CS 4803 / 7643
Deep Learning, Fall 2020

Reinforcement Learning: Module 1/3

Presented by Nirbhay Modhe
Reinforcement Learning
Introduction
Types of Machine Learning

**Supervised Learning**
- Train Input: \(\{X, Y\}\)
- Learning output: \(f : X \rightarrow Y, P(y|x)\)
- e.g. classification

**Unsupervised Learning**
- Input: \(\{X\}\)
- Learning output: \(P(x)\)
- Example: Clustering, density estimation, etc.

**Reinforcement Learning**
- Evaluative feedback in the form of **reward**
- No supervision on the right action
RL: Sequential decision making in an environment with evaluative feedback.

**What is Reinforcement Learning?**

- **Environment** may be unknown, non-linear, stochastic and complex.
- **Agent** learns a **policy** to map states of the environments to actions.
  - Seeking to maximize cumulative reward in the long run.

---

Figure Credit: Rich Sutton
What is Reinforcement Learning?

**RL:** Sequential decision making in an environment with evaluative feedback.

### Evaluative Feedback
- Pick an action, receive a reward (positive or negative)
- No supervision for what the “correct” action is or would have been, unlike supervised learning

### Sequential Decisions
- Plan and execute actions over a sequence of states
- Reward may be delayed, requiring optimization of future rewards (long-term planning).
At each time step $t$, the agent:
- Receives observation $o_t$
- Executes action $a_t$

At each time step $t$, the environment:
- Receives action $a_t$
- Emits observation $o_{t+1}$
- Emits scalar reward $r_{t+1}$

Slide credit: David Silver
Signature Challenges in Reinforcement Learning

- **Evaluative feedback**: Need trial and error to find the right action
- **Delayed feedback**: Actions may not lead to immediate reward
- **Non-stationarity**: Data distribution of visited states changes when the policy changes
- **Fleeting nature of time and online data**

Slide adapted from: Richard Sutton
Robot Locomotion

- **Objective**: Make the robot move forward
- **State**: Angle and position of the joints
- **Action**: Torques applied on joints
- **Reward**: +1 at each time step upright and moving forward

Figures copyright John Schulman et al., 2016. Reproduced with permission.

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

- **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

Figures copyright Volodymyr Mnih et al., 2013. Reproduced with permission.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Examples of RL tasks

**Go**

- **Objective**: Defeat opponent
- **State**: Board pieces
- **Action**: Where to put next piece down
- **Reward**: -1 if win at the end of game, 0 otherwise

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Markov Decision Processes
**MDPs**: Theoretical framework underlying RL

- An MDP is defined as a tuple:
  - Set of possible states
  - Set of possible actions
  - Distribution of reward
  - Transition probability distribution, also written as $p(s'|s,a)$
  - Discount factor

- **Interaction trajectory**

- **Markov property**: Current state completely characterizes state of the environment

- Assumption: Most recent observation is a sufficient statistic of history $(S, A, R, T)$
**MDPs**: Theoretical framework underlying RL

An MDP is defined as a tuple \((S, \mathcal{A}, R, T, \gamma)\)

- \(S\): Set of possible states
- \(\mathcal{A}\): Set of possible actions
- \(R(s, a, s')\): Distribution of reward
- \(T(s, a, s')\): Transition probability distribution, also written as \(p(s'|s,a)\)
- \(\gamma\): Discount factor

**Markov property**: Current state completely characterizes state of the environment

**Assumption**: Most recent observation is a sufficient statistic of history \((S, A, R, T, \gamma)\)
MDPs: Theoretical framework underlying RL

An MDP is defined as a tuple $(S, A, R, T, \gamma)$

$S$: Set of possible states

$A$: Set of possible actions

$R(s, a, s')$: Distribution of reward

$T(s, a, s')$: Transition probability distribution, also written as $p(s'|s, a)$

$\gamma$: Discount factor

Interaction trajectory: $\ldots S_t, a_t, r_{t+1}, S_{t+1}, a_{t+1}, r_{t+2}, S_{t+2}, \ldots$
MDPs: Theoretical framework underlying RL

An MDP is defined as a tuple \((S, A, R, T, \gamma)\)

- \(S\): Set of possible states
- \(A\): Set of possible actions
- \(R(s, a, s')\): Distribution of reward
- \(T(s, a, s')\): Transition probability distribution, also written as \(p(s'|s, a)\)
- \(\gamma\): Discount factor

Interaction trajectory: \(\ldots S_t, a_t, r_{t+1}, S_{t+1}, a_{t+1}, r_{t+2}, S_{t+2}, \ldots\)

Markov property: Current state completely characterizes state of the environment

Assumption: Most recent observation is a sufficient statistic of history

\[
p(S_{t+1} = s'|S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, \ldots S_0 = s_0) = p(S_{t+1} = s'|S_t = s_t, A_t = a_t)
\]
**Fully observed MDP**
- Agent receives the true state $s_t$ at time $t$
- Example: Chess, Go

**Partially observed MDP**
- Agent perceives its own partial observation $o_t$ of the state $s_t$ at time $t$, using past states e.g. with an RNN
- Example: Poker, First-person games (e.g. Doom)

Source: https://github.com/mwydmuch/ViZDoom
Fully observed MDP:
- Agent receives the true state $s_t$ at time $t$
- Example: Chess, Go

Partially observed MDP:
- Agent perceives its own partial observation $o_t$ of the state $s_t$ at time $t$, using past

We will assume **fully observed MDPs** for this lecture.

Source: [https://github.com/mwydmuch/ViZDoom](https://github.com/mwydmuch/ViZDoom)
In Reinforcement Learning, we assume an underlying MDP with unknown:
- Transition probability distribution $T$
- Reward distribution $R$

MDP
$(S, A, R, T, \gamma)$
In Reinforcement Learning, we assume an underlying MDP with unknown:

- Transition probability distribution $T$
- Reward distribution $R$

Evaluative feedback comes into play, trial and error necessary
In Reinforcement Learning, we assume an underlying MDP with unknown:
- Transition probability distribution \( T \)
- Reward distribution \( R \)

Evaluative feedback comes into play, trial and error necessary

For this lecture, assume that we know the true reward and transition distribution and look at algorithms for solving MDPs i.e. finding the best policy
- Rewards known everywhere, no evaluative feedback
- Know how the world works i.e. all transitions

MDPs in the context of RL
A Grid World MDP

- Agent lives in a 2D grid environment.
- State: Agent's 2D coordinates.
- Actions: N, E, S, W.
- Rewards: +1/−1 at absorbing states.
- Walls block agent's path.
- Actions do not always go as planned. 20% chance that agent drifts one cell left or right of direction of motion (except when blocked by wall).

Figure credits: Pieter Abbeel
Agent lives in a 2D grid environment
Agent lives in a 2D grid environment

- State: Agent’s 2D coordinates
- Actions: N, E, S, W
- Rewards: +1/-1 at absorbing states

Figure credits: Pieter Abbeel
Agent lives in a 2D grid environment

- State: Agent’s 2D coordinates
- Actions: N, E, S, W
- Rewards: +1/-1 at absorbing states

- Walls block agent’s path
- Actions to not always go as planned
  - 20% chance that agent drifts one cell left or right of direction of motion (except when blocked by wall).

Figure credits: Pieter Abbeel
Solving MDPs by finding the **best/optimal policy**
Solving MDPs by finding the **best/optimal policy**

Formally, a **policy** is a mapping from states to actions

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
</tbody>
</table>
- Solving MDPs by finding the **best/optimal policy**

- Formally, a **policy** is a mapping from states to actions
  - Deterministic \( \pi(s) = a \)

\[
\begin{align*}
  n &= |S| \\
  m &= |A|
\end{align*}
\]
Solving MDPs by finding the **best/optimal policy**

Formally, a **policy** is a mapping from states to actions

- Deterministic \( \pi(s) = a \)
Solving MDPs by finding the **best/optimal policy**

- Formally, a **policy** is a mapping from states to actions
  - Deterministic \( \pi(s) = a \)
  - Stochastic \( \pi(a|s) = \mathbb{P}(A_t = a|S_t = s) \)

\[
\begin{align*}
  n &= |S| \\
  m &= |A| \\
\end{align*}
\]
Solving MDPs by finding the **best/optimal policy**

Formally, a **policy** is a mapping from states to actions

- Deterministic \( \pi(s) = a \)
- Stochastic \( \pi(a|s) = P(A_t = a|S_t = s) \)

\[
\begin{align*}
\pi & \\
\pi(s) & = a \\
\pi(a|s) & = P(A_t = a|S_t = s)
\end{align*}
\]
Solving MDPs by finding the best/optimal policy

Formally, a policy is a mapping from states to actions
- Deterministic: \( \pi(s) = a \)
- Stochastic: \( \pi(a|s) = \mathbb{P}(A_t = a|S_t = s) \)

What is a good policy?
- Maximize current reward? Sum of all future rewards?
- Discounted sum of future rewards!
  - Discount factor: \( \gamma \)

Discounted sum of future rewards:
- Worth Now: \( 1 \)
- Worth Next Step: \( \gamma \)
- Worth In Two Steps: \( \gamma^2 \)
Today, we saw

- **MDPs**: Theoretical framework underlying RL, solving MDPs
- **Policy**: How an agents acts at states (to be continued in next lecture)

Next Lecture:

- **Value function (Utility)**: How good is a particular state or state-action pair?
- **Algorithms** for solving MDPs (Value Iteration)
- Departure from known rewards and transitions: **Reinforcement Learning**