Deep Reinforcement Learning

CS440/ECE448 Lecture 35

Mark Hasegawa-Johnson, 4/2022
CC-BY 4.0: you may remix or redistribute if you cite the source

Image: Megajuice, CC0, https://commons.wikimedia.org/w/index.php?curid=57895741
Review: Reinforcement Learning

• Markov Decision Process (MDP): Given $P(s'|s,a)$ and $R(s)$, you can solve for $\pi^*(s)$, the optimal policy, by finding $U(s)$, the value of each state, using either value iteration or policy iteration.

• Model-Based Reinforcement Learning: If $P(s'|s,a)$ and $R(s)$ are unknown, you can find for $\pi(s)$ by using the observation-model-policy loop.

• Model-Free Reinforcement Learning: Instead of learning $P(s'|s,a)$ and then calculating $\pi(s)$, we can directly find the optimum action by learning $Q(s,a)$. 
Outline

• Imitation learning: learn the optimal policy by imitating a human
• Deep Q learning: compute $Q(s,a)$ using a neural network
Why can’t we just learn a model (neural net, or even a table lookup) that does this:

\[ a = \pi(s) \]
Probabilistic Policy

If we have \(|A|\) possible actions, \(1 \leq a \leq |A|\), we could train the network to learn a hidden layer \(h(s)\) so that:

\[
\pi_a(s) = \frac{\exp(w_a^T h(s))}{\sum_{k=1}^{\mid A \mid} \exp(w_k^T h(s))} = P(A = a|S = s)
\]

Meaning “the probability that the best action is \(a\).”
How do we train it?

- Training data only give us \((s_i, a_i, s'_i, R_i)\).
- BAD IDEA: train the network to choose \(A = a_i\) that maximizes the immediate reward, \(R_i\), and just ignore future rewards.
- GOOD IDEA: Train the network to maximize \(U(s'_i) = \text{sum of all future rewards}\).
- PROBLEM: we don’t know \(U(s'_i)\).
How to make Policy Learning trainable

1. Actor-Critic RL. We’ll come back to this next time.
2. Imitation learning.
Imitation learning

• In some applications, you cannot bootstrap yourself from random policies
  – High-dimensional state and action spaces where most random trajectories fail miserably
  – Expensive to evaluate policies in the physical world, especially in cases of failure

• **Solution:** learn to imitate sample trajectories or demonstrations
  – This is also helpful when there is no natural reward formulation
Imitation learning

- $\tilde{s}_t$ = a representation of the state of the environment at time $t$ (can be a real-valued vector)
- $a_t$ = the action that a human actor performed in response to this state (must be discrete)
- $f_k(\tilde{s}_t) = k^{th}$ element in the softmax output of a neural network, given $\tilde{s}_t$ as the input
- Training criterion: train the neural network in order to minimize

$$\mathcal{L} = -\log f_{a_t}(\tilde{s}_t)$$
Overview of imitation learning methods

Methods differ in:

- Feature representation: raw pixels/joint angles, or have you already used some other method to learn a deep feature representation?
- Training criterion: classification (discrete actions), or regression (continuous actions)?

Hussein et al. *Imitation Learning: A Survey of Learning Methods*, 2018.
Example: Coarse-to-Fine Imitation Learning

Edward Johns, Coarse-to-Fine Imitation Learning: Robot Manipulation from a Single Demonstration, 2021.
Outline

• Imitation learning: learn the optimal policy by imitating a human

• Deep Q learning: compute $Q(s,a)$ using a neural network
Review: Q-Learning

• Q(s,a) – the “quality” of an action
  \[ Q(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a)U(s') \]
  \[ U(s) = \max_{a \in A(s)} Q(s, a) \]

• Q-learning

• Off-policy learning: TD
  \[ Q_{local}(s_t, a_t) = R_t(s_t) + \gamma \max_{a' \in A(s_{t+1})} Q_t(s_{t+1}, a') \]
  \[ Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha (Q_{local}(s_t, a_t) - Q_t(s_t, a_t)) \]

• On-policy learning: SARSA
  \[ a_{t+1} = \pi_t(s_{t+1}) \]
  \[ Q_{local}(s_t, a_t) = R_t(s_t) + \gamma Q_t(s_{t+1}, a_{t+1}) \]
Deep Q learning

Instead of discrete $s$, suppose $\tilde{s}$ is a vector of real numbers, e.g., the image from the robot’s eye camera:

$$\tilde{s} = [s_1, ..., s_D] = \begin{bmatrix} s_1 \\ \vdots \\ s_D \end{bmatrix}$$

Instead of discrete $a$, suppose $\tilde{a}$ is a vector, e.g., cannon angle and velocity,

$$\tilde{a} = [a_1, ..., a_C]$$

Deep Q-learning uses a neural network to compute an estimate $f(\tilde{s}, \tilde{a})$ which is as close as possible to $Q(\tilde{s}, \tilde{a})$. 

Copyright Taito.
Suppose we train the neural network weights in order to minimize the mean-squared error (MMSE):

$$\mathcal{L} = \frac{1}{2} E[(f(\tilde{s}, \tilde{a}) - Q(\tilde{s}, \tilde{a}))^2]$$

(where I’m using $E[\cdot]$ as a lazy way to write “average over all training runs of the game”).

Then, for each weight $w$, we update as

$$w \leftarrow w - \eta \frac{d\mathcal{L}}{dw}$$
What makes deep Q learning harder than normal neural network training

• We don’t know the true value of $Q(\hat{s}, \hat{a})$ for any of the training runs!

• $Q(\hat{s}, \hat{a})$ is defined to be the expected value of performing action $\hat{a}$. We never know its true expected value: all we know is whether we won or lost that particular game.

• So we can’t compute $\mathcal{L}$, and we can’t compute $\frac{d\mathcal{L}}{dw}$, and we can’t update $w$!
Remember that Q learning was defined as

\[ Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left( Q_{local}(s_t, a_t) - Q_t(s_t, a_t) \right) \]

where \( Q_{local}(s_t, a_t) \) is defined, e.g., in TD as

\[ Q_{local}(s_t, a_t) = R_t(s_t) + \gamma \max_{a'} Q_t(s_{t+1}, a') \]

...for \( s_{t+1} \) equal to the next state we reach after action \( a_t \) on this particular game.
The solution: \( Q_{\text{local}} \)

Let’s define deep Q learning using the same \( Q_{\text{local}} \):

\[
\mathcal{L} = \frac{1}{2} E [(f(\hat{s}_t, \hat{a}_t) - Q_{\text{local}}(\hat{s}_t, \hat{a}_t))^2]
\]

where \( Q_{\text{local}}(\hat{s}_t, \hat{a}_t) \) is:

\[
Q_{\text{local}}(\hat{s}_t, \hat{a}_t) = R_t(\hat{s}_t) + \gamma \max_{\hat{a}'} f(\hat{s}_{t+1}, \hat{a}')
\]

Now we have an \( L \) that depends only on things we know \((f(\hat{s}_t, \hat{a}_t), R_t(\hat{s}_t), \text{ and } f(\hat{s}_{t+1}, \hat{a}'))\), so it can be calculated, differentiated, and used to update the neural network.
Dealing with training instability

• Challenges
  – Target values are not fixed
  – Successive experiences are correlated and dependent on the policy
  – Policy may change rapidly with slight changes to parameters, leading to drastic change in data distribution

• Solutions
  – Freeze target Q network
  – Use experience replay
Experience replay

• At each time step:
  – Take action $\tilde{a}_t$ according to epsilon-greedy policy
  – Store experience $(\tilde{s}_t, \tilde{a}_t, r_{t+1}, \tilde{s}_{t+1})$ in replay memory buffer

• Learning:
  – Randomly sample a minibatch, $\mathcal{D}$, from the replay buffer.

\[ \mathcal{D} = \text{randomly sampled set of tuples} \]
Deep Q learning in Atari

Mnih et al. Human-level control through deep reinforcement learning, Nature 2015
Deep Q learning in Atari

- End-to-end learning of $Q(s,a)$ from pixels $s$
- Output is $Q(s,a)$ for 18 joystick/button configurations
- Reward is change in score for that step

Mnih et al. Human-level control through deep reinforcement learning, Nature 2015
Deep Q learning in Atari

• Input state $s$ is stack of raw pixels from last 4 frames
• Network architecture and hyperparameters fixed for all games

Mnih et al. Human-level control through deep reinforcement learning, Nature 2015
Deep Q learning in Atari

Deep Q-Learning Playing Atari Breakout
Summary: Deep RL, Part 1

- Imitation learning: learn the optimal policy by imitating a human
  \[ \mathcal{L} = -\log f_{a_t}(\tilde{s}_t) \]

- Deep Q learning: compute \( Q(s,a) \) using a neural network
  \[ \mathcal{L} = \frac{1}{2} E[(f(\tilde{s}_t, \tilde{a}_t) - Q_{\text{local}}(\tilde{s}_t, \tilde{a}_t))^2] \]
  \[ Q_{\text{local}}(\tilde{s}_t, \tilde{a}_t) = R_t(\tilde{s}_t) + \gamma \max_{\tilde{a}'} f(\tilde{s}_{t+1}, \tilde{a}') \]