Q-Learning and Policy Gradients

Lecture 17
What will you take home today?

---

Reinforcement Learning

- Recap of Q-Learning
- Deep Q-Learning
- Policy Gradients

Representation Learning
The Reinforcement Learning Problem

From 'Optimal and Learning Control' by Buchli et al.
The Reinforcement Learning Problem

\[ R_0 = r_0 + \alpha r_1 + \alpha^2 r_2 + \cdots + \alpha^n r_n + \cdots = \sum_{k=0}^{\infty} \alpha^k r_k \]

\[ \pi^* = \arg \max_{\pi} E[R_0] \]

\[ R_n = \sum_{k=0}^{\infty} \alpha^k r_{n+k} \]
Optimal policy

Value function

\[ V^*(x) \geq V^{\pi}(x) \]

\[ V^*(x) = \max_{\pi} V^{\pi}(x) \]

Q-function

\[ Q^*(x, u) = \max_{\pi} Q^{\pi}(x, u) \]

Relation

\[ V^*(x) = \sum_{\pi} \pi^*(x, u) Q^*(x, u) \]
Policy Iteration

\[ V \rightarrow V^\pi \rightarrow \pi \rightarrow \text{greedy}(V) \rightarrow \text{evaluation} \rightarrow \ldots \]

Assume:
- We know the reward function per state
- We know the transition function

From "Optimal and Learning Control" by Buchli et al.
Sample-based RL

Directly estimate the Q function from samples

\( \Pi^*(x) = \text{argmax}_u Q^*(x,u) \)

---

From ‘Optimal and Learning Control’ by Buchli et al.

---

Algorithm 2 Policy Iteration

1. Initialization
   select \( V(x) \in \mathbb{R} \) and \( \pi(x) \in U \) arbitrarily for all \( x \in X \)

2. Policy evaluation
   repeat
   \[ \Delta \leftarrow 0 \]
   for each: \( x \in X \)
   \[ v \leftarrow V(x) \]
   \[ V(x) \leftarrow \sum_u \pi(x,u) \sum_{x'} \mathbb{P}_{x \rightarrow x'}^u [R_{x \rightarrow x'}^u + \alpha V(x')] \]
   \[ \Delta \leftarrow \max(\Delta, |v - V(x)|) \]
   until \( \Delta < \theta \) (a small positive number)

3. Policy Improvement
   \( \text{policyIsStable} \leftarrow \text{true} \)
   for \( x \in X \) do
   \[ b \leftarrow \pi(x) \]
   \[ \pi(x) \leftarrow \text{argmax}_u \sum_{x'} \mathbb{P}_{x \rightarrow x'}^u [R_{x \rightarrow x'}^u + \alpha V(x')] \]
   if \( b \neq \pi(x) \) then
   \[ \text{policyIsStable} \leftarrow \text{false} \]
   end if
   end for
   if \( \text{policyIsStable} \) then
   stop
   else
   go to 2
   end if

Return: a policy, \( \pi \), such that: \( \pi(x) = \text{argmax}_u \sum_{x'} \mathbb{P}_{x \rightarrow x'}^u [R_{x \rightarrow x'}^u + \alpha V(x')] \)
Policy Iteration with Q-function

1. Model-free Policy Improvement

Given $Q^\pi(x, u)$

$\pi(x) = \arg\max_u Q^\pi(x, u)$

Does not require to know $Q^\pi(x, u)$

Follows from Policy Improvement Theorem

3. Policy Improvement

```plaintext
policyIsStable ← true
for $x \in X$ do
    $b ← \pi(x)$
    $\pi(x) ← \arg\max_u \sum_x [P_{x, x'} R_{x, x'}^u + \alpha V(x')]$
    if $b \neq \pi(x)$ then
        policyIsStable ← false
    end if
end for
if policyIsStable then
    stop
else
    go to 2
end if
```
Policy Iteration with $Q$-function

- Model-free Policy Evaluation

\[
\tilde{Q}^\pi(x,u) = E_{x_{n+1}} \left[ R_n \mid x_n = x, u_n = u \right]
\]

\[
\tilde{Q}_N^\pi(x,u) = \frac{1}{N} \sum_{i=1}^{N} r_i^\pi(x,u) + \alpha r_{i+1}^\pi(x,u) + \alpha^2 r_{i+2}^\pi(x,u) \ldots
\]

- Inefficient

\[
\tilde{Q}_{N+1}^\pi(x,u) = \tilde{Q}_N^\pi(x,u) + \frac{1}{N+\lambda} \left( R_n^u - \tilde{Q}_N^\pi(x,u) \right)
\]

- Recursive

2. Policy evaluation

repeat

\[
\Delta \leftarrow 0
\]

for each: \( x \in X \)

\[
v \leftarrow V(x)
\]

\[
V(x) \leftarrow \sum_{u} \pi(x,u) \sum_{x'} P_{xx'}^{u} \left[ R_n^u + \alpha V(x') \right]
\]

\[
\Delta \leftarrow \max(\Delta, |v - V(x)|)
\]

until \( \Delta < \theta \) (a small positive number)

From ‘Optimal and Learning Control’ by Buchli et al.
Sample-based RL

1. Estimate the Q function instead of the value function.
2. Replace Bellman update by numerical averaging over samples.
3. Use the estimated Q function to derive an improved policy (greedy).

Some challenges: maintain exploration → epsilon greedy.
Q-Learning Combines Sample-based RL and Dynamic Programming

Bellman Equation for the optimal Q-function

\[ Q^*(x_n, u_n) = E \left\{ r_n + \alpha \max_{u'} Q^*(x', u') \mid x_n = x, u_n = u \right\} \]

\[ \hat{Q}(x, u) = \frac{1}{N} \sum_{i=1}^{N} \left( r_i + \alpha \max_u Q(x', u) \right) \]

- does not consider the entire tail of an episode
Q-Learning Combines Sample-based RL and Dynamic Programming

\[ \tilde{Q}^{i+1}(x_n, u_n) = \tilde{Q}^i(x_n, u_n) + \omega_{i+1} \left[ r_n^{i+1} + \alpha \max_{u_n'} \tilde{Q}^i(x'_n, u'_n) - \tilde{Q}^i(x_n, u_n) \right] \]

\[ \tilde{Q}_{N+1}^\pi(x, u) = \tilde{Q}_N^\pi(x, u) + \omega_{N+1} \cdot (R_{N+1} - \tilde{Q}_N^\pi(x, u)) \]

Recursive update for Q-learning

Sample-based RL
Q-Learning Algorithm

Algorithm 7 Q-Learning

Initialize $Q(x, u)$ arbitrarily

Repeat for each episode:

Initialize $x$

repeat (for each step of episode):

Choose $u$ from $x$ using policy derived from $Q$

(e.g., $\varepsilon$-greedy)

Take action $u$, observe $r, x'$

$Q(x, u) \leftarrow Q(x, u) + \omega[r + \gamma \max_{u'} Q(x', u') - Q(x, u)]$

$x \leftarrow x'$

until $x$ is terminal

<--- ensuring same level of exploration
<--- update rule from previous slide
What is Deep RL or Deep Q-Learning?

\[ Q(s, a; \theta) \approx Q^*(s, a) \]

Use a function approximator that approximates the Q function

\[ \Rightarrow \text{NN} \]
Deep Q-Learning

\[ Q^*(s, a) = \mathbb{E}_{s' \sim \epsilon} \left[ r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right] \]

Forward Pass:

Loss function: \[ \mathcal{L}_i(\theta_i) = \mathbb{E}_{s, a \sim p} \left[ (y_i - Q(s, a; \theta_i))^2 \right] \]

where \[ y_i = \mathbb{E}_{s' \sim \epsilon} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \mid s, a \right] \]

Target values

\[ Q(x, u) \leftarrow Q(x, u) + \omega [r + \gamma \max_u Q(x', u') - Q(x, u)] \]

\[ x \leftarrow x' \]
Deep Q-Learning

**Backward Pass**: Gradient update w.r.t. the Q function parameters $\Theta$

$$\nabla_{\Theta_i} L_i(\Theta_i) = \mathbb{E}_{s,a \sim p(.; \Theta_i), s' \sim \mathcal{E}} \left[ (r + \gamma \max_{a'} Q(s', a'; \Theta_{i-1}) - Q(s, a; \Theta_i)) \nabla_{\Theta_i} Q(s, a; \Theta_i) \right]$$

*Target values* $y_i$
Case Study: Playing Atari

**Objective:** Complete the game with the highest score

**State:** Raw pixel inputs of the game state

**Action:** Game controls e.g. Left, Right, Up, Down

**Reward:** Score increase/decrease at each time step
Q-network Architecture

$Q(s, a; \theta)$: neural network with weights $\theta$

- FC-4 (Q-values)
- FC-256
- 32 4x4 conv, stride 2
- 16 8x8 conv, stride 4

Current state $s_t$: 84x84x4 stack of last 4 frames (after RGB->grayscale conversion, downsampling, and cropping)

[Mnih et al. NIPS Workshop 2013; Nature 2015]
Experience Replay

Q-learning is offline but policy determines what states are visited & updated

- Samples are correlated

- Updates a memory of transitions (s<sub>t</sub>, a<sub>t</sub>, r<sub>t</sub>, s<sub>t+1</sub>) from gameplay experience

- Train Q-networks: sample random minibatches from this memory

- Not only updating from consecutive samples

- Creates sample efficiency
Q-Learning with Experience Replay

Algorithm 1: Deep Q-learning with Experience Replay

1. Initialize replay memory $\mathcal{D}$ to capacity $N$.
2. Initialize action-value function $Q$ with random weights.
3. For episode $= 1, M$
   a. Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$.
   b. For $t = 1, T$
      i. With probability $\epsilon$ select a random action $a_t$, otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$.
      ii. Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$.
      iii. Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocessed $\phi_{t+1} = \phi(s_{t+1})$.
      iv. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$.
      v. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$.
      vi. Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$.
      vii. Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3.
   c. End for
   d. End for

[Mnih et al. NIPS Workshop 2013; Nature 2015]
Video by Károly Zsolnai-Fehér.

https://www.youtube.com/watch?v=V1eYniJ0Rnk
Grasping by Q-Learning

Grasping by Q-Learning

Policy Gradients
Policy Gradients – Overall idea

\[ J = E \left[ \Phi(x(N)) + \sum_{k=0}^{N-1} L_k(x(k), u(k)) \right] \]

\[ x(n+1) = f_n(x(n), u) + w(n) \]

\[ \theta^* = \arg \min_{\theta} J(\theta) = \arg \min_{\theta} E \left[ \Phi(x(N)) + \sum_{k=0}^{N-1} L_k(x(k), u(k)) \right] \]

\[
\begin{cases}
  x(n+1) = f_n(x(n), u) + w(n) \\
u(n, x) = \mu(n, x; \theta)
\end{cases}
\]
Policy Gradients - Algorithm

---

**Algorithm 11** Gradient Descent Algorithm

given
    A method to compute \( \nabla_{\theta} J(\theta) \) for all \( \theta \)
    An initial value for the parameter vector: \( \theta \leftarrow \theta_0 \)

repeat
    Compute the cost function gradient at \( \theta \)
    \( g = \nabla_{\theta} J(\theta) \)
    Update the parameter vector
    \( \theta \leftarrow \theta - \omega g \)

until convergence

---

From ‘Optimal and Learning Control’ by Buchli et al.
How do we get the gradients? – REINFORCE Algorithm

Inuition: Gradient Estimator

$$\nabla_{\theta} J(\theta) \approx \sum_{t=0}^{\infty} r(s) \nabla_{\theta} \log P(a|s)$$

- Increase the probability of these actions if reward was high
- Reward over an episode
How to compute the gradient estimates?
Reproducibility in Reinforcement Learning


2. Benjamin Recht: Optimization Perspectives on Learning to Control (ICML 2018 tutorial)
What will you take home today?

Reinforcement Learning
  Recap of Q-Learning
  Deep Q-Learning
  Policy Gradients

Representation Learning
Representation Learning
Representation Learning

Current state $s_t$: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)
Why Representation Learning?

2. Policy evaluation
   repeat
     \( \Delta \leftarrow 0 \)
   for each: \( x \in \mathcal{X} \)
     \( v \leftarrow V(x) \)
     \[
     V(x) \leftarrow \sum_u \pi(x, u) \sum_{x'} \mathcal{P}^u_{xx'} \left[ R^u_{xx'} + \alpha V(x') \right]
     \]
     \( \Delta \leftarrow \max(\Delta, |v - V(x)|) \)
   until \( \Delta < \theta \) (a small positive number)
Case Study: Making Sense of Vision and Touch

Learning a Multimodal Representation from vision and touch