CS 4803 / 7643: Deep Learning

Topics:
- Dynamic Programming (Q-Value Iteration)
- Reinforcement Learning (Intro, Q-Learning, DQNs)

Nirbhay Modhe
Georgia Tech
Topics we’ll cover

• Overview of RL
  • RL vs other forms of learning
  • RL “API”
  • Applications
• Framework: Markov Decision Processes (MDP’s)
  • Definitions and notations
  • Policies and Value Functions
  • Solving MDP’s
    • Value Iteration (recap)
    • Q-Value Iteration (new)
    • Policy Iteration
• Reinforcement learning
  • Value-based RL (Q-learning, Deep-Q Learning)
  • Policy-based RL (Policy gradients)
Topics we’ll cover

- Overview of RL
  - RL vs other forms of learning
  - RL “API”
  - Applications
- Framework: Markov Decision Processes (MDP’s)
  - Definitions and notations
  - Policies and Value Functions
  - Solving MDP’s
    - Value Iteration (recap)
    - Q-Value Iteration (new)
    - Policy Iteration
- Reinforcement learning
  - Value-based RL (Q-learning, Deep-Q Learning)
  - Policy-based RL (Policy gradients)
Recap
Recap

- **Markov Decision Process (MDP)**
  - Defined by \((S, A, R, \mathbb{T}, \gamma)\)
    - \(S\) : set of possible states [start state = \(s_0\), optional terminal / absorbing state]
    - \(A\) : set of possible actions
    - \(R(s, a, s')\) : distribution of reward given (state, action, next state) tuple
    - \(\mathbb{T}(s, a, s')\) : transition probability distribution, also written as \(p(s'|s, a)\)
    - \(\gamma\) : discount factor
Recap

- **Markov Decision Process (MDP)**
  - Defined by \((S, A, R, \mathbb{T}, \gamma)\)
    - \(S\) : set of possible states [start state = \(s_0\), optional terminal / absorbing state]
    - \(A\) : set of possible actions
    - \(R(s, a, s')\) : distribution of reward given (state, action, next state) tuple
    - \(\mathbb{T}(s, a, s')\) : transition probability distribution, also written as \(p(s'|s, a)\)
    - \(\gamma\) : discount factor

- Value functions, optimal quantities, bellman equation

- Algorithms for solving MDP’s
  - Value Iteration
Value Function

Following policy $\pi$ that produces sample trajectories $s_0, a_0, r_0, s_1, a_1, \ldots$.
Value Function

Following policy \( \pi \) that produces sample trajectories \( s_0, a_0, r_0, s_1, a_1, \ldots \)

How good is a state?
The value function at state \( s \), is the expected cumulative reward from state \( s \) (and following the policy thereafter):

\[
V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi \right]
\]
Value Function

Following policy $\pi$ that produces sample trajectories $s_0, a_0, r_0, s_1, a_1, \ldots$

How good is a state?
The value function at state $s$, is the expected cumulative reward from state $s$ (and following the policy thereafter):

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi \right]$$

How good is a state-action pair?
The Q-value function at state $s$ and action $a$, is the expected cumulative reward from taking action $a$ in state $s$ (and following the policy thereafter):

$$Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$
Optimal Quantities

Given *optimal* policy $\pi^*$ that produces sample trajectories $s_0, a_0, r_0, s_1, a_1, \ldots$

How good is a state?
The *optimal value function* at state $s$, and acting optimally thereafter

$$V^*(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi^* \right]$$
Optimal Quantities

Given optimal policy $\pi^*$ that produces sample trajectories $s_0, a_0, r_0, s_1, a_1, \ldots$

How good is a state?
The optimal value function at state $s$, and acting optimally thereafter

$$V^*(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi^* \right]$$

How good is a state-action pair?
The optimal Q-value function at state $s$ and action $a$, is the expected cumulative reward from taking action $a$ in state $s$ and acting optimally thereafter

$$Q^*(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi^* \right]$$
Bellman Optimality Equations

• Relations:

\[ V^*(s) = \max_a Q^*(s, a) \]

\[ \pi^*(s) = \arg \max_a Q^*(s, a) \]
Bellman Optimality Equations

• Relations:

\[ V^*(s) = \max_a Q^*(s, a) \]
\[ \pi^*(s) = \arg \max_a Q^*(s, a) \]

• Recursive optimality equations:
Bellman Optimality Equations

- Relations:
  \[
  V^*(s) = \max_a Q^*(s, a) \quad \Rightarrow \quad \pi^*(s) = \arg\max_a Q^*(s, a)
  \]

- Recursive optimality equations:
  \[
  Q^*(s, a) = \mathbb{E}_{s' \sim p(s'|s, a)} \left[ r(s, a) + \gamma V^*(s') \right]
  \]
Bellman Optimality Equations

• Relations:

\[ V^*(s) = \max_{a} Q^*(s, a) \]
\[ \pi^*(s) = \arg \max_{a} Q^*(s, a) \]

• Recursive optimality equations:

\[ Q^*(s, a) = \mathbb{E}_{s' \sim p(s'|s,a)} \left[ r(s, a) + \gamma V^*(s) \right] \]
\[ = \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma V^*(s) \right] \]
Bellman Optimality Equations

• Relations:

\[ V^*(s) = \max_a Q^*(s, a) \quad \pi^*(s) = \arg \max_a Q^*(s, a) \]

• Recursive optimality equations:

\[
Q^*(s, a) = \mathbb{E}_{s' \sim p(s'|s, a)} \left[ r(s, a) + \gamma V^*(s) \right] \\
= \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma V^*(s) \right] \\
= \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma \max_{a'} Q^*(s', a') \right]
\]
Bellman Optimality Equations

• Relations:

\[ V^*(s) = \max_a Q^*(s, a) \quad \pi^*(s) = \arg \max_a Q^*(s, a) \]

• Recursive optimality equations:

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

\[ = \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma V^*(s) \right] \]

\[ = \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma \max_{a'} Q^*(s', a') \right] \]

\[ V^*(s) = \max_a \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma V^*(s') \right] \]
Value Iteration (VI)

- Based on the bellman optimality equation

\[ V^*(s) = \max_a \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma V^*(s') \right] \]
Value Iteration (VI)

- Based on the bellman optimality equation

$$V^*(s) = \max_a \sum_{s'} p(s'|s,a) [r(s,a) + \gamma V^*(s')]$$

- Algorithm
  - Initialize values of all states $V^0(s) = 0$
  - While not converged:
    - For each state:
      $$V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a) [r(s,a) + \gamma V^i(s')]$$
    - Repeat until convergence (no change in values)

Time complexity per iteration $O(|S|^2|A|)$
Q-Value Iteration

• Value Iteration Update:

\[ V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma V^i(s') \right] \]

• Q-Value Iteration Update:

\[ Q^{i+1}(s, a) \leftarrow \]
Q-Value Iteration

• Value Iteration Update:

\[ V_{i+1}^i(s) \leftarrow \max_a \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma V^i(s') \right] \]

• Q-Value Iteration Update:

\[ Q_{i+1}^i(s, a) \leftarrow \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma \max_{a'} Q^i(s', a') \right] \]

The algorithm is same as value iteration, but it loops over actions as well as states
Policy Iteration
Policy Iteration

• Policy iteration: Start with arbitrary $\pi_0$ and refine it.

$$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \ldots \rightarrow \pi^*$$
Policy Iteration

- Policy iteration: Start with arbitrary $\pi_0$ and refine it.

$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \ldots \rightarrow \pi^*$

- Involves repeating two steps:
  - Policy Evaluation: Compute $V^\pi$ (similar to VI)
  - Policy Refinement: Greedily change actions as per $V^\pi$

$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \ldots \rightarrow \pi^* \rightarrow V^{\pi^*}$
Policy Iteration

- Policy iteration: Start with arbitrary $\pi_0$ and refine it.

$$
\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \ldots \rightarrow \pi^*
$$

- Involves repeating two steps:
  - Policy Evaluation: Compute $V^\pi$ (similar to VI)
  - Policy Refinement: Greedily change actions as per $V^\pi$

$$
\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \ldots \rightarrow \pi^* \rightarrow V^{\pi^*}
$$

- Why do policy iteration?
  - $\pi_i$ often converges to $\pi^*$ much sooner than $V^{\pi_i}$
Summary

• Value Iteration
  – Bellman update to state value estimates

• Q-Value Iteration
  – Bellman update to (state, action) value estimates

• Policy Iteration
  – Policy evaluation + refinement
Learning Based Methods
Learning Based Methods

- Typically, we don’t know the environment

  - $\mathbb{T}(s, a, s')$ unknown, how actions affect the environment.

  - $\mathbb{R}(s, a, s')$ unknown, what/when are the good actions?
Learning Based Methods

• Typically, we don’t know the environment
  – $\mathbb{P}(s, a, s')$ unknown, how actions affect the environment.
  – $\mathbb{R}(s, a, s')$ unknown, what/when are the good actions?

• But, we can learn by trial and error.
  – Gather experience (data) by performing actions.
    $$\{s, a, s', r\}_{i=1}^{N}$$
  – Approximate unknown quantities from data.

Reinforcement Learning
Learning Based Methods

• Old Dynamic Programming Demo
  – https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

• RL Demo
  – https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html
(Deep) Learning Based Methods
(Deep) Learning Based Methods

• In addition to not knowing the environment, sometimes the state space is too large.
(Deep) Learning Based Methods

- In addition to not knowing the environment, sometimes the state space is too large.

- A value iteration updates takes $O(|S|^2|A|)$
  - Not scalable to high dimensional states e.g.: RGB images.
(Deep) Learning Based Methods

• In addition to not knowing the environment, sometimes the state space is too large.

• A value iteration update takes \(O(|S|^2|A|)\)
  – Not scalable to high dimensional states e.g.: RGB images.

• Solution: Deep Learning!
  – Use deep neural networks to learn low-dimensional representations.

Deep Reinforcement Learning
Reinforcement Learning
Reinforcement Learning

• Value-based RL
  – (Deep) Q-Learning, approximating $Q^*(s, a)$ with a deep Q-network

(C) Dhruv Batra
Reinforcement Learning

• Value-based RL
  – (Deep) Q-Learning, approximating $Q^*(s, a)$ with a deep Q-network

• Policy-based RL
  – Directly approximate optimal policy $\pi^*$ with a parametrized policy $\pi_{\theta}^*$
Reinforcement Learning

- **Value-based RL**
  - (Deep) Q-Learning, approximating $Q^*(s, a)$ with a deep Q-network

- **Policy-based RL**
  - Directly approximate optimal policy $\pi^*$ with a parametrized policy $\pi^*_\theta$

- **Model-based RL**
  - Approximate transition function $T(s', a, s)$ and reward function $R(s, a)$
  - Plan by looking ahead in the (approx.) future!
Reinforcement Learning

- Value-based RL
  - (Deep) Q-Learning, approximating $Q^* (s, a)$ with a deep Q-network

- Policy-based RL
  - Directly approximate optimal policy $\pi^*$ with a parametrized policy $\pi^*_\theta$

- Model-based RL
  - Approximate transition function $T(s', a, s)$ and reward function $R(s, a)$
  - Plan by looking ahead in the (approx.) future!

Homework!
Value-based Reinforcement Learning

Deep Q-Learning
Deep Q-Learning

- Q-Learning with linear function approximators
  \[ Q(s, a; w, b) = w_a^T s + b_a \]
  - Has some theoretical guarantees
Deep Q-Learning

• Q-Learning with linear function approximators
  \[ Q(s, a; w, b) = w^T_a s + b_a \]
  – Has some theoretical guarantees

• Deep Q-Learning: Fit a deep Q-Network
  \[ Q(s, a; \theta) \]
  – Works well in practice
  – Q-Network can take RGB images

Image Credits: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Deep Q-Learning
Deep Q-Learning

• Assume we have collected a dataset

\[ \{(s, a, s', r)_{i=1}^{N}\} \]

• We want a Q-function that satisfies:

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

• Loss for a single data point:

\[
\text{MSE Loss} := (Q_{\text{new}}(s, a) - (r + \max_a Q_{\text{old}}(s', a)))^2
\]

Predicted Q-Value

Target Q-Value
Deep Q-Learning

- Minibatch of \( \{(s, a, s', r)\}_{i=1}^{B} \)

- Forward pass:

  \[
  \begin{align*}
  \text{State} & \quad \rightarrow \quad \text{Q-Network} \\
  B \times D & \quad \rightarrow \quad B \times n_{\text{actions}}
  \end{align*}
  \]
Deep Q-Learning

- Minibatch of $\{(s, a, s', r)_i\}_{i=1}^B$

- Forward pass:
  \[
  \text{State} \rightarrow \text{Q-Network} \rightarrow \text{Q-Values per action}
  \]

  \[
  B \times D \rightarrow B \times n_{actions}
  \]
Deep Q-Learning

• Minibatch of \( \{(s, a, s', r)_i\}_{i=1}^B \)

• Forward pass:

\[
\begin{align*}
\text{State} & \rightarrow \text{Q-Network} & \rightarrow \text{Q-Values per action} \\
B \times D & & B \times n_{actions}
\end{align*}
\]

• Compute loss:

\[
(Q_{\text{new}}(s, a) - (r + \max_a Q_{\text{old}}(s', a)))^2
\]
Deep Q-Learning

- Minibatch of $\{ (s, a, s', r)_i \}_{i=1}^B$

- Forward pass:
  
  \[
  \begin{align*}
  \text{State} & \quad \rightarrow \quad \text{Q-Network} & \rightarrow \quad \text{Q-Values per action} \\
  B \times D & & B \times n_{\text{actions}}
  \end{align*}
  \]

- Compute loss:
  \[
  \left( Q_{\text{new}}(s, a) - (r + \max_a Q_{\text{old}}(s', a)) \right)^2
  \]
  \[
  \theta_{\text{new}} \quad \theta_{\text{old}}
  \]
Deep Q-Learning

- Minibatch of $\{(s, a, s', r)_i\}_{i=1}^B$

- **Forward pass:**
  - State $B \times D$ to Q-Network $B \times W \times C$ to Q-Values per action $B \times n_{actions}$

- **Compute loss:**
  $$\left( Q_{new}(s, a) - (r + \max_a Q_{old}(s', a)) \right)^2$$

- **Backward pass:**
  $$\frac{\partial Loss}{\partial \theta_{new}}$$
Deep Q-Learning

\[
\text{MSE Loss} := \left( Q_{\text{new}}(s, a) - (r + \max_a Q_{\text{old}}(s', a)) \right)^2
\]

- In practice, for stability:
  - Freeze \( Q_{\text{old}} \) and update \( Q_{\text{new}} \) parameters
  - Set \( Q_{\text{old}} \leftarrow Q_{\text{new}} \) at regular intervals
How to gather experience?

$$\{(s, a, s', r)_i\}_{i=1}^{N}$$

This is why RL is hard
How To Gather Experience?

\[ D = f\left(\pi_{\text{trained}}\right) \]

\[ \pi_{\text{gather}} \rightarrow \text{Environment} \rightarrow \text{Data} \rightarrow \text{Train} \]

\[ \pi_{\text{gather}} \rightarrow \pi_{\text{trained}} \]

\[ \{(s, a, s', r)_i\}_{i=1}^N \]
How To Gather Experience?

Challenge 1: Exploration vs Exploitation

Challenge 2: Non iid, highly correlated data
Exploration Problem

• What should $\pi_{\text{gather}}$ be?
  
  – Greedy? -> Local minima, no exploration

  \[
  \arg \max_a Q(s, a; \theta)
  \]
Exploration Problem

• What should \( \pi_{\text{gather}} \) be?
  
  – Greedy? \( \rightarrow \) Local minima, no exploration
    \[
    \arg \max_a Q(s, a; \theta)
    \]

• An exploration strategy:
  
  – \( \epsilon \)-greedy
    \[
    a_t = \begin{cases} 
    \arg \max_a Q(s, a) & \text{with probability } 1 - \epsilon \\
    \text{random action} & \text{with probability } \epsilon
    \end{cases}
    \]
Correlated Data Problem

- Samples are correlated => high variance gradients => inefficient learning

- Current Q-network parameters determines next training samples => can lead to bad feedback loops
  - e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size.
Experience Replay

- Address this problem using experience replay

  - A replay buffer stores transitions $(s, a, s', r)$
Experience Replay

- Address this problem using experience replay
  - A replay buffer stores transitions \((s, a, s’, r)\)
  - Continually update replay buffer as game (experience) episodes are played, older samples discarded
Experience Replay

• Address this problem using experience replay

  – A replay buffer stores transitions \((s, a, s', r)\)

  – Continually update replay buffer as game (experience) episodes are played, older samples discarded

  – Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples
Q-Learning Algorithm

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

for episode = 1, $M$ do

1. Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
2. for $t = 1, T$ do

1. With probability $\epsilon$ select a random action $a_t$
2. otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
3. Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
4. Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocessed $\phi_{t+1} = \phi(s_{t+1})$
5. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
6. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$

Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

Experience Replay
Epsilon-greedy
Q Update

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Case study: Playing Atari Games

- Objective: Complete the game with the highest score
- State: Raw pixel inputs from the game state
- Action: Game controls e.g.: Left, Right, Up, Down
- Reward: Score increase/decrease at each time step
Playing Atari Games

• **Q-Network architecture**

• **State:**
  - Stack of 4 image frames, grayscale conversion, down-sampling and cropping to (84 x 84 x 4)

• Last FC layer has #(actions) dimensions (predicts Q-values)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Atari Games

Breakout

Pong

https://www.youtube.com/watch?v=V1eYniJ0RnK

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
In today’s class, we looked at

• Dynamic Programming
  – Q-Value Iteration
  – Policy Iteration

• Reinforcement Learning (RL)
  – The challenges of (deep) learning based methods
  – Value-based RL algorithms
    • Deep Q-Learning

Next class:
  – Policy-based RL algorithms
Thanks!