward from taking } a_t \text{ in } s_t
\]

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

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

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

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

the better this estimate, the lower the variance

unbiased, but high variance single-sample estimate

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

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

\[ \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \]
Value function fitting

\[ Q^\pi(s_t, a_t) = \sum_{t'=t}^T E_{\pi^\theta}[r(s_{t'}, a_{t'})|s_t, a_t] \]

\[ V^\pi(s_t) = E_{a_t \sim \pi^\theta(a_t|s_t)}[Q^\pi(s_t, a_t)] \]

\[ A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \]

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

fit what to what?

\[ Q^\pi, V^\pi, A^\pi? \]

\[ Q^\pi(s_t, a_t) \approx \sum_{t'=t}^T E_{\pi^\theta}[r(s_{t'}, a_{t'})|s_t, a_t] + E_{\pi^\theta}[A^\pi(s_{t'}, a_{t'})|s_t, a_t] \]

\[ A^\pi(s_t, a_t) \approx r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t) \]

let’s just fit \( V^\pi(s) \)!
Policy evaluation

\[ V^\pi(s_t) = \sum_{t'=t}^T E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t] \]

\[ J(\theta) = E_{s_1 \sim p(s_1)}[V^\pi(s_1)] \]

how can we perform policy evaluation?

Monte Carlo policy evaluation (this is what policy gradient does)

\[ V^\pi(s_t) \approx \sum_{t'=t}^T r(s_{t'}, a_{t'}) \]

\[ V^\pi(s_t) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t'=t}^T r(s_{t'}, a_{t'}) \] (requires us to reset the simulator)
Monte Carlo evaluation with function approximation

\[ V^\pi(s_t) \approx \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \]

not as good as this: \[ V^\pi(s_t) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \]
but still pretty good!

training data: \( \left\{ \left( s_{i,t}, \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right) \right\} \)

\( y_{i,t} \)

supervised regression: \[ \mathcal{L}(\phi) = \frac{1}{2} \sum_{i} \left\| \hat{V}^\pi(\phi)(s_i) - y_i \right\|^2 \]

the same function should fit multiple samples!
Can we do better?

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

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: \( \left\{ \left( s_{i,t}, r(s_{i,t}, a_{i,t}) + \hat{V}_{\pi} (s_{i,t+1}) \right) \right\} \)

\( y_{i,t} \)

supervised regression: \( \mathcal{L}(\phi) = \frac{1}{2} \sum_{i} \left\| \hat{V}_{\phi} (s_{i}) - y_{i} \right\|^2 \)

sometimes referred to as a “bootstrapped” estimate
Policy evaluation examples

TD-Gammon, Gerald Tesauro 1992

- An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, expert players have now switched from the traditional move of TD-Gammon's preference, 13-12, 24-25. TD-Gammon's analysis is given in Table 2.

- 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 [3].

- Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, expert players have now switched from the traditional move of TD-Gammon's preference, 13-12, 24-25. TD-Gammon's analysis is given in Table 2.

- 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 [3].

AlphaGo, Silver et al. 2016

- 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 [3].

- Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, expert players have now switched from the traditional move of TD-Gammon's preference, 13-12, 24-25. TD-Gammon's analysis is given in Table 2.

- 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 [3].

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

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

batch actor-critic algorithm:

1. sample \( \{s_i, a_i\} \) from \( \pi_\theta(a|s) \) (run it on the robot)
2. fit \( \hat{V}_\theta(s) \) to sampled reward sums
3. evaluate \( \hat{A}_\pi(s_i, a_i) = r(s_i, a_i) + \hat{V}_\phi(s'_i) - \hat{V}_\phi(s_i) \)
4. \( \nabla_{\theta} J(\theta) \approx \sum_i \nabla_{\theta} \log \pi_\theta(a_i|s_i) \hat{A}_\pi(s_i, a_i) \)
5. \( \theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta) \)

\[
V_\pi(s_{i,i}) \approx \sum_{T_{i,i}}^T \mathbb{I}(s_i \in \mathcal{S}) \mathbb{I}(a_i \in \mathcal{A}) \mathbb{I}(\hat{V}_\phi(s_i) \approx \hat{V}_\phi(s_{i,i})
\]

\[
\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi(s_i) - y_i \right\|^2
\]

\[
V_\pi(s_t) = \sum_{t' = t}^T E_{\pi_\theta} [r(s_{t'}, a_{t'})|s_t]
\]

\[\text{fit } \hat{V}_\pi \]

\[\text{fit a model to estimate return}
\]

\[\text{generate samples (i.e. run the policy)}
\]

\[\text{improve the policy}
\]

\[\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)\]
Aside: discount factors

\[ y_{i,t} \approx r(s_{i,t}, a_{i,t}) + \hat{V}_\pi(s_{i,t+1}) \]

\[ \mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi(s_i) - y_i \right\|^2 \]

what if \( T \) (episode length) is \( \infty \)?
\( \hat{V}_\pi \) can get infinitely large in many cases

simple trick: better to get rewards sooner than later

\[ y_{i,t} \approx r(s_{i,t}, a_{i,t}) + \gamma \hat{V}_\pi(s_{i,t+1}) \]

discount factor \( \gamma \in [0, 1] \) (0.99 works well)

\( \gamma \) changes the MDP:

\[ \hat{p}(s'|s, a) = (1 - \gamma) \]

\[ \tilde{p}(s'|s, a) = \gamma \tilde{p}(s'|s, a) \]
Actor-critic algorithms (with discount)

batch actor-critic algorithm:
1. sample \( \{s_i, a_i\} \) from \( \pi_\theta(a|s) \) (run it on the robot)
2. fit \( \hat{V}_\phi^\pi(s) \) to sampled reward sums
3. evaluate \( \hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \gamma \hat{V}_\phi^\pi(s'_i) - \hat{V}_\phi^\pi(s_i) \)
4. \( \nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \hat{A}^\pi(s_i, a_i) \)
5. \( \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \)

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. \( \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \)
Architecture design

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 $\hat{A}^\pi(s, a) = r(s, a) + \gamma \hat{V}_\phi^\pi(s') - \hat{V}_\phi(s)$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a|s) \hat{A}^\pi(s, a)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

+ simple & stable
- no shared features between actor & critic
Can we use just a value function?
Can we omit policy gradient completely?

$A^\pi(s_t, a_t)$: 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 $\pi$ at least as good as any $a_t \sim \pi(a_t|s_t)$ regardless of what $\pi(a_t|s_t)$ is!

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

as good as $\pi$ (probably better)

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

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

$\pi \leftarrow \pi'$
Policy iteration

High level idea:

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

let’s evaluate \( V^\pi(s) \)!
Dynamic programming

Let’s assume we know \( p(s'|s, a) \), and \( s \) and \( a \) are both discrete (and small)

<table>
<thead>
<tr>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td>0.4</td>
<td>0.4</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.7</td>
<td>0.5</td>
</tr>
</tbody>
</table>

16 states, 4 actions per state

can store full \( V^\pi(s) \) in a table!

\( T \) is \( 16 \times 16 \times 4 \) tensor

bootstrapped update: \( V^\pi(s) \leftarrow E_{a \sim \pi(a|s)}[r(s, a) + \gamma E_{s' \sim p(s'|s, a)}[V^\pi(s')]] \)

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

simplified: \( V^\pi(s) \leftarrow r(s, \pi(s)) + \gamma E_{s' \sim p(s'|s,\pi(s))}[V^\pi(s')] \)
Policy iteration with dynamic programming

Policy iteration:
1. evaluate $V^\pi(s)$
2. set $\pi \leftarrow \pi'$

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

Policy evaluation:
$V^\pi(s) \leftarrow r(s, \pi(s)) + \gamma \mathbb{E}_{s' \sim p(s'|s, \pi(s))}[V^\pi(s')]$

16 states, 4 actions per state
- can store full $V^\pi(s)$ in a table!
- $T$ is $16 \times 16 \times 4$ tensor
Even simpler dynamic programming

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

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

\[ \arg \max_{a_t} A^\pi(s_t, a_t) = \arg \max_{a_t} Q^\pi(s_t, a_t) \]

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

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 \( V(s) \leftarrow \max_a Q(s, a) \)

arg \max_a Q(s, a) → policy

approximates the new value!

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

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

\[ V^\pi(s) \leftarrow \max_a Q^\pi(s, a) \]
Fitted value iteration

how do we represent $V(s)$?
big table, one entry for each discrete $s$
normal net function $V: \mathcal{S} \rightarrow \mathbb{R}$

$s$ $V(s)$

parameters $\phi$

\[ \mathcal{L}(\phi) = \frac{1}{2} \left\| V_\phi(s) - \max_a Q^\pi(s, a) \right\|^2 \]

fitted value iteration algorithm:
1. set $y_i \leftarrow \max_a (r(s_i, a_i) + \gamma E[V_\phi(s_i')]])$
2. set $\phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i \|V_\phi(s_i) - y_i\|^2$

$s = 0: V(s) = 0.2$
$s = 1: V(s) = 0.3$
$s = 2: V(s) = 0.5$

$|\mathcal{S}| = (255^3)^{200 \times 200}$
(more than atoms in the universe)

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

improve the policy

generate samples (i.e. run the policy)

fit a model to estimate return

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

curse of dimensionality
What if we don’t know the transition dynamics?

fitted value iteration algorithm:
1. set $y_i \leftarrow \max_{a_i} (r(s_i, a_i) + \gamma E[V_{\phi}(s'_i)])$
2. set $\phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i \|V_{\phi}(s_i) - y_i\|^2$

need to know outcomes for different actions!

Back to policy iteration...

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

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

policy evaluation:

$V^\pi(s) \leftarrow r(s, \pi(s)) + \gamma \mathbb{E}_{s' \sim p(s'|s, \pi(s))}[V^\pi(s')]$

$Q^\pi(s, a) \leftarrow r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)}[Q^\pi(s', \pi(s'))]$ can fit this using samples
Can we do the “max” trick again?

policy iteration:
1. evaluate $V^\pi(s)$
2. set $\pi \leftarrow \pi'$

fitted value iteration algorithm:
1. set $y_i \leftarrow \max_{a_i} (r(s_i, a_i) + \gamma E[V_\phi(s'_i)])$
2. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|V_\phi(s_i) - y_i\|^2$

forget policy, compute value directly

can we do this with Q-values also, without knowing the transitions?

fitted Q iteration algorithm:
1. set $y_i \leftarrow r(s_i, a_i) + \gamma E[V_\phi(s'_i)]$  
   approximate $E[V(s'_i)] \approx \max_{a'} Q_\phi(s'_i, a'_i)$
2. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

+ works even for off-policy samples (unlike actor-critic)
+ only one network, no high-variance policy gradient
- no convergence guarantees for non-linear function approximation (more on this later)
Fitted Q-iteration

full fitted Q-iteration algorithm:

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

parameters

dataset size \(N\), collection policy

iterations \(K\)

gradient steps \(S\)

\(s\)

\(a\)

\(Q_\phi(s, a)\)

parameters \(\phi\)
Q-Learning
Online Q-learning algorithms

full fitted Q-iteration algorithm:
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 \( \phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i \| Q_\phi(s_i, a_i) - y_i \|^2 \)

online Q iteration algorithm:
1. take some action \( a_i \) and observe \( (s_i, a_i, s'_i, r_i) \)
2. \( y_i = r(s_i, a_i) + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i) \)
3. \( \phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i) \)

Q_\phi(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q_\phi(s', a')

fit a model to estimate return

generate samples (i.e. run the policy)

improve the policy

a = \arg\max_a Q_\phi(s, a)

off policy, so many choices here!
Exploration with Q-learning

online Q iteration algorithm:
1. take some action $a_i$ and observe $(s_i, a_i, s'_i, r_i)$
2. $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i)$

$\pi(a_t|s_t) = \begin{cases} 
1 - \epsilon & \text{if } a_t = \arg\max_a Q_\phi(s_t, a_t) \\
\epsilon/(|A| - 1) & \text{otherwise}
\end{cases}$

$\pi(a_t|s_t) \propto \exp(Q_\phi(s_t, a_t))$

final policy:
$\pi(a_t|s_t) = \begin{cases} 
1 & \text{if } a_t = \arg\max_a Q_\phi(s_t, a_t) \\
0 & \text{otherwise}
\end{cases}$

why is this a bad idea for step 1?

“epsilon-greedy”

“Boltzmann exploration”
What’s wrong?

online $Q$ iteration algorithm:

1. take some action $a_i$ and observe $(s_i, a_i, s'_i, r_i)$
2. $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i)$

these are correlated!

isn’t this just gradient descent? that converges, right?

Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)])$$

no gradient through target value
Correlated samples in online Q-learning

1. take some action $a_i$ and observe $(s_i, a_i, s'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_{\phi}}{d\phi} (s_i, a_i)(Q_{\phi}(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a' i} Q_{\phi}(s'_i, a'_i)])$

- sequential states are strongly correlated
- target value is always changing
Replay buffers

online Q iteration algorithm:
1. take some action $a_i$ and observe $(s_i, a_i, s'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi} (s_i, a_i) (Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)])$

full fitted Q-iteration algorithm:
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 $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2$

special case with $K = 1$, and one gradient step
any policy will work! (with broad support)
just load data from a buffer here
still use one gradient step

Fitted Q-iteration

dataset of transitions
Replay buffers

Q-learning with a replay buffer:

1. sample a batch \((s_i, a_i, s'_i, r_i)\) from \(B\)

2. \(\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)])\)

+ multiple samples in the batch (low-variance gradient)

but where does the data come from?

need to periodically feed the replay buffer...

\[(s, a, s', r)\]

dataset of transitions ("replay buffer")

\(\pi(a|s)\) (e.g., \(\epsilon\)-greedy)

off-policy Q-learning
Putting it together

full Q-learning with replay buffer:

1. collect dataset \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy, add it to \( \mathcal{B} \)

2. sample a batch \( (s_i, a_i, s'_i, r_i) \) from \( \mathcal{B} \)

3. \( \phi \leftarrow \phi - \alpha \sum_i \left( \frac{dQ\phi}{d\phi}(s_i, a_i)(Q\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q\phi(s'_i, a'_i)]) \right) \)

K = 1 is common, though larger K more efficient
What’s wrong?

online Q iteration algorithm:
1. take some action $a_i$ and observe $(s_i, a_i, s'_i, r_i)$
2. $y_i = r(s_i, a_i) + \gamma \max_{a'} Q(\phi)(s'_i, a'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ(\phi)}{d\phi}(s_i, a_i)(Q(\phi)(s_i, a_i) - y_i)$

Q-learning is *not* gradient descent!

$\phi \leftarrow \phi - \alpha \frac{dQ(\phi)}{d\phi}(s_i, a_i)(Q(\phi)(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q(\phi)(s'_i, a'_i)])$

This is still a problem!

no gradient through target value

these are correlated!

use replay buffer
Q-Learning and Regression

full Q-learning with replay buffer:

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

2. sample a batch \( (s_i, a_i, s'_i, r_i) \) from \( B \)

3. \begin{align*}
\phi &\leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)])
\end{align*}

one gradient step, moving target

full fitted Q-iteration algorithm:

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

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

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

perfectly well-defined, stable regression
Q-Learning with target networks

Q-learning with replay buffer and target network:

1. save target network parameters: \( \phi' \leftarrow \phi \)

2. collect dataset \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy, add it to \( B \)

3. sample a batch \((s_i, a_i, s'_i, r_i)\) from \( B \)

4. \( \phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s'_i, a'_i)]) \)

\( \) targets don’t change in inner loop!
“Classic” deep Q-learning algorithm (DQN)

Q-learning with replay buffer and target network:
1. save target network parameters: $\phi' \leftarrow \phi$
2. collect dataset $\{(s_i, a_i, s'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
3. sample a batch $(s_i, a_i, s'_i, r_i)$ from $\mathcal{B}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s'_i, a'_i)])$

“classic” deep Q-learning algorithm:
1. take some action $a_i$ and observe $(s_i, a_i, s'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{s_j, a_j, s'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{a'j} Q_{\phi'}(s'_j, a'_j)$ using target network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(s_j, a_j)(Q_\phi(s_j, a_j) - y_j)$
5. update $\phi'$: copy $\phi$ every $N$ steps

Mnih et al. ‘13
Representing the Q-function

\[ \hat{Q}_\phi(s) \] more common with continuous actions

\[ \hat{Q}_\phi(s, a_1) \]

\[ \hat{Q}_\phi(s, a_2) \]

\[ \hat{Q}_\phi(s, a_3) \] more common with discrete actions
Back to actor-critic

online Q iteration algorithm:

1. take some action $a_i$ and observe $(s_i, a_i, s'_i, r_i)$
2. $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_{\phi}(s'_i, a'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_{\phi}}{d\phi}(s_i, a_i)(Q_{\phi}(s_i, a_i) - y_i)$

off policy, so many choices here!

with **continuous actions**, this is very inconvenient (but not impossible)

**Idea:** use actor-critic, but with Q-functions (to train off-policy)

1. take some action $a_i$ and observe $(s_i, a_i, s'_i, r_i)$, add it to $B$
2. sample mini-batch $\{s_j, a_j, s'_j, r_j\}$ from $B$ uniformly
3. compute $y_j = r_j + \gamma E_{a'_j \sim \pi'_\theta(a'_j|s'_j)}[Q_{\phi'}(s'_j, a'_j)]$ using target $\phi'$ and $\theta'$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_{\phi}}{d\phi}(s_j, a_j)(Q_{\phi}(s_j, a_j) - y_j)$
5. $\theta \leftarrow \theta + \beta \sum_j \nabla_{\theta} E_{a \sim \pi_{\theta}(a|s_j)}[Q_{\phi}(s_j, a)]$ ← policy gradient
6. update $\phi'$ and $\theta'$ every $N$ steps
Simple practical tips for Q-learning

• Q-learning takes some care to stabilize
  • Test on easy, reliable tasks first, make sure your implementation is correct

• Large replay buffers help improve stability
  • Looks more like fitted Q-iteration

• It takes time, be patient – might be no better than random for a while

• Start with high exploration (epsilon) and gradually reduce

*Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. “Prioritized experience replay”. arXiv preprint arXiv:1511.05952 (2015), Figure 7*

*Slide partly borrowed from J. Schulman*
Q-learning with convolutional networks

- “Human-level control through deep reinforcement learning,” Mnih et al. ‘13
- Q-learning with convolutional networks
- Uses replay buffer and target network
- One-step backup
- One gradient step
- Can be improved a lot with double Q-learning (and other tricks)
Large-scale Q-learning with continuous actions (QT-Opt)

stored data from all past experiments \( \{(s_i, a_i, s'_i)\}_i \)

live data collection

trading 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 threads

\[
\min_\theta ||Q_\theta(s, a) - Q_T(s, a)||^2
\]

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

• Classic papers

• Deep reinforcement learning Q-learning papers
  • Mnih et al. (2013). Human-level control through deep reinforcement learning: Q-learning with convolutional networks for playing Atari.
  • Lillicrap et al. (2016). Continuous control with deep reinforcement learning: continuous Q-learning with actor network for approximate maximization.