AI6101 Introduction to AI and AI Ethics

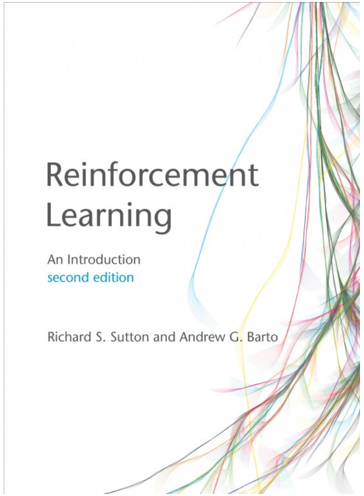
Reinforcement Learning

Assoc Prof Bo AN

www.ntu.edu.sg/home/boan
Email: boan@ntu.edu.sg
Office: N4-02b-55

Lesson Outline

- Reinforcement learning (RL)
- Model-free prediction and control
- Value-based methods
- Advanced materials:
  - Policy gradient methods
  - Exploitation vs exploration

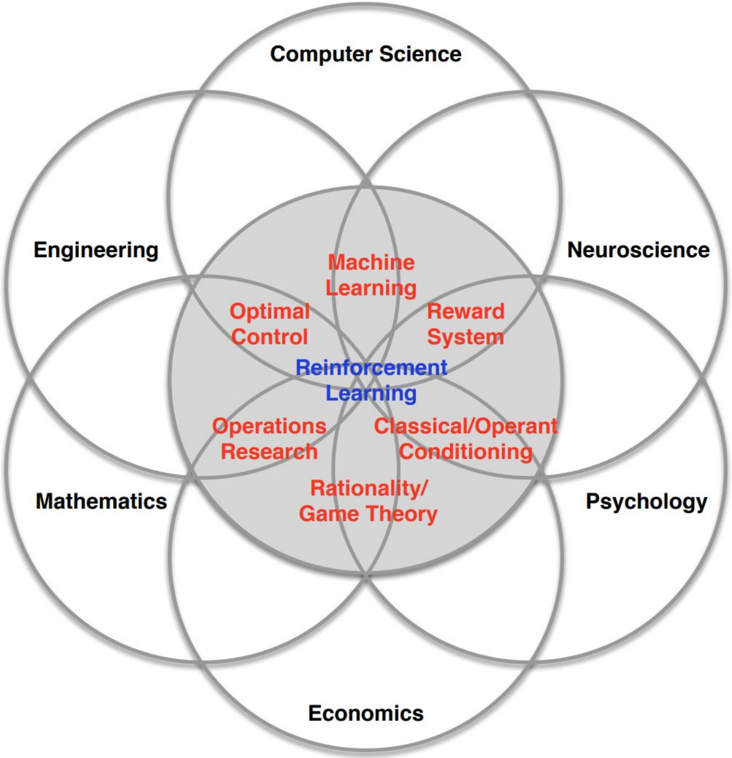


Reinforcement Learning

What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

Many Faces of Reinforcement Learning



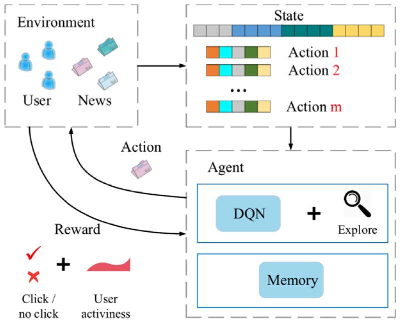
Examples of Reinforcement Learning

Robotic control

AlphaGo

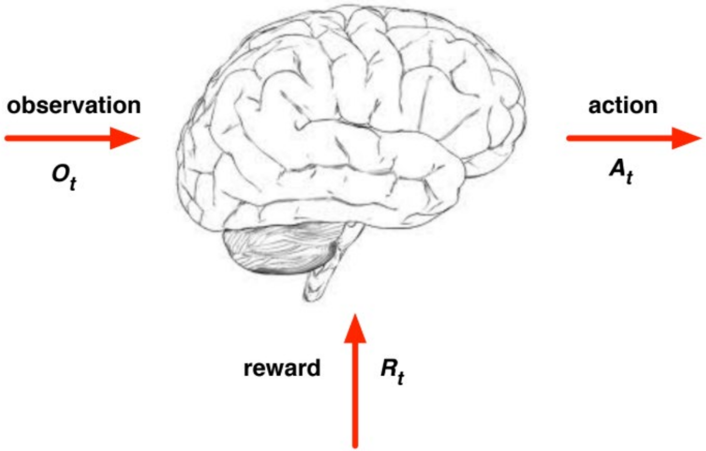
The Game of Go

Video games



Recommendation System

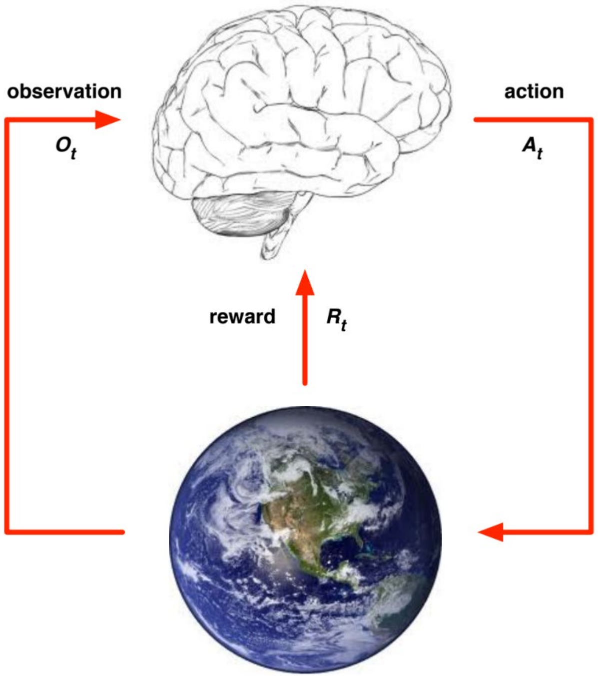
Agent and Environment



- The observation is the perception of the environment for agent
- The action will change the outside environment
- The reward is a scalar value indicates how well agent is doing at step t

The agent’s job is to maximize the cumulative reward

Agent and Environment



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Major Components of an RL Agent

- Policy: agent’s behavior function
- Value function: how good is each state and/or action
- Model: agent’s representation of the environment

Categorizing RL agents

- Value Based
  - No Policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No Value Function
- Actor Critic
  - Policy
  - Value Function

- Model Free
  - Policy and/or Value Function
  - No Model
- Model Based
  - Policy and/or Value Function
  - Model

Model-free Prediction and Control

Normally, we model RL problems as an Markov Decision Process (MDP)

- **States** s, beginning with initial state s_0
- **Actions** a
- **Transition model** $P(s' \mid s, a)$
- **Reward function** $R(s)$

What if the distributions and the transitions of MDP are unkonwn?

- Model-free RL
    - learn from samples

Model-free Prediction: Monte Carlo

- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions / rewards
- MC learns from complete episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to episodic MDPs
  - All episodes must terminate

Monte Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, \ldots, S_k \sim \pi$$

- The return is the total discounted reward:

$$G_t = R_{\mathrm{t+1}} + \gamma R_{t+1} + \ \ldots + \ \gamma^{T-1} R_T$$

- The value function is the expected return:

$$V_\pi(s) = E_\pi[G_t | S_t = s]$$

Model-free Control: TD Learning

Temporal Difference Learning

- TD methods learn directly from episodes of experience
- TD is model-free: no knowledge of MDP transitions / rewards
- TD learns from incomplete episodes, by bootstrapping
- TD updates a guess towards a guess

TD Learning

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, \ldots, S_k \sim \pi$$

- Update value V(St) toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) = V(S_t) + \alpha(\underbrace{\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{Target}} - V(S_t)}_{\text{TD error}})$$

MC vs. TD

- TD can learn before knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn without the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

Value-based Methods

- Learning a Policy $\pi(a|s)$ via a Value function $Q(s,a)$
  - For example, Q-learning and SARSA
- Q-learning
  - Using a Q table to get the best action $argmax_{a^*}\, Q(s,a)$
  - Update Q

$$Q(s,a) \leftarrow (1-\alpha)\cdot Q(s_t,a_t) + \underbrace{\alpha\cdot}_{\text{Learning rate}} \underbrace{(r_t + \gamma\cdot\max_a Q(s_{t+1},a))}_{\text{Learned value}}$$

Value-based Methods: Q-learning

- Q-learning Introduction
  - A model-free off-policy reinforcement learning
  - Learns an optimal action-selection policy for any given MDP
    - The action-selection policy is a table storing Q-values given state and action
    - The action with maximal Q value is chosen as the optimal action given a state input

Given state s_1

	a_0	a_1	a_2
s_0	1.	2	3.
s_1	2	3	1
s_2	3	4	2

Q table

Return action a_1 with maximal Q value

Value-based Methods: Q-learning

- Q-learning training
  - Learning from samples: (S, A, R, S')
  - Via Bellman Equation to update the Q table
  - Update Q table

Off-policy: updating the Q with different policy

$$Q(S,A) = Q(S,A) + \alpha \cdot [R + \gamma \cdot \max_{A'}(Q(S',A')) - Q(S,A))]$$

  - Take actions with epsilon-greedy (for exploration)

$$\hat{\pi} = \arg max_a \hat{Q}(s,a) \quad w.p \; 1-\epsilon$$

Value-based Methods: Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Value-based Methods: SARSA

- SARSA Introduction
  - Similar to Q-learning with some differences
    - On-policy: update the Q-table with the (s, a, r, s') samples generate by the current policy

on-policy: updating the Q with current policy

$$Q(S, A) = Q(S, A) + \alpha \cdot [R + \gamma \cdot \cancel{\max_{A}}(Q(S', A')) - Q(S, A))]$$

The next state and next action in transition samples

  - Epsilon greedy can still be used to output actions like Q-learning

For code, refer to https://www.geeksforgeeks.org/sarsa-reinforcement-learning/

Value-based Methods: SARSA

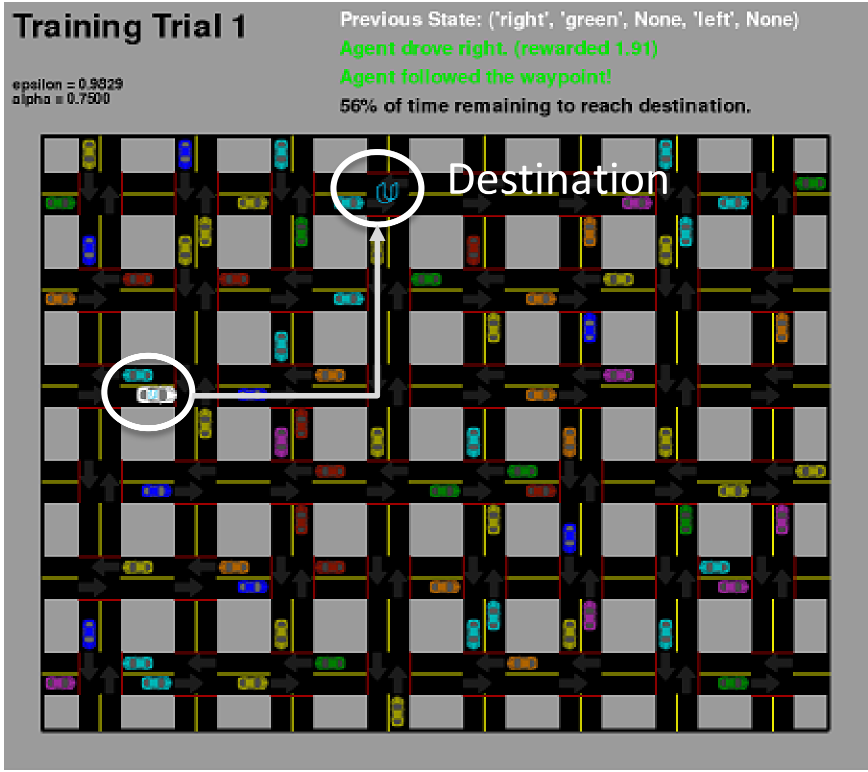
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma Q(S', A') - Q(S, A)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Example: Smart Cab Example

- Training the smart cab
  - Actions: **stop, forward, left, right**
  - State: intersection info
  - Environment: a grid world environment
  - Rewards: proximity to the destination & penalty
  - Transitions: unknown
- Using Q-learning



Available online
https://github.com/GoingMyWay/OpenCourse/tree/master/Udacity/MLND/smartcab
https://www.ritchieng.com/machine-learning-proj-smart-cab/

Example: Smart Cab Example

- The code of the agent

```
class LearningAgent(Agent):
    """An agent that learns to drive in the smartcab world."""

    def __init__(self, env):
        # sets self.env = env, state = None, next_waypoint = None, and a default color
        if not isinstance(env, Environment):
            raise TypeError('invalid type %s' % type(env))

        self.env, self.state, self.next_waypoint = env, None, None
        super(LearningAgent, self).__init__(env)
        self.color = 'red'  # override color
        self.planner = RoutePlanner(self.env, self)  # simple route planner to get next_waypoint
        # TODO: Initialize any additional variables here
        self.q_matrix, self.alpha, self.gamma, self.epsilon = dict(), 0.9, 0.6, 0.1
        self.pre_state, self.pre_action, self.pre_reward = None, None, None
        self.default_q, self.num_success, self.env.acc = 1, 0, 0.0
```

Example: Smart Cab Example

- Output actions

```python
def select_action_q(self, state):
    # random action selction
    if random.random() < self.epsilon:
        action = random.choice(self.env.valid_actions)
        return ActionQValue(action, self.q_matrix.get((state, action), self.default_q))
    else:
        # output action argmax_a'{Q(s',a')} from the Q-matrix
        _valid_actions = self.env.valid_actions
        random.shuffle(_valid_actions)
        _max_q = max([self.q_matrix.get((state, _action), self.default_q) for _action in _valid_actions])
        for _action in _valid_actions:
            if self.q_matrix.get((state, _action), self.default_q) == _max_q:
                return ActionQValue(_action, _max_q)
```

- Interact with the environment (no code)

Example: Smart Cab Example

- Updating Q table

```
# TODO: Learn policy based on state, action, reward
self.q_matrix[(self.pre_state, self.pre_action)] = \
    (1 - self.alpha) * self.q_matrix[(self.pre_state, self.pre_action)] + \
    self.alpha * (self.pre_reward + self.gamma * self.select_action_q(self.state).q_value)
```

Advanced Materials

Policy Gradient Methods: Background

- How do value function work as a policy?
  - Output actions with the best Q values
- Can we directly learn a policy mapping states to actions?
- Policy gradient methods
  - Learn a *parameterized* policy that can select actions without consulting a value function
  - Use $\pi(a|s,\boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ for the probability that action *a* is taken at time *t* given that the environment is in state *s* at time *t*
  - The value functions can also be parameterized as $\hat{v}(s,\mathbf{w})$ with parameters $\mathbf{w} \in \mathbb{R}^{d}$ (optional)
  - Update the parameters by gradient ascent given some performance measures $J(\boldsymbol{\theta})$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}$$

Policy Gradient Methods: Linear Example

- A linear function approximation example of policy gradient methods
  - Parameters of policy function of a linear function, soft-max policy

$$\theta = \{\theta_0, \theta_1, \theta_2\} \quad \text{action} = \{a_0, a_1, a_2\} \quad s = \{s_0, s_1\}$$

$$\pi(s, a_i) = \frac{e^{a_i\theta_0 + s_0\theta_1 + s_1\theta_2}}{\sum_{j=0}^{|A|} e^{a_j\theta_0 + s_0\theta_1 + s_1\theta_2}}$$

Sampling actions with these probability

  - We introduce optimization rules in the following slides
- Deep Neural networks can also be used as the approximation function
  - Deep Reinforcement Learning (DRL)

Policy Gradient Methods

- Policy gradient (PG) methods model and optimize the policy directly

$$\pi_\theta(s, a) = \mathbb{P}\left[a \mid s, \theta\right]$$

- By maximizing performance measure w.r.t π_θ

$$J(\theta) = V^{\pi_\theta} = E[R]$$

Policy Gradient Methods

- Policy gradient (PG) methods model and optimize the policy directly
- The policy is modeled with a parameterized function respect to θ, $\pi_\theta(a|s)$

Performance measure

Transition function

$$J(\theta) = \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \boxed{\pi_\theta(a|s)}\boxed{Q^\pi(s,a)}$$

Value function

Gradients

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \pi_\theta(a|s)$$
$$\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s)$$

Policy Gradient Methods: REINFORCE

- REINFORCE (Monte-Carlo policy gradient) relies on an estimated return by Monte-Carlo methods using episode samples to update the policy parameter θ

$$
\begin{aligned}
\nabla_\theta J(\theta) &\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s) \\
&= \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\
&= \mathbb{E}_\pi[Q^\pi(s,a) \nabla_\theta \ln \pi_\theta(a|s)] && \text{; Because } (\ln x)' = 1/x \\
&= \mathbb{E}_\pi[G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)] && \text{; Because } Q^\pi(S_t, A_t) = \mathbb{E}_\pi[G_t|S_t, A_t]
\end{aligned}
$$

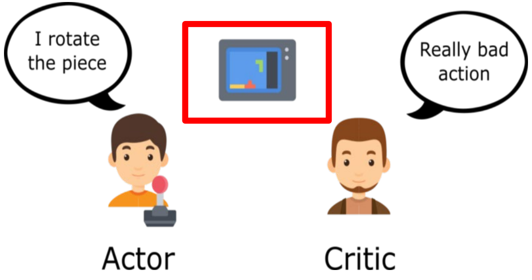
$$\boxed{G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}}$$

Policy Gradient Methods: REINFORCE

1. Initialize the policy parameter θ at random.
2. Generate one trajectory on policy π_θ: $S_1, A_1, R_2, S_2, A_2, \ldots, S_T$.
3. For t=1, 2, … , T:
    1. Estimate the the return G_t;
    2. Update policy parameters: $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$

Policy Gradient Methods: Actor-Critic

- Actor-critic methods consist of two models
  - Critic updates the value function parameters w and depending on the algorithm it could be action-value $Q_w(a|s)$ or state-value $V_w(s)$
  - Actor updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic



1. Initialize s, θ, w at random; sample $a \sim \pi_\theta(a|s)$.
2. For $t = 1 \dots T$:
   1. Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$;
   2. Then sample the next action $a' \sim \pi_\theta(a'|s')$;
   3. Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s)$;
   4. Compute the correction (TD error) for action-value at time t:
      $\boxed{\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)}$
      and use it to update the parameters of action-value function:
      $\boxed{w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)}$
   5. Update $a \leftarrow a'$ and $s \leftarrow s'$.

Exploitation vs Exploration

- Online decision-making involves a fundamental choice:
  - Exploitation Make the best decision given current information
  - Exploration Gather more information
- The best long-term strategy may involve short-term sacrifices
- Gather enough information to make the best overall decisions

Examples

Restaurant Selection

- Exploitation Go to your favorite restaurant
- Exploration Try a new restaurant

Online Banner Advertisements

- Exploitation Show the most successful advert
- Exploration Show a different advert

Oil Drilling

- Exploitation Drill at the best known location
- Exploration Drill at a new location

Game Playing

- Exploitation Play the move you believe is best
- Exploration Play an experimental move

Principles

Naive Exploration

- Add noise to greedy policy (e.g. ϵ-greedy)

Optimistic Initialization

- Assume the best until proven otherwise

Optimism in the Face of Uncertainty

- Prefer actions with uncertain values

Probability Matching

- Select actions according to probability they are best

Information State Search

- Lookahead search incorporating value of information