Reinforcement learning

Yifeng Tao
School of Computer Science
Carnegie Mellon University

Slides adapted from Matt Gormley, Eric Xing

Learning Paradigms

Paradigm	Data
Supervised	$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ $\quad \mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$
↪ Regression	$y^{(i)} \in \mathbb{R}$
↪ Classification	$y^{(i)} \in \{1, \ldots, K\}$
↪ Binary classification	$y^{(i)} \in \{+1, -1\}$
↪ Structured Prediction	$\mathbf{y}^{(i)}$ is a vector
Unsupervised	$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ $\quad \mathbf{x} \sim p^*(\cdot)$
Semi-supervised	$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$
Online	$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots\}$
Active Learning	$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ and can query $y^{(i)} = c^*(\cdot)$ at a cost
Imitation Learning	$\mathcal{D} = \{(s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \ldots\}$
Reinforcement Learning	$\mathcal{D} = \{(s^{(1)}, a^{(1)}, r^{(1)}), (s^{(2)}, a^{(2)}, r^{(2)}), \ldots\}$

[Slide from Matt Gormley]

Examples of Reinforcement Learning

- How should a robot behave so as to optimize its "performance"? (Robotics)

- How to automate the motion of a helicopter? (Control Theory)

- How to make a good chess-playing program? (Artificial Intelligence)

[Slide from Matt Gormley]

Robot in a room

actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step

- what's the strategy to achieve max reward?
- what if the actions were NOT deterministic?

[Slide from Eric Xing]

History of Reinforcement Learning

- Roots in the psychology of animal learning (Thorndike,1911).
- Another independent thread was the problem of optimal control, and its solution using dynamic programming (Bellman, 1957).
- Idea of temporal difference learning (on-line method), e.g., playing board games (Samuel, 1959).
- A major breakthrough was the discovery of Q-learning (Watkins, 1989).

[Slide from Eric Xing]

What is special about RL?

- RL is learning how to map states to actions, so as to maximize a numerical reward over time.
- Unlike other forms of learning, it is a multistage decision-making process (often Markovian).
- An RL agent must learn by trial-and-error. (Not entirely supervised, but interactive)
- Actions may affect not only the immediate reward but also subsequent rewards (Delayed effect).

[Slide from Eric Xing]

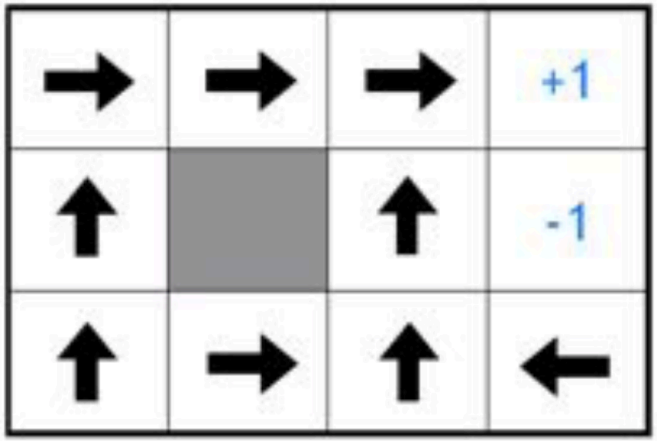
Elements of RL

- A policy
  - A map from state space to action space.
  - May be stochastic.
- A reward function
  - It maps each state (or, state-action pair) to a real number, called reward.
- A value function
  - Value of a state (or, state-action pair) is the total expected reward, starting from that state (or, state-action pair).

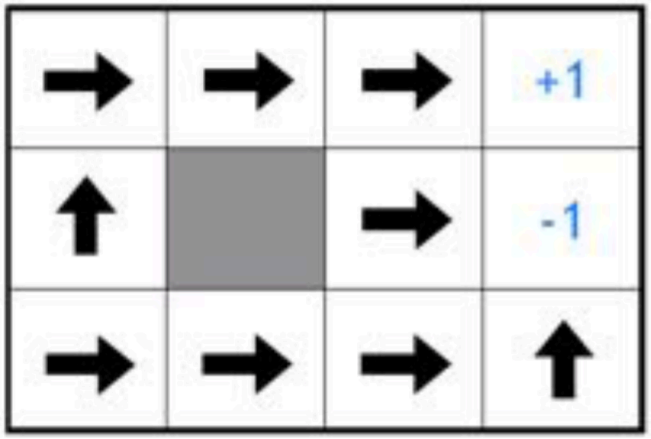
[Slide from Eric Xing]

Policy

- Reward for each step: -0.1



- Reward for each step -2



[Slide from Eric Xing]

The Precise Goal

- To find a policy that maximizes the Value function.
  - transitions and rewards usually not available
- There are different approaches to achieve this goal in various situations.
- Value iteration and Policy iteration are two more classic approaches to this problem. But essentially both are dynamic programming.
- Q-learning is a more recent approaches to this problem. Essentially it is a temporal-difference method.

[Slide from Eric Xing]

Reinforcement Learning

- Train a policy to maximize the discounted, cumulative reward R_{t0}:

$$R_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t$$

- γ: should be a constant between 0 and 1

- Bellman equation (deterministic):

$$Q^{\pi}(s,a) = r + \gamma Q^{\pi}(s', \pi(s'))$$

- Bellman equation (stochastic):

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s')$$

[Slide from Matt Gormley]

Value Iteration

Algorithm 1 Value Iteration

procedure VALUEITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
  Initialize value function $V(s) = 0$ or randomly
  while not converged **do**
    for $s \in \mathcal{S}$ **do**
      for $a \in \mathcal{A}$ **do**
        $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')$
      $V(s) = \max_a Q(s,a)$
  Let $\pi(s) = \text{argmax}_a Q(s,a), \ \forall s$
  return π

[Slide from Matt Gormley]

Value Iteration Convergence

very abridged

Theorem 1 (Bertsekas (1989))
V converges to V^*, if each state is visited infinitely often

Holds for both asynchronous and sychronous updates

Theorem 2 (Williams & Baird (1993))
if $max_s|V^{t+1}(s) - V^t(s)| < \epsilon$
then $max_s|V^{t+1}(s) - V^*(s)| < \frac{2\epsilon\gamma}{1-\gamma}, \forall s$

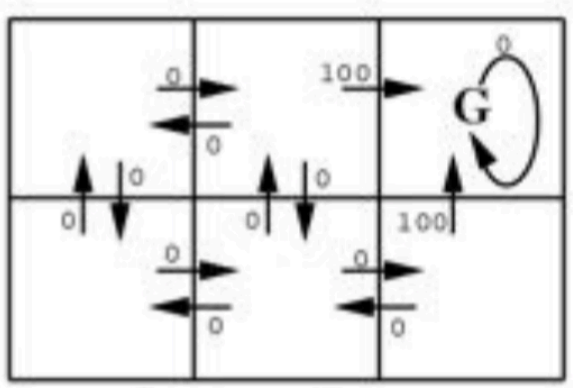
Provides reasonable stopping criterion for value iteration

Theorem 3 (Bertsekas (1987))
greedy policy will be optimal in a finite number of steps (even if not converged to optimal value function!)

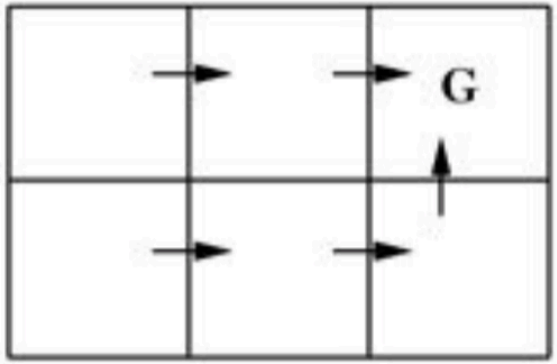
Often greedy policy converges well before the value function

[Slide from Matt Gormley]

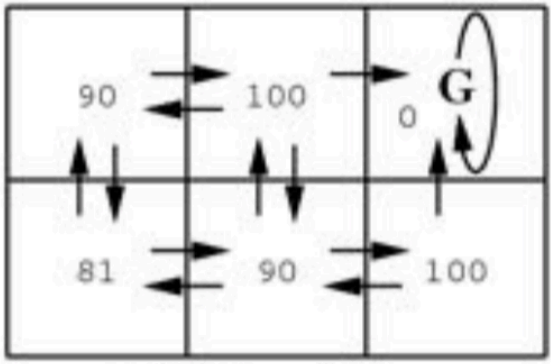
Example: Robot Localization



$r(s, a)$ (immediate reward) values



One optimal policy



$V^*(s)$ values

[Slide from Matt Gormley]

Value Iteration Variants

- Variant 1: w/ Q(s,a) table

Algorithm 1 Value Iteration

procedure VALUEITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
  Initialize value function $V(s) = 0$ or randomly
  while not converged **do**
    for $s \in \mathcal{S}$ **do**
      for $a \in \mathcal{A}$ **do**
        $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s')$
      $V(s) = \max_a Q(s,a)$
  Let $\pi(s) = \text{argmax}_a\, Q(s,a),\ \forall s$
  return π

→

- Variant 2: w/o Q(s,a) table

Algorithm 1 Value Iteration

procedure VALUEITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
  Initialize value function $V(s) = 0$ or randomly
  while not converged **do**
    for $s \in \mathcal{S}$ **do**
      $V(s) = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s')$
  Let $\pi(s) = \text{argmax}_a\, R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V(s'),\ \forall s$
  return π

[Slide from Matt Gormley]

Synchronous vs. Asynchronous Value Iteration

Algorithm 1 Asynchronous Value Iteration

procedure ASYNCHRONOUSVALUEITERATION($R(s,a), p(\cdot|s,a)$)
  Initialize value function $V(s)^{(0)} = 0$ or randomly
  $t = 0$
  while not converged **do**
    for $s \in \mathcal{S}$ **do**
      $V(s)^{(t+1)} = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')^{(t)}$
      $t = t + 1$
  Let $\pi(s) = \text{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s'), \ \forall s$
  return π

asynchronous updates: compute and update V(s) for each state one at a time

Algorithm 1 Synchronous Value Iteration

procedure SYNCHRONOUSVALUEITERATION($R(s,a), p(\cdot|s,a)$)
  Initialize value function $V(s)^{(0)} = 0$ or randomly
  $t = 0$
  while not converged **do**
    for $s \in \mathcal{S}$ **do**
      $V(s)^{(t+1)} = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')^{(t)}$
    $t = t + 1$
  Let $\pi(s) = \text{argmax}_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s'), \ \forall s$
  return π

synchronous updates: compute all the fresh values of V(s) from all the stale values of V(s), then update V(s) with fresh values

[Slide from Matt Gormley]

Value Iteration Convergence

very abridged

Theorem 1 (Bertsekas (1989))
V converges to V^*, if each state is visited infinitely often

Holds for both asynchronous and sychronous updates

Theorem 2 (Williams & Baird (1993))
if $max_s|V^{t+1}(s) - V^t(s)| < \epsilon$
then $max_s|V^{t+1}(s) - V^*(s)| < \frac{2\epsilon\gamma}{1-\gamma}, \forall s$

Provides reasonable stopping criterion for value iteration

Theorem 3 (Bertsekas (1987))
greedy policy will be optimal in a finite number of steps (even if not converged to optimal value function!)

Often greedy policy converges well before the value function

[Slide from Matt Gormley]

Policy Iteration

Algorithm 1 Policy Iteration

1: **procedure** POLICYITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)

2: Initialize policy π randomly

3: **while** not converged **do**

4: Solve Bellman equations for fixed policy π

$$V^{\pi}(s) = R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,\pi(s)) V^{\pi}(s'), \ \forall s$$

5: Improve policy π using new value function

$$\pi(s) = \operatorname*{argmax}_{a} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{\pi}(s')$$

6: **return** π

[Slide from Matt Gormley]

Policy Iteration

Algorithm 1 Policy Iteration

1: **procedure** POLICYITERATION($R(s,a)$ ..., $p(\cdot|s,a)$ transition probabilities)

2: Initialize policy π randomly

3: **while** not converged **do**

4: Solve Bellman equations for fixed policy π

$$V^{\pi}(s) = R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,\pi(s)) V^{\pi}(s'), \ \forall s$$

5: Improve policy π using new value function

$$\pi(s) = \operatorname*{argmax}_{a} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{\pi}(s')$$

6: **return** π

Compute value function for fixed policy is easy

System of |S| equations and |S| variables

Greedy policy w.r.t. current value function

Greedy policy might **remain the same** for a particular state if there is no better action

[Slide from Matt Gormley]

Value Iteration vs. Policy Iteration

- Value iteration requires $O(|A|\,|S|^2)$ computation per iteration

- Policy iteration requires $O(|A|\,|S|^2 + |S|^3)$ computation per iteration
- In practice, policy iteration converges in fewer iterations

Algorithm 1 Value Iteration

procedure VALUEITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
  Initialize value function $V(s) = 0$ or randomly
  while not converged **do**
    for $s \in \mathcal{S}$ **do**
      for $a \in \mathcal{A}$ **do**
        $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s')$
      $V(s) = \max_a Q(s,a)$
  Let $\pi(s) = \text{argmax}_a\, Q(s,a),\ \forall s$
  return π

Algorithm 1 Policy Iteration

procedure POLICYITERATION($R(s,a)$ reward function, $p(\cdot|s,a)$ transition probabilities)
  Initialize policy π randomly
  while not converged **do**
    Solve Bellman equations for fixed policy π

$$V^\pi(s) = R(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,\pi(s)) V^\pi(s'),\ \forall s$$

    Improve policy π using new value function

$$\pi(s) = \underset{a}{\text{argmax}}\, R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s')$$

  return π

[Slide from Matt Gormley]

Deep Q-Learning

Question: *What if our state space S is too large to represent with a table?*

Examples:

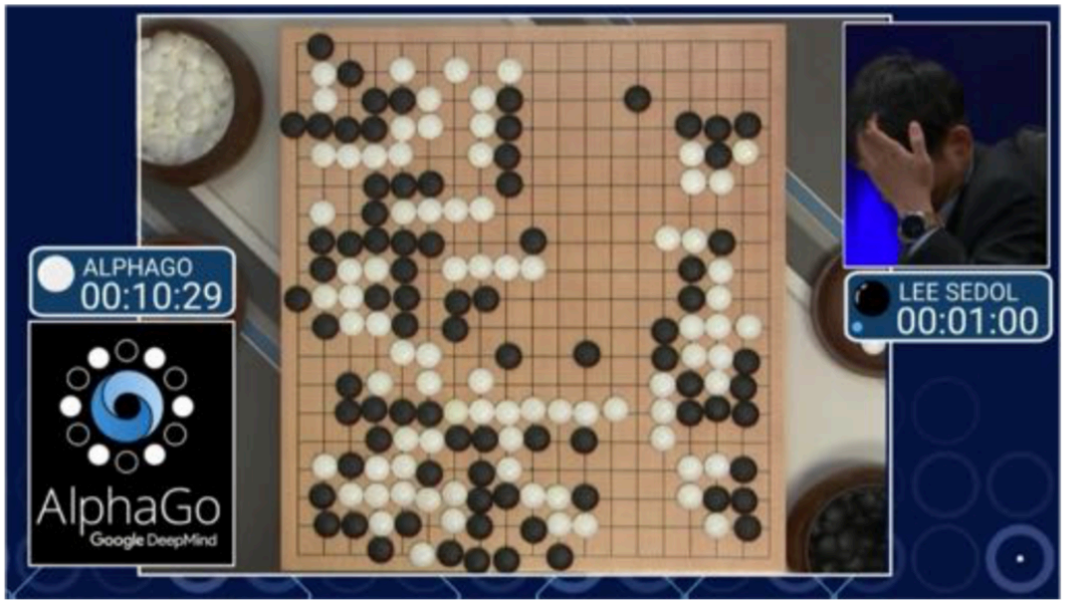
- s_t = pixels of a video game
- s_t = continuous values of a sensors in a manufacturing robot
- s_t = sensor output from a self-driving car

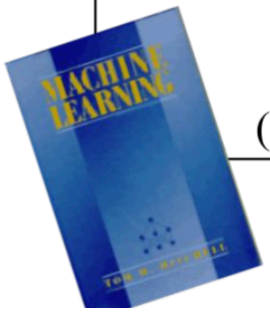
Answer: Use a parametric function to approximate the table entries

[Slide from Matt Gormley]

TD Gammon → Alpha Go

Learning to beat the masters at board games

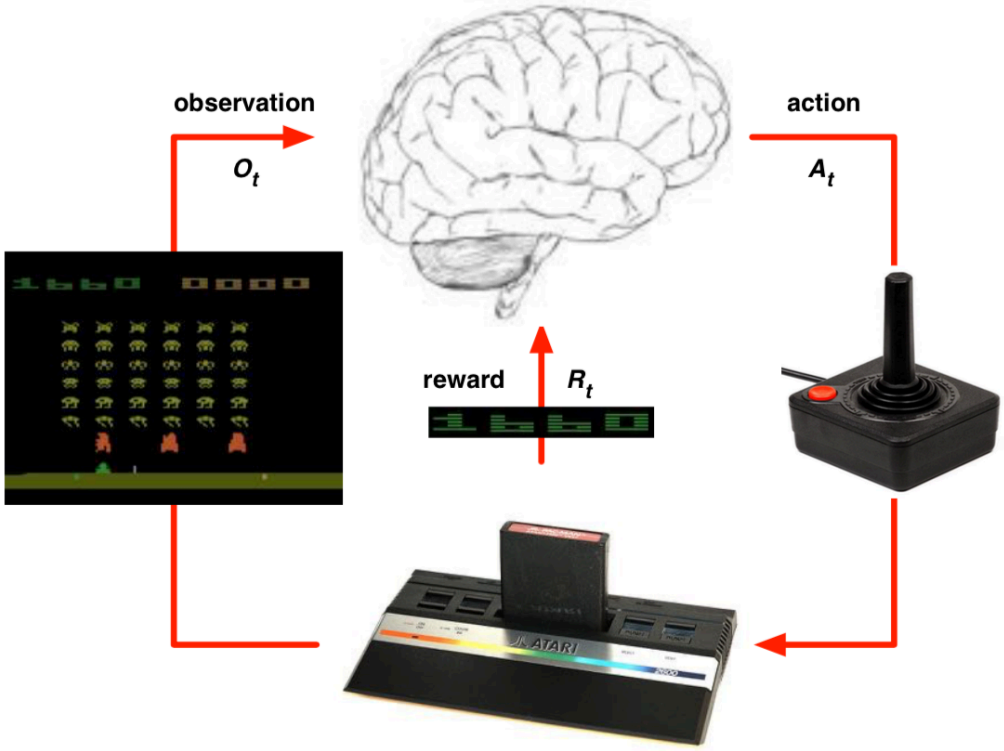
THEN	NOW
"…the world's top computer program for backgammon, TD-GAMMON (Tesauro, 1992, 1995), learned its strategy by playing over one million practice games against itself…" (Mitchell, 1997)	



[Slide from Matt Gormley]

Playing Atari with Deep RL

- Setup: RL system observes the pixels on the screen
- It receives rewards as the game score
- Actions decide how to move the joystick / buttons



[Slide from Matt Gormley]

Deep Q-Network (DQN) algorithm

- Goal: train *Q(s, a)* to fit the unknown reward (Q) function.
- Then, best policy:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q^*(s, a)$$

- Bellman equation:

$$Q^{\pi}(s, a) = r + \gamma Q^{\pi}(s', \pi(s'))$$

- Temporal difference error:

$$\delta = Q(s, a) - (r + \gamma \max_{a} Q(s', a))$$

- Huber loss:

$$\mathcal{L} = \frac{1}{|B|} \sum_{(s,a,s',r) \in B} \mathcal{L}(\delta)$$

$$\text{where} \quad \mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

- *B*: a batch of transitions, sampled from the replay memory

[Slide from https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html]

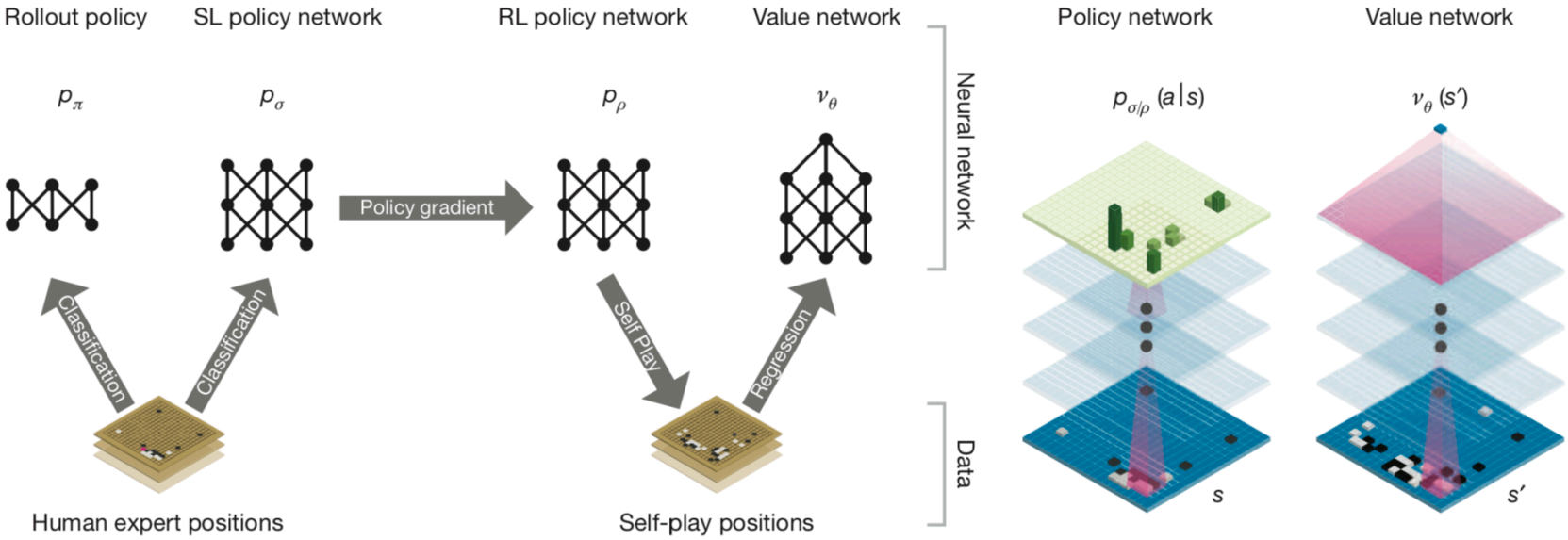
Experience Replay

- **Problems** with online updates for Deep Q-learning:
  - not i.i.d. as SGD would assume
  - quickly forget rare experiences that might later be useful to learn from
- **Uniform Experience Replay** (Lin, 1992):
  - Keep a *replay memory* $D = \{e_1, e_2, \ldots, e_N\}$ of N most recent experiences $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$
  - Alternate two steps:
    1. Repeat T times: randomly sample e_i from D and apply a Q-Learning update to e_i
    2. Agent selects an action using epsilon greedy policy to receive new experience that is added to D
- **Prioritized Experience Replay** (Schaul et al, 2016)
  - similar to Uniform ER, but sample so as to prioritize experiences with high error

[Slide from Matt Gormley]

Alpha Go

- State space is too large to represent explicitly since # of sequences of moves is $O(b^d)$
  - Go: $b=250$ and $d=150$
  - Chess: $b=35$ and $d=80$
- Key idea:
  - Define a neural network to approximate the value function
  - Train by policy gradient



[Slide from Matt Gormley]

Constructing Genetic Association Database

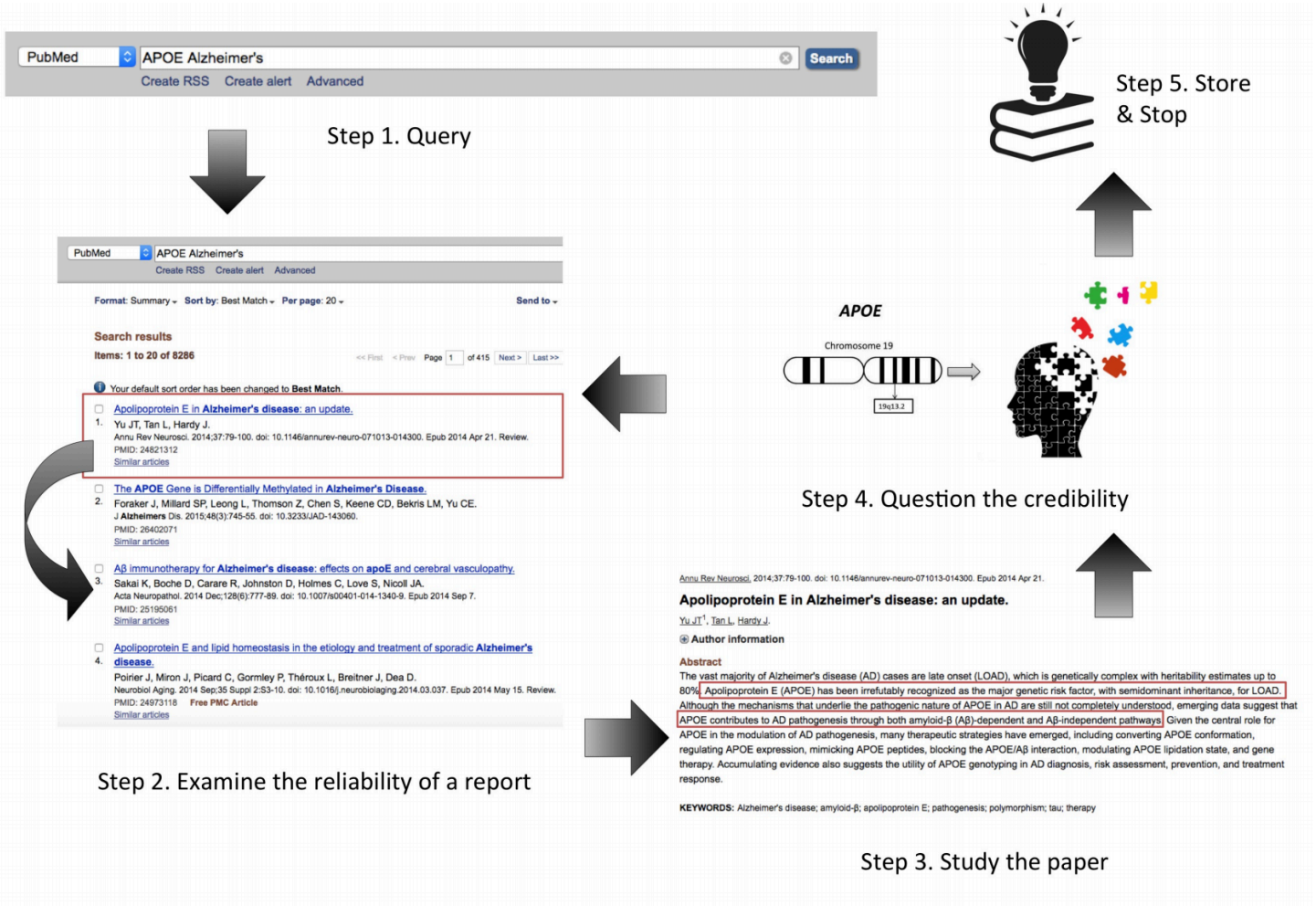


Fig. 1: Overview of Eir's possible behaviors

[Slide from Wang et al.]

Constructing Genetic Association Database

$$Q_{i+1}(s,a) = \mathbf{E}[r + \gamma \max_{a'} Q_i(s', a')|s, a]$$

Actions: Action (we use a to denote action throughout this paper) is a set of Eir's behaviors to simulate a real researcher, including

1. Query the search engine.
2. Evaluate whether the article is reliable.
3. Read the article for detailed information.
4. Exam credibility of the information and querying again.
5. Stop.

States: The state s in the MDP describes the research status of Eir, possible candidate states include the ones that are precedent or after each aforementioned action. There are only a countable number of actions, but we use continuous real-valued vector to represent each state so that we could have a better modeling power to distinguish Eir's research status after each action. The state is constructed with a variety of information, including the embedding vector that the Bidirectional LSTM yields, the confidence of biomedical text mining module, the confidence of selecting an article to read, etc.

Rewards: The reward function is chosen to maximize the intermediate paper selection accuracy and final extraction accuracy together while minimizing the number of queries. The accuracy component is calculated using the difference between the accuracy of the current and the previous set of entity values.

Transitions: Transition $T(s'|s, a)$ is modeled as a function of how the next state s' is updated given the current state s and action a taken.

[Slide from Wang et al.]

Take home message

- Reward, value, and policy in reinforcement learning
- Value iteration and convergence guarantee
- Policy iteration
- Deep Q-learning uses neural network to approximate Q-functions

References

- Matt Gormley. 10601 Introduction to Machine Learning: http://www.cs.cmu.edu/~mgormley/courses/10601/index.html
- Eric Xing, Tom Mitchell. 10701 Introduction to Machine Learning: http://www.cs.cmu.edu/~epxing/Class/10701-06f/
- Adam Paszke. Reinforcement Learning (DQN) Tutorial: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html
- Haohan Wang et al. 2019: Automatic Human-like Mining and Constructing Reliable Genetic Association Database with Deep Reinforcement Learning