Reinforcement Learning (1): Discrete MDP, Value Iteration, Policy Iteration

Piyush Rai

CS5350/6350: Machine Learning

November 29, 2011

Reinforcement Learning

- Supervised Learning: Uses explicit supervision (input-output pairs)
- **Reinforcement Learning:** No explicit supervision

Reinforcement Learning

- Supervised Learning: Uses explicit supervision (input-output pairs)
- **Reinforcement Learning:** No explicit supervision
- Learning is modeled as interactions of an agent with an environment
  - Based on using a *feedback mechanism* (in form of a reward function)

Reinforcement Learning

- Supervised Learning: Uses explicit supervision (input-output pairs)
- **Reinforcement Learning:** No explicit supervision
- Learning is modeled as interactions of an agent with an environment
  - Based on using a *feedback mechanism* (in form of a reward function)
- Applications:
  - Robotics (autonomous driving, robot locomotion, etc.)
  - (Computer) Game Playing
  - Online Advertising
  - Information Retrieval (interactive search)
  - .. and many more

Markov Decision Processes (MDP)

Used for modeling the environment the agent is acting in

Markov Decision Processes (MDP)

Used for modeling the environment the agent is acting in

Defined by a tuple $(S, A, \{P_{sa}\}, \gamma, R)$

Markov Decision Processes (MDP)

Used for modeling the environment the agent is acting in

Defined by a tuple $(S, A, \{P_{sa}\}, \gamma, R)$

- S is a set of states (today's class: finite state space)

Markov Decision Processes (MDP)

Used for modeling the environment the agent is acting in

Defined by a tuple $(S, A, \{P_{sa}\}, \gamma, R)$

- S is a set of states (today's class: finite state space)
- A is a set of actions

Markov Decision Processes (MDP)

Used for modeling the environment the agent is acting in

Defined by a tuple $(S, A, \{P_{sa}\}, \gamma, R)$

- S is a set of states (today's class: finite state space)
- A is a set of actions
- P_{sa} is a probability distribution over the state space

Markov Decision Processes (MDP)

Used for modeling the environment the agent is acting in

Defined by a tuple $(S, A, \{P_{sa}\}, \gamma, R)$

- S is a set of states (today's class: finite state space)
- A is a set of actions
- P_{sa} is a probability distribution over the state space
  - i.e., probability of switching to some state s' if we took action a in state s

Markov Decision Processes (MDP)

Used for modeling the environment the agent is acting in

Defined by a tuple $(S, A, \{P_{sa}\}, \gamma, R)$

- S is a set of states (today's class: finite state space)
- A is a set of actions
- P_{sa} is a probability distribution over the state space
  - i.e., probability of switching to some state s' if we took action a in state s
  - For finite state spaces, P_{sa} is a vector of size $|S|$ (and sums to 1)

Markov Decision Processes (MDP)

Used for modeling the environment the agent is acting in

Defined by a tuple $(S, A, \{P_{sa}\}, \gamma, R)$

- S is a set of states (today's class: finite state space)
- A is a set of actions
- P_{sa} is a probability distribution over the state space
  - i.e., probability of switching to some state s' if we took action a in state s
  - For finite state spaces, P_{sa} is a vector of size $|S|$ (and sums to 1)
- $R : S \times A \mapsto \mathbb{R}$ is the reward function (function of state-action pairs)
  - Note: Often the reward is a function of the state only $R : S \mapsto \mathbb{R}$

Markov Decision Processes (MDP)

Used for modeling the environment the agent is acting in

Defined by a tuple $(S, A, \{P_{sa}\}, \gamma, R)$

- S is a set of states (today's class: finite state space)
- A is a set of actions
- P_{sa} is a probability distribution over the state space
  - i.e., probability of switching to some state s' if we took action a in state s
  - For finite state spaces, P_{sa} is a vector of size $|S|$ (and sums to 1)
- $R : S \times A \mapsto \mathbb{R}$ is the reward function (function of state-action pairs)
  - Note: Often the reward is a function of the state only $R : S \mapsto \mathbb{R}$
- $\gamma \in [0, 1)$ is called discount factor for future rewards

MDP Dynamics

- Start in some state $s_0 \in S$

MDP Dynamics

- Start in some state $s_0 \in S$
- Choose action $a_0 \in A$ in state s_0

MDP Dynamics

- Start in some state $s_0 \in S$
- Choose action $a_0 \in A$ in state s_0
- New MDP state $s_1 \in S$ chosen according to $P_{s_0 a_0}$: $s_1 \sim P_{s_0 a_0}$

MDP Dynamics

- Start in some state $s_0 \in S$
- Choose action $a_0 \in A$ in state s_0
- New MDP state $s_1 \in S$ chosen according to $P_{s_0 a_0}$: $s_1 \sim P_{s_0 a_0}$
- Choose action $a_1 \in A$ in state s_1

MDP Dynamics

- Start in some state $s_0 \in S$
- Choose action $a_0 \in A$ in state s_0
- New MDP state $s_1 \in S$ chosen according to $P_{s_0 a_0}$: $s_1 \sim P_{s_0 a_0}$
- Choose action $a_1 \in A$ in state s_1
- New MDP state $s_2 \in S$ chosen according to $P_{s_1 a_1}$: $s_2 \sim P_{s_1 a_1}$

MDP Dynamics

- Start in some state $s_0 \in S$
- Choose action $a_0 \in A$ in state s_0
- New MDP state $s_1 \in S$ chosen according to $P_{s_0 a_0}$: $s_1 \sim P_{s_0 a_0}$
- Choose action $a_1 \in A$ in state s_1
- New MDP state $s_2 \in S$ chosen according to $P_{s_1 a_1}$: $s_2 \sim P_{s_1 a_1}$
- Choose action $a_2 \in A$ in state s_2, and so on..

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots$$

Payoff and Expected Payoff

- Payoff defines the cumulative reward
- Upon visiting states $s_0, s_1, \ldots$ with actions $a_0, a_1, \ldots$, the payoff:

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \ldots$$

Payoff and Expected Payoff

- Payoff defines the cumulative reward
- Upon visiting states $s_0, s_1, \ldots$ with actions $a_0, a_1, \ldots$, the payoff:

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \ldots$$

- Reward at time t is discounted by γ^t (note: $\gamma < 1$)
  - We care more about immediate rewards, rather than the future rewards

Payoff and Expected Payoff

- Payoff defines the cumulative reward
- Upon visiting states $s_0, s_1, \ldots$ with actions $a_0, a_1, \ldots$, the payoff:

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \ldots$$

- Reward at time t is discounted by γ^t (note: $\gamma < 1$)
  - We care more about immediate rewards, rather than the future rewards
- If rewards defined in terms of states only, then the payoff:

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

Payoff and Expected Payoff

- Payoff defines the cumulative reward
- Upon visiting states $s_0, s_1, \ldots$ with actions $a_0, a_1, \ldots$, the payoff:

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \ldots$$

- Reward at time t is discounted by γ^t (note: $\gamma < 1$)
  - We care more about immediate rewards, rather than the future rewards
- If rewards defined in terms of states only, then the payoff:

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

- We want to choose actions over time to maximize the expected payoff:

$$\mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots]$$

- Expectation is w.r.t. all possibilities for the initial state

Policy Function

- **Policy** is a function $\pi : S \mapsto A$, mapping from the states to the actions
- For an agent with policy π, the action in state s: $a = \pi(s)$

Policy Function

- **Policy** is a function $\pi : S \mapsto A$, mapping from the states to the actions
- For an agent with policy π, the action in state s: $a = \pi(s)$
- **Value Function** for a policy π

$$V^{\pi}(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots | s_0 = s, \pi]$$

- $V^{\pi}(s)$ is the expected payoff starting in state s **and** following policy π

Policy Function

- **Policy** is a function $\pi : S \mapsto A$, mapping from the states to the actions
- For an agent with policy π, the action in state s: $a = \pi(s)$
- **Value Function** for a policy π

$$V^\pi(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots | s_0 = s, \pi]$$

- $V^\pi(s)$ is the expected payoff starting in state s **and** following policy π
- **Bellman's Equation:** Gives a recursive definition of the Value Function:

$$\begin{aligned} V^\pi(s) &= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s') \\ &= R(s) + \mathbb{E}_{s' \sim P_{s\pi(s)}}[V^\pi(s')] \end{aligned}$$

- It's the immediate reward + expected sum of future discounted rewards

Computing the Value Function

- Bellman's equation can be used to compute the value function $V^\pi(s)$

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s')$$

- For an MDP with finite many state, it gives us $|S|$ equations with $|S|$ unknowns $\Rightarrow$ Efficiently solvable

Computing the Value Function

- Bellman's equation can be used to compute the value function $V^\pi(s)$

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s')$$

- For an MDP with finite many state, it gives us $|S|$ equations with $|S|$ unknowns $\Rightarrow$ Efficiently solvable
- **Optimal Value Function** is defined as:

$$V^*(s) = \max_\pi V^\pi(s)$$

- It's the best possible payoff that any policy π can give

Computing the Value Function

- Bellman's equation can be used to compute the value function $V^\pi(s)$

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s')V^\pi(s')$$

- For an MDP with finite many state, it gives us $|S|$ equations with $|S|$ unknowns $\Rightarrow$ Efficiently solvable
- **Optimal Value Function** is defined as:

$$V^*(s) = \max_\pi V^\pi(s)$$

- It's the best possible payoff that any policy π can give
- The Optimal Value Function can also be defined as:

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s')V^*(s')$$

Optimal Policy

- The **Optimal Value Function**:

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

Optimal Policy

- The **Optimal Value Function**:

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

- **Optimal Policy** $\pi^* : S \mapsto A$:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

- The optimal policy for state s gives the action a that maximizes the optimal value function for that state

Optimal Policy

- The **Optimal Value Function**:

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s')V^*(s')$$

- **Optimal Policy** $\pi^* : S \mapsto A$:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s')V^*(s')$$

- The optimal policy for state s gives the action a that maximizes the optimal value function for that state
- For every state s and every policy π

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$$

- **Note:** π^* is the optimal policy function for all states s
  - Doesn't matter what the initial MDP state is

Finding the Optimal Policy

- Optimal Policy $\pi^* : S \mapsto A$:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s') \tag{1}$$

- **Two standard methods** to find it

Finding the Optimal Policy

- Optimal Policy $\pi^* : S \mapsto A$:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s') \tag{1}$$

- **Two standard methods** to find it
  - **Value Iteration:** Zero-initialize and iteratively refine $V(s)$ as it will converge towards $V^*(s)$. Finally use equation 1 to find the optimal policy π^*

Finding the Optimal Policy

- Optimal Policy $\pi^* : S \mapsto A$:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s') \quad (1)$$

- **Two standard methods** to find it
  - **Value Iteration:** Zero-initialize and iteratively refine $V(s)$ as it will converge towards $V^*(s)$. Finally use equation 1 to find the optimal policy π^*
  - **Policy Iteration:** Random-initialize and iteratively refine $\pi(s)$ by alternating between computing $V(s)$ and then $\pi(s)$ as per equation 1. π eventually converges to the optimal policy π^*

Finding the Optimal Policy: Value Iteration

Iteratively compute/refine the value function V until convergence

1. For each state s, initialize $V(s) := 0$.
2. Repeat until convergence {

   For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s')V(s')$.

   }

Finding the Optimal Policy: Value Iteration

Iteratively compute/refine the value function V until convergence

1. For each state s, initialize $V(s) := 0$.
2. Repeat until convergence {

   For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s')V(s')$.

   }

- Value Iteration property: V converges to V^*

Finding the Optimal Policy: Value Iteration

Iteratively compute/refine the value function V until convergence

1. For each state s, initialize $V(s) := 0$.
2. Repeat until convergence {

   For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$.

   }

- Value Iteration property: V converges to V^*
- Upon convergence, use $\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$

Finding the Optimal Policy: Value Iteration

Iteratively compute/refine the value function V until convergence

1. For each state s, initialize $V(s) := 0$.
2. Repeat until convergence {

   For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s')V(s')$.

   }

- Value Iteration property: V converges to V^*
- Upon convergence, use $\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s')V^*(s')$
- **Note:** The inner loop can update $V(s)$ for all states simultaneously, or in some order

Finding the Optimal Policy: Policy Iteration

Iteratively compute/refine the policy π until convergence

1. Initialize π randomly.
2. Repeat until convergence {
   (a) Let $V := V^{\pi}$.
   (b) For each state s, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s')V(s')$.

   }

Finding the Optimal Policy: Policy Iteration

Iteratively compute/refine the policy π until convergence

1. Initialize π randomly.
2. Repeat until convergence {
   (a) Let $V := V^{\pi}$.
   (b) For each state s, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s')V(s')$.

   }

- Step (a) the computes the value function for the current policy π
  - Can be done using Bellman's equations (solving $|S|$ equations in $|S|$ unknowns)

Finding the Optimal Policy: Policy Iteration

Iteratively compute/refine the policy π until convergence

1. Initialize π randomly.
2. Repeat until convergence {
   (a) Let $V := V^{\pi}$.
   (b) For each state s, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s')V(s')$.

   }

- Step (a) the computes the value function for the current policy π
  - Can be done using Bellman's equations (solving $|S|$ equations in $|S|$ unknowns)
- Step (b) gives the policy that is greedy w.r.t. V

Learning an MDP Model

- So far we assumed:
  - State transition probabilities $\{P_{sa}\}$ are given
  - Rewards $R(s)$ at each state are known
- Often we don't know these and want to learn these

Learning an MDP Model

- So far we assumed:
  - State transition probabilities $\{P_{sa}\}$ are given
  - Rewards $R(s)$ at each state are known
- Often we don't know these and want to learn these
- These are learned using experience (i.e., a set of previous trials)

$$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} s_3^{(1)} \xrightarrow{a_3^{(1)}} \dots$$
$$s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} s_3^{(2)} \xrightarrow{a_3^{(2)}} \dots$$
$$\dots$$

- $s_i^{(j)}$ is the state at time i of trial j
- $a_i^{(j)}$ is the corresponding action at that state

Learning an MDP Model

- Given this experience, the MLE estimate of state transition probabilities:

$$P_{sa}(s') = \frac{\#\text{ of times we took action } a \text{ in state } s \text{ and got to } s'}{\#\text{ of times we took action } a \text{ in state } s}$$

Learning an MDP Model

- Given this experience, the MLE estimate of state transition probabilities:

$$P_{sa}(s') = \frac{\#\text{ of times we took action } a \text{ in state } s \text{ and got to } s'}{\#\text{ of times we took action } a \text{ in state } s}$$

- Note: if action a is never taken in state s, the above ratio is $0/0$
    - In that case: $P_{sa}(s') = 1/|S|$ (uniform distribution over all states)

Learning an MDP Model

- Given this experience, the MLE estimate of state transition probabilities:

$$P_{sa}(s') = \frac{\#\text{ of times we took action } a \text{ in state } s \text{ and got to } s'}{\#\text{ of times we took action } a \text{ in state } s}$$

- Note: if action a is never taken in state s, the above ratio is $0/0$
  - In that case: $P_{sa}(s') = 1/|S|$ (uniform distribution over all states)
- P_{sa} is easy to update if we gather more experience (i.e., do more trials)
  - .. just add counts in the numerator and denominator

Learning an MDP Model

- Given this experience, the MLE estimate of state transition probabilities:

$$P_{sa}(s') = \frac{\#\text{ of times we took action } a \text{ in state } s \text{ and got to } s'}{\#\text{ of times we took action } a \text{ in state } s}$$

- Note: if action a is never taken in state s, the above ratio is $0/0$
  - In that case: $P_{sa}(s') = 1/|S|$ (uniform distribution over all states)
- P_{sa} is easy to update if we gather more experience (i.e., do more trials)
  - .. just add counts in the numerator and denominator
- Likewise, the expected reward $R(s)$ in state s can be computed
  - $R(s)$ = average reward in state s across all the trials

MDP Learning + Policy Learning

Alternate between learning the MDP (P_{sa} and R), and learning the policy

Policy learning step can be done using value iteration or policy iteration

MDP Learning + Policy Learning

Alternate between learning the MDP (P_{sa} and R), and learning the policy

Policy learning step can be done using value iteration or policy iteration

The Algorithm (uses value iteration)

- Randomly initialize policy π

MDP Learning + Policy Learning

Alternate between learning the MDP (P_{sa} and R), and learning the policy

Policy learning step can be done using value iteration or policy iteration

The Algorithm (uses value iteration)

- Randomly initialize policy π
- Repeat until convergence
  1. Execute policy π in the MDP to generate a set of trials

MDP Learning + Policy Learning

Alternate between learning the MDP (P_{sa} and R), and learning the policy

Policy learning step can be done using value iteration or policy iteration

The Algorithm (uses value iteration)

- Randomly initialize policy π
- Repeat until convergence
  1. Execute policy π in the MDP to generate a set of trials
  2. Use this "experience" to estimate P_{sa} and R

MDP Learning + Policy Learning

Alternate between learning the MDP (P_{sa} and R), and learning the policy

Policy learning step can be done using value iteration or policy iteration

The Algorithm (uses value iteration)

- Randomly initialize policy π
- Repeat until convergence
  1. Execute policy π in the MDP to generate a set of trials
  2. Use this "experience" to estimate P_{sa} and R
  3. Apply value iteration with the estimated P_{sa} and R

     $\Rightarrow$ Gives a new estimate of the value function V

MDP Learning + Policy Learning

Alternate between learning the MDP (P_{sa} and R), and learning the policy

Policy learning step can be done using value iteration or policy iteration

The Algorithm (uses value iteration)

- Randomly initialize policy π
- Repeat until convergence
  1. Execute policy π in the MDP to generate a set of trials
  2. Use this "experience" to estimate P_{sa} and R
  3. Apply value iteration with the estimated P_{sa} and R

     $\Rightarrow$ Gives a new estimate of the value function V
  4. Update policy π as the greedy policy w.r.t. V

MDP Learning + Policy Learning

Alternate between learning the MDP (P_{sa} and R), and learning the policy

Policy learning step can be done using value iteration or policy iteration

The Algorithm (uses value iteration)

- Randomly initialize policy π
- Repeat until convergence
  1. Execute policy π in the MDP to generate a set of trials
  2. Use this "experience" to estimate P_{sa} and R
  3. Apply value iteration with the estimated P_{sa} and R

     $\Rightarrow$ Gives a new estimate of the value function V
  4. Update policy π as the greedy policy w.r.t. V

Note: Step 3 can be made more efficient by initializing V with values from the previous iteration

Value Iteration vs Policy Iteration

- Small state spaces: Policy Iteration typically very fast and converges quickly

Value Iteration vs Policy Iteration

- Small state spaces: Policy Iteration typically very fast and converges quickly
- Large state spaces: Policy Iteration may be slow
  - Reason: Policy Iteration needs to solve a large system of linear equations
  - Value iteration is preferred in such cases

Value Iteration vs Policy Iteration

- Small state spaces: Policy Iteration typically very fast and converges quickly
- Large state spaces: Policy Iteration may be slow
  - Reason: Policy Iteration needs to solve a large system of linear equations
  - Value iteration is preferred in such cases
- Very large state spaces: Value function can be *approximated* using some regression algorithm
  - Optimality guarantee is lost however

Next Class

- Continuous state MDP
  - State-space discretization
  - Value function approximation