Reinforcement Learning

Instructor: Pranjal Awasthi
So Far

• Passive Learning
• Semi supervised Learning  
  – These are examples of learning from feedback

• Today  
  – Reinforcement learning  
  – Learning from “delayed” feedback
Reinforcement Learning

• Most common applications
  – Robotics
  – Game playing
Motivating Example

• Goal: Teach a robot to fly a helicopter
• Passive Learning approach: provide training data
  • \((X_i, y_i)\)
  • \(X_i = \) various sensor values
  • \(y_i = \) next direction to take
Motivating Example

- Goal: Win the game
- Passive Learning approach: provide training data
  - \((X_i, y_i)\)
  - \(X_i = \text{state of the game}\)
  - \(y_i = \text{next action to take}\)
Reinforcement Learning (RL)

• Most common applications
  – Robotics
  – Game playing

• In many scenarios, passive learning not the right approach
  – Hard to collect/define training data
  – RL idea:
    • Design systems that imitate human learning
    • Take an action, get feedback, update strategy, repeat
Reinforcement Learning

• Basic Setup
  – There’s an agent $A_{agent}$
  – Set of states $S = \{s_1, s_2, \ldots, s_{|S|}\}$
  – Set of Actions $A = \{a_1, a_2, \ldots, a_{|A|}\}$
  – Agent interacts with the environment as follows
    • Starts at some state
    • Takes an action and moves to a different state
    • Gets a reward for taking the action
  – Goal: Maximize long term reward
Motivating Example

\[ S_1 \xrightarrow{a_1} S_2 \]
\[ S_1 \xrightarrow{a_1} S_2 \]

after taking \( a_1 \) from \( S_1 \)

\[ S_2 \text{ is non-deterministic} \]

if \( S_2 \text{ does not crash} \)

\[ \text{round} = 1 \]

else \[ \text{round} = 0 \]
Motivating Example
Reinforcement Learning

• Basic Setup
  – There’s an agent \( A \)
  – Set of states \( S = \{ s_1, s_2, \ldots, s_{|S|} \} \)
  – Set of Actions \( A = \{ a_1, a_2, \ldots, a_{|A|} \} \)
  – Reward function
    • \( R(s', a, s) \) = reward for taking action \( a \) at \( s \) and going to \( s' \)
      \[ R(s', a, s) \in [0, 1] \]
  – Transition probability
    • \( P(s' | s, a) = \text{Pr}[\text{reaching } s' \text{ from } s \text{ by taking action } a] \)
Reinforcement Learning

• Basic Setup
  – There’s an agent $A$
  – Set of states $S = \{s_1, s_2, ... s_{|S|}\}$
  – Set of Actions $A = \{a_1, a_2, ... a_{|A|}\}$
  – $R(s, a, s') = \text{reward for taking action } a \text{ at } s \text{ and going to } s'$
  – $P(s' | s,a) = \text{Pr[reaching } s' \text{ from } s \text{ by taking action } a]$}

• Goal: Learn a good “policy” $\pi$
  – $\pi: S \rightarrow A$
Reinforcement Learning

current state $s_1$

policy $\pi$

transition $\pi(s_1)$, $R(s_2, \pi(s_1), s_1)$, $s_2$

transition $\pi(s_2)$, $R(s_3, \pi(s_2), s_2)$, $s_3$

...
Reinforcement Learning

• Fundamental question:
  – If agent is at state $s$, what is the optimal action to take?
  – Take action that gets maximum long term reward

• $V(s) = \max_a \sum_{s'} P(s'|s, a)[R(s', s, a) + V(s')]$, $\forall s \in S$
Reinforcement Learning

• Fundamental question:
  – If agent is at state $s$, what is the optimal action to take?
  – Take action that gets maximum long term reward

• $V(s) = \max_a \sum_{s'} P(s'|s,a)[R(s', s, a) + \gamma V(s')]$

• $\gamma =$ discount factor

$0 < \gamma < 1$

Future rewards are discounted.
Reinforcement Learning

• Fundamental question:
  – If agent is at state $s$, what is the optimal action to take?
  – Take action that gets maximum long term reward

  $$V(s) = \max_a \sum_{s'} P(s'|s, a)[R(s', s, a) + \gamma V(s')]$$

• Two Algorithms to learn optimal policy
  – Value Iteration
  – Policy Iteration
Value Iteration

- Idea: Solve Bellman’s equations iteratively

Initialize $V^0(s)$ arbitrarily, $\forall s$

For $t = 1, 2, 3, \ldots$

$\forall s$, $V_t(s) = \max_a \left\{ \sum_{s'} P(s'|s,a) [R(s',s) + \gamma V_{t-1}(s')] \right\}$

Runtime: $O(|S||A|)$ per iteration.
Theorem:
Value iteration converges to the optimal $V^*(s)$ satisfying

$$\forall s \in S, V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s',s,a) + \gamma V^*(s')]$$

Convergence rate:
$$\|V_{t+1} - V^*\|_\infty \leq \gamma \|V_t - V^*\|_\infty$$
$$\max_s |V_{t+1}(s) - V^*(s)| \leq \gamma \max_s |V_t(s) - V^*(s)|$$
Policy Iteration

• Idea: Search over the space of policies
  – Compute the value of a given policy
  – Greedily move to a new policy

\[
\begin{align*}
\text{Start from an arbitrary policy } \pi^0 \\
\text{For } t = 1, 2, \ldots \\
\quad \text{Compute } V^{\pi_t}(S) \quad \forall S \\
\quad \text{Greedily update to a new policy } \pi_t \\
\end{align*}
\]
Policy Iteration

Theorem:
Policy iteration converges to a policy $\pi$ with $V^*_\pi$ satisfying

$$\forall s \in S, V^*_\pi(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s, a)[R(s', s, a) + \gamma V^*_\pi(s')]$$

In a lot of cases,

$$\left\| V^t - V^* \right\|_\infty \leq \gamma \left\| V^t - V^* \right\|_\infty^2$$

$$V^t(s) = \sum_{s'} P(s'|s, \pi(s))\left[ R(s', s, \pi(s)) + \gamma V^t(s') \right]$$
Reinforcement Learning

• Fundamental question:
  – If agent is at state $s$, what is the optimal action to take?

• Two Algorithms to learn optimal policy
  – Value Iteration
  – Policy Iteration

• Two questions
  – Where do $P(s'|s, a)$ and $R(s', s, a)$ come from?
  – What if $|S|$ is huge?
Learning model parameters

• Goal: learn transition prob’s and reward functions
• Approach 1
  – Explore the state space and learn the values using MLE
Learning model parameters

• Goal: learn transition prob’s and reward functions
• Approach 1
  – Explore the state space and learn the values using MLE
  – Good for small state spaces
  – Some states might be hard to reach
  – Leads to strongest theoretical results
Learning model parameters

• Goal: learn transition prob’s and reward functions
• Approach 2
  – Learn optimal policy directly
  – Exploration/exploitation tradeoff
  – Q-learning: a popular algorithm
Q-Learning

- For each state and action maintain $Q(s, a)$
Q-Learning

• For each state and action maintain $Q(s, a)$

• Initialize $Q_0(s, a)$ arbitrarily, choose initial state $s_0$

• For $t = 1, 2, ...$
  – From $s_{t-1}$ choose an action $a$, go to state $s'$ and observe reward $r(s', s, a)$
  – Update: $Q_t(s, a) = r(s', s, a) + \gamma \max_{a'} Q_{t-1}(s', a')$
What if $|S|$ is huge?

- Approximate value $V()$ with a parametric form
  - Represent state $s$ with a feature vector $\phi(s)$
  - Assume: $V^*(s) = f(\theta, \phi(s))$
    - Ex: $f(\theta, \phi(s)) = \theta \cdot \phi(s)$
  - New goal: learn the parameter $\theta$

\[
\begin{align*}
\left(\phi(s_1), V(s_1)\right), \left(\phi(s_2), V(s_2)\right), \ldots, \left(\phi(s_m), V(s_m)\right)
\end{align*}
\]

- Start with $\Theta^0$
- Generate training set using $\Theta^0$
- Update to $\Theta_1$
Value Iteration with function approximation

- Choose $S' \subset S$
- Guess a value $\theta_0$ for parameters
- For $t = 1, 2, ...$
  - Step 1: Bellman update
    $$V_t(s) = \max_a \sum_{s' \in S} P(s'|s, a) [R(s', s, a) + \gamma V_{\theta_{t-1}}(s')]$$
  - Step 2: supervised learning
    $$\theta_t = \arg\min_{\theta} \sum_{s \in S'} \left( f(\theta, \phi(s)) - V_t(s) \right)^2$$

most useful way to do $\Delta$
Reinforcement Learning

• Basic Setup
  – There’s an agent $A$
  – Set of states $S = \{s_1, s_2, \ldots, s_{|S|}\}$
  – Set of Actions $A = \{a_1, a_2, \ldots, a_{|A|}\}$
  – Agent interacts with the environment as follows
    • Starts at some state
    • Takes an action and moves to a different state
    • Gets a reward for taking the action
  – Goal: Maximize long term reward