About us

Nakamoto & Turing Labs is a
NYC-based research institute focused on
investment, consulting service and **education** in
the field of blockchain and AI

We have offices in:
New York, Hangzhou, Shenzhen, Singapore

Website: https://www.ntlabs.io

NAKAMOTO
&TURING
LABS

Partnerships

DeFi
Cloud storage
Logistics finance
Distributed storage
Crypto wallet
Supply chain

NAKAMOTO
&TURING
LABS

Become our ambassador

We are actively looking for blockchain/AI enthusiasts to host meetup events in different cities as part of the Nakamoto & Turing Labs Network.

Learn more: https://www.ntlabs.io/ambassador-program

NAKAMOTO &TURING LABS

Have something exciting to share? Be our speaker!

If you are an expert on a specific blockchain/AI-related topic that you are passionate about and would love to share and discuss with our community, don't hesitate to let us know!

Email us a brief talk outline to **milo@ntlabs.io**. From there, we can schedule a meeting and figure out how to make this happen!

NAKAMOTO &TURING LABS

Stay informed of our latest events

Ask questions
Request resources
Share information

milo@ntlabs.io

NAKAMOTO &TURING LABS

Reinforcement Learning Explained

overview and applications

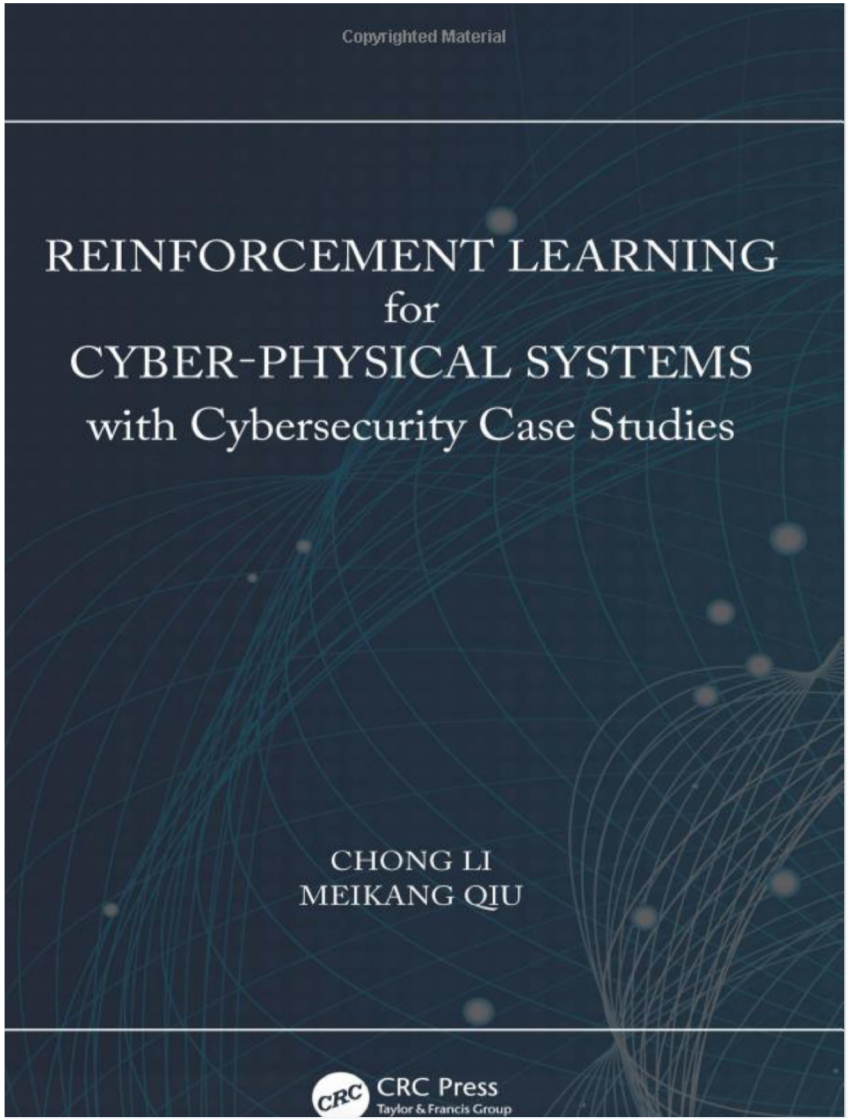
Chong Li

Outline

- Introduction
- Reinforcement Learning Framework & Problem
- Model-based Reinforcement Learning
- Model-free Reinforcement Learning
- Deep Reinforcement Learning
- Applications

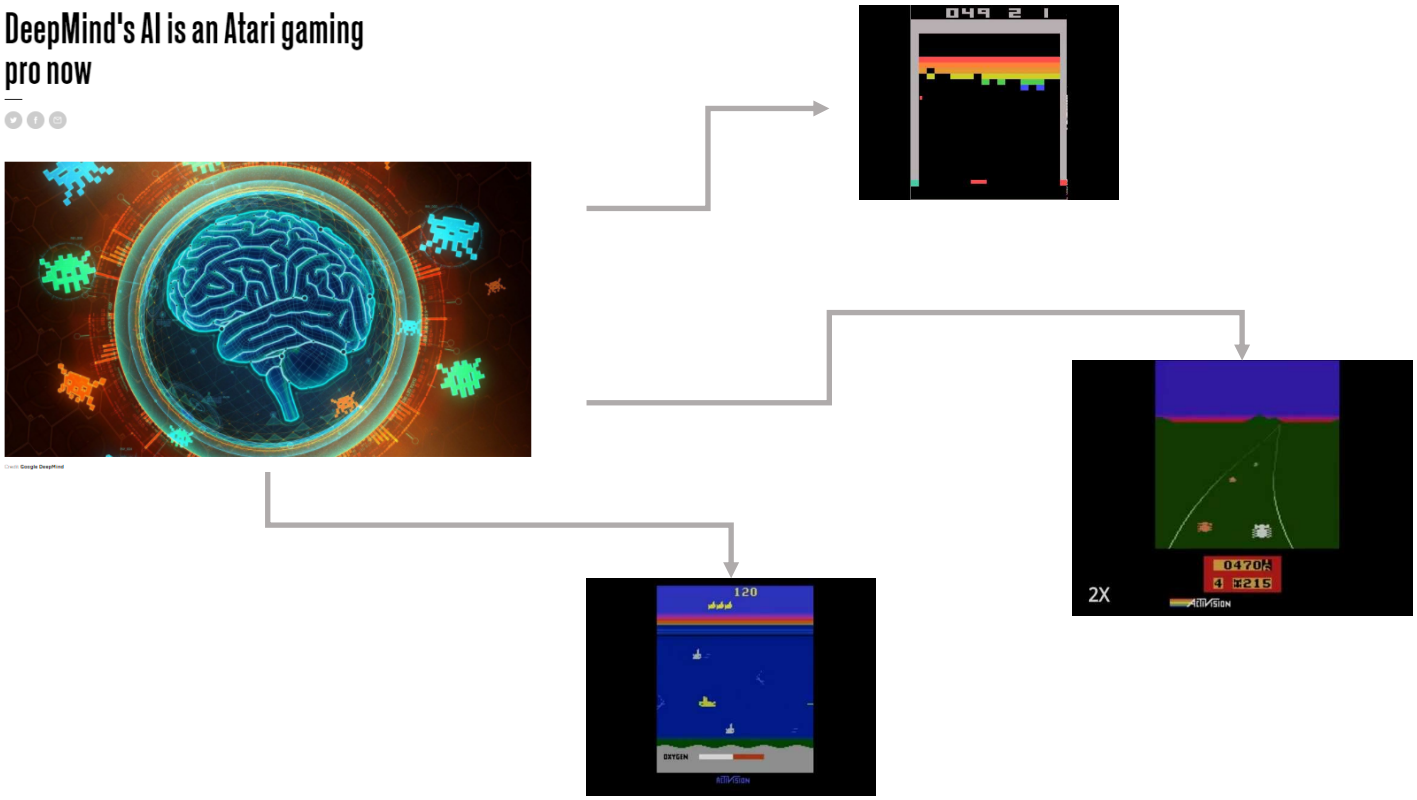
Outline

- Introduction
- Reinforcement Learning Framework & Problem
- Model-based Reinforcement Learning
- Model-free Reinforcement Learning
- Deep Reinforcement Learning
- Applications


Copyrighted Material
REINFORCEMENT LEARNING
for
CYBER-PHYSICAL SYSTEMS
with Cybersecurity Case Studies
CHONG LI
MEIKANG QIU
CRC Press
Taylor & Francis Group

Significant RL Success

- Achieved human-level performance on Atari games from pixel-level visual input, in conjunction with deep learning (Google DeepMind 2015)



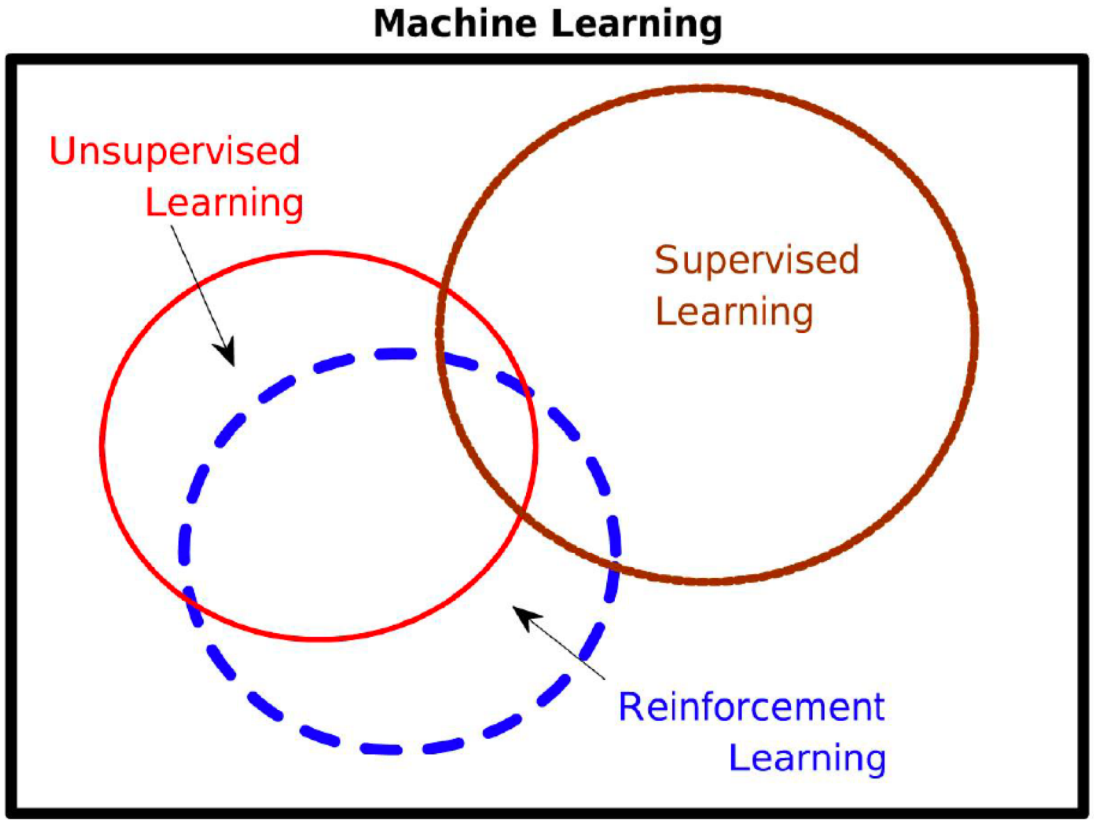
Significant RL Success

- AlphaGo was developed by Google DeepMind in London in October 2015
- It became the first Computer Go program to beat a human professional Go player on a full-sized 19×19 board. In March 2016, it beat Lee Sedol in a five game match
- AlphaGo Zero created without using data from human, and stronger.

What is RL?

- Agent-oriented learning — learning by interacting with an environment to achieve a goal
  - more realistic and ambitious than other kinds of machine learning
- Learning by trial and error, with only delayed evaluative feedback (reward)
  - the kind of machine learning most like natural learning
  - learning that can tell for itself when it is right or wrong
- Example: a child develops his personality receiving feedback from his surroundings

Branches of Machine Learning



Venn diagram describing the relationship among various forms of machine learning

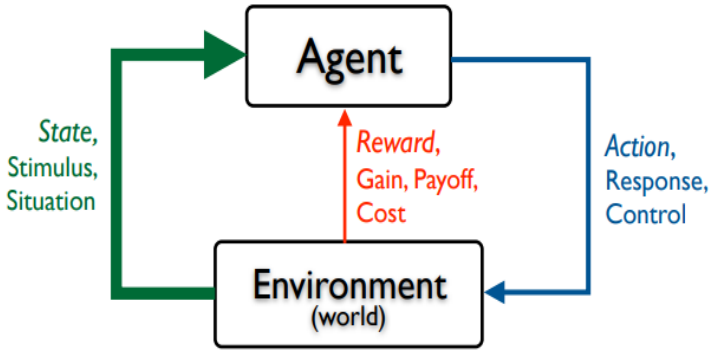
Outline

- Introduction
- Reinforcement Learning Framework & Problem
- Model-based Reinforcement Learning
- Model-free Reinforcement Learning
- Deep Reinforcement Learning
- Applications

Basic RL Interface

- At each time step, an agent
  - executes action
  - receives observation of the environment
  - receives reward from the environment

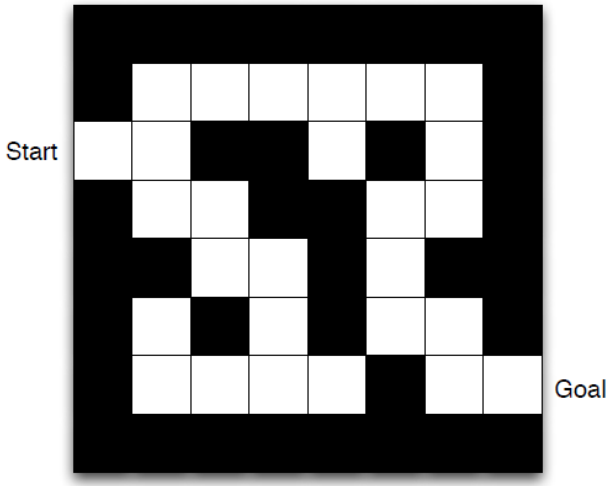
- At each time step, the environment
  - receives action from the agent
  - emits observation/changes its own states
  - emits reward to the agent



Elements of RL – State

- State: the information used to determine what happens next.

- Maze example: state *s* = agent's location



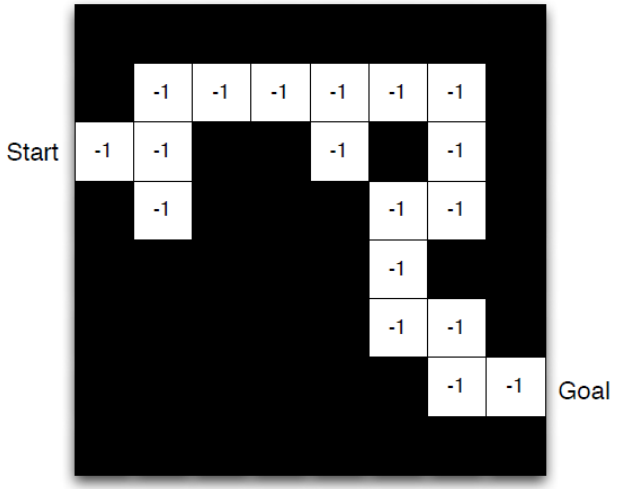
Elements of RL – Reward and Return

- Reward is a mapping from each perceived state of the environment to a single number, indicating the intrinsic desirability of that state
- Reward is immediate
- Return is a cumulative sequence of received rewards after a given time step
- Finite step return:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

- Discounted return: $0 \leq \gamma \leq 1$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$



Reward： -1 for each move

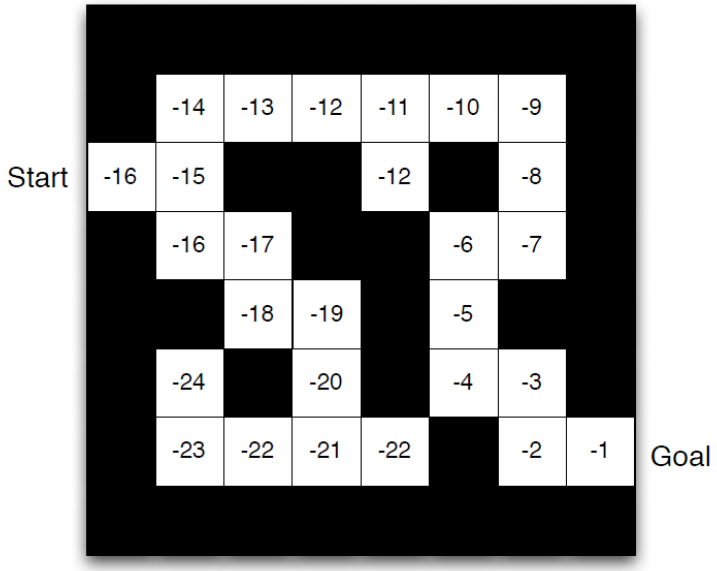
Elements of RL – Value Function

- Functions of states (or of state-action pairs) that estimate *how good* it is for the agent to be in a given state
- "how good" refers to the expected return
- The value of a state is defined formally as

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right]$$

- The value of an action-state pair is defined formally as

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s, A_t = a\right]$$



Numbers represent value of each state

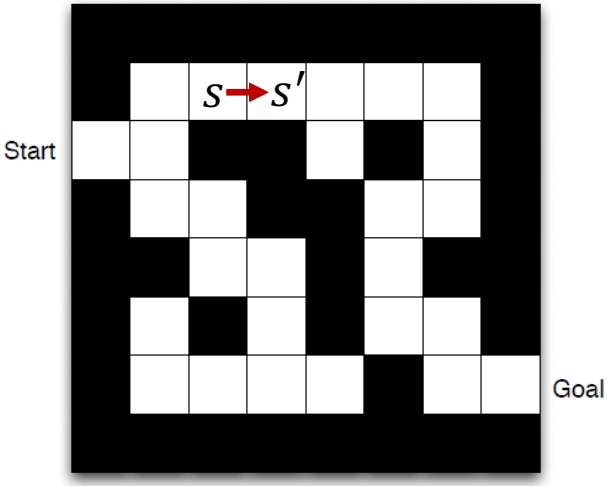
Elements of RL – Model (Optional)

- Mimic the behavior of the environment
- Used for planning (decide on a course of actions by considering future situations before experienced)
- Model:

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

- Maze example:

$$P[s'|s, a = \ move\ toward\ the\ east] = 1$$
$$P[s'|s, a = \ move\ toward\ the\ north] = 0$$
$$P[s'|s, a = \ move\ toward\ the\ south] = 0$$
$$P[s'|s, a = \ move\ toward\ the\ west] = 0$$
$$R = E[R|s, a = \ move\ toward\ the\ east] = -1$$



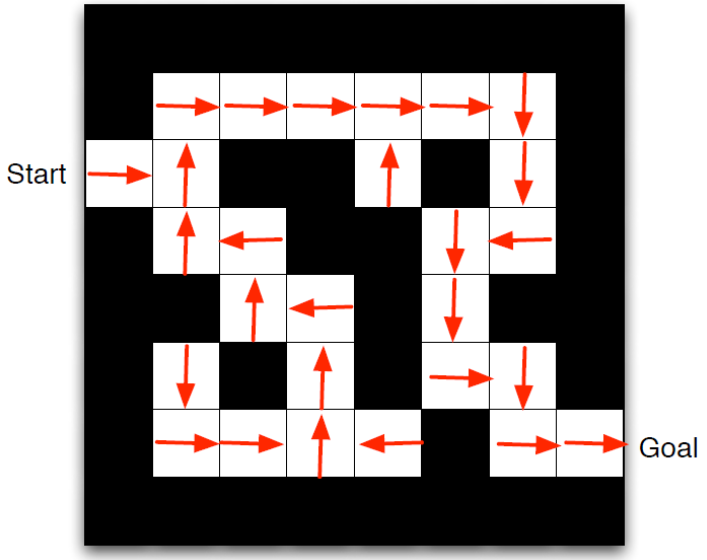
The maze map

Elements of RL - Policy

- A policy, π , is a mapping from perceived states of the environment to (the probability of) actions $a \in \mathcal{A}(s)$ to be taken when in those states.

$$s \in \mathcal{S} \longrightarrow \pi(a|s)$$

- can be stochastic
- In maze example, actions: N, E, S, W



Arrows represent policy (action) for each state

What is RL problem?

- RL problem is a considerable abstraction of the problem of goal-directed learning from interaction with the environment
- RL methods/solutions specify how the agent changes its policy as a result of its experience.
- The agent's goal is to maximize the total amount of reward it receives over the long run

MDP

- A Markov decision process (MDP) is an environment in which all states are Markov
- RL essentially solves a MDP problem.

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

Outline

- Introduction
- Reinforcement Learning Framework & Problem
- Model-based Reinforcement Learning
- Model-free Reinforcement Learning
- Deep Reinforcement Learning
- Applications

What is Model-based RL?

- Model-based RL methods always assume a perfect mathematical model on the environment's dynamics.
- Optimal action can be derived for each state.
- Advantage: fewer samples to complete training.
- Drawback: highly rely on the model hypothesis
- Solution: dynamic programming

What is dynamic programming (DP)

- DP refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a MDP
- Classical DP algorithms are of limited utility in RL. Why?
  - Need perfect model
  - Great computational expense
- Still important theoretically
- DP-based RL algorithms:
  - Policy iteration
  - Value iteration

Outline

- Introduction
- Reinforcement Learning Framework & Problem
- Model-based Reinforcement Learning
- Model-free Reinforcement Learning
- Deep Reinforcement Learning
- Applications

What is Model-Free RL?

- In most cases, it is hard for the agent to know the environment before interacting with it.
  - Example: a navigating robot explores an unknown underground cave

- Basic idea: the agent tries to take some actions based on its historical experience, observe how the environment responses, and then find an optimal policy in the long run

- RL Solutions:
  - Monte Carlo – based algorithms
  - Temporal-Difference (TD) algorithms: Q-learning, SARSA
  - Policy gradient
  - Actor-critic

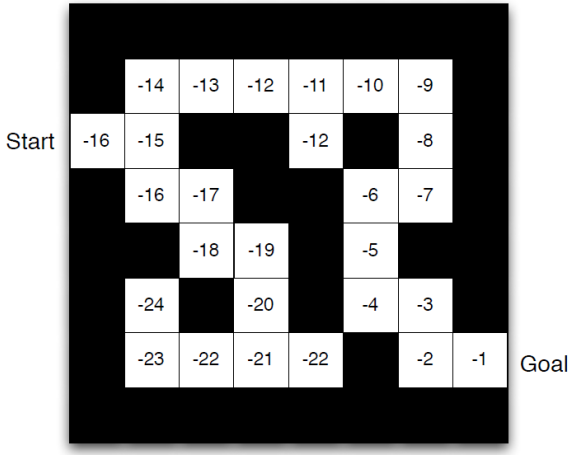
Q-learning

Algorithm 12 Q-learning Algorithm

1: **Input**: Arbitrary $Q(S, A)$ and $Q(\text{terminal state}, \cdot) = 0$
2: **Output**: (near-) optimal policy π^*
3: **Repeat**: for each episode
4: Initialize S
5: **repeat**
6: Choose action A from state S following the policy π derived by Q (e.g., ϵ-greedy policy)
7: Take action A, obtain reward R and the next state S'
8: Update Q values as

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max_a Q(S', a) - Q(S, A))$$

9: $S \leftarrow S'$
10: **until** S is the terminal state



Numbers represent value of each state

Outline

- Introduction
- Reinforcement Learning Framework & Problem
- Model-based Reinforcement Learning
- Model-free Reinforcement Learning
- Deep Reinforcement Learning
- Applications

What's Wrong with RL?

- As an academic discipline, AI is founded in 1956
- Stagnation for decades, why?

- How to solve large-scale reinforcement learning problems:
  - Backgammon: 10^20 states
  - Computer GO: 10^170 states
  - Autonomous driving: continuous state space

- Curse of dimensionality

Winter is coming …

Deep Reinforcement Learning

- A single agent which can solve human-level task:

AI = Reinforcement Learning + Deep Learning

- Examples:
  - Games: Atari, Go, …
  - User interaction: recommend, personalize, …
  - Robotics: walk, swim, …

Value Function Approximation

- So far we have represented value function by a *lookup table*
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with *function approximation*

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

Generalise from seen states to unseen states
Update parameter $\mathbf{w}$ using MC or TD learning

Which Function Approximator?

- There are many function approximators:
  - Linear: linear combinations of features
  - Non-linear: (deep) neural network => Deep Reinforcement Learning

Outline

- Introduction
- Reinforcement Learning Framework & Problem
- Model-based Reinforcement Learning
- Model-free Reinforcement Learning
- Deep Reinforcement Learning
- Applications

Workflow of a RL project

- Define the RL elements
  - States
  - Action
  - Policy
  - Reward
  - Environment Model (optional)
- Train the agent: iterate many times until good enough
- Deploy the agent

Learning to Ride a Bicycle

States: The system state variables are

$$s = (\theta, \dot{\theta}, \omega, \dot{\omega}, \ddot{\omega})^T,$$

where θ is the angle the handle bars are displaced from normal and $\dot{\theta}$ is the angular velocity of the angle. ω is the angle from vertical to bicycle, $\dot{\omega}$ is the angular velocity, and $\ddot{\omega}$ is the angular acceleration.

Actions: Overall, the agent has 6 possible actions including the selection of torque T being applied to the handle bars, and the center of mass d being displaced from the bicycle's plan,

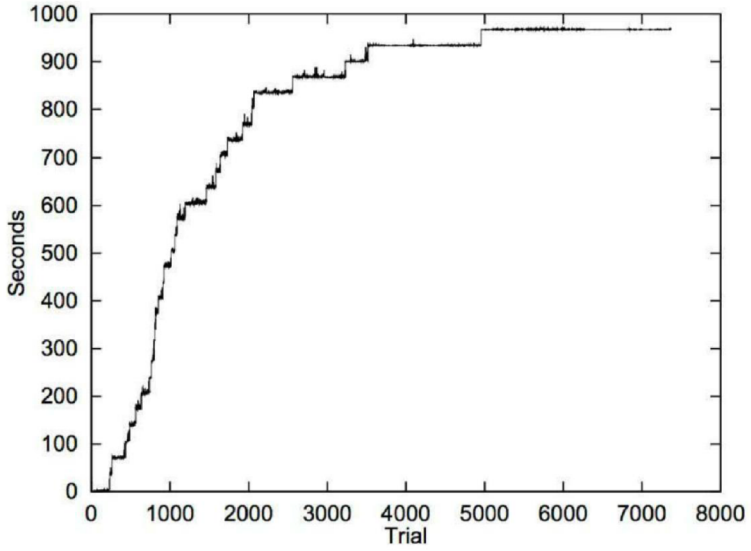
$$T \in \{-2N, 0N, +2N\}$$

$$d \in \{-2cm, 0cm, +2cm\}.$$

Reward: The agent's reward corresponds to the number of seconds that the agent keeps balancing the bicycle.

Learning to Ride a Bicycle

- We use SARSA algorithm

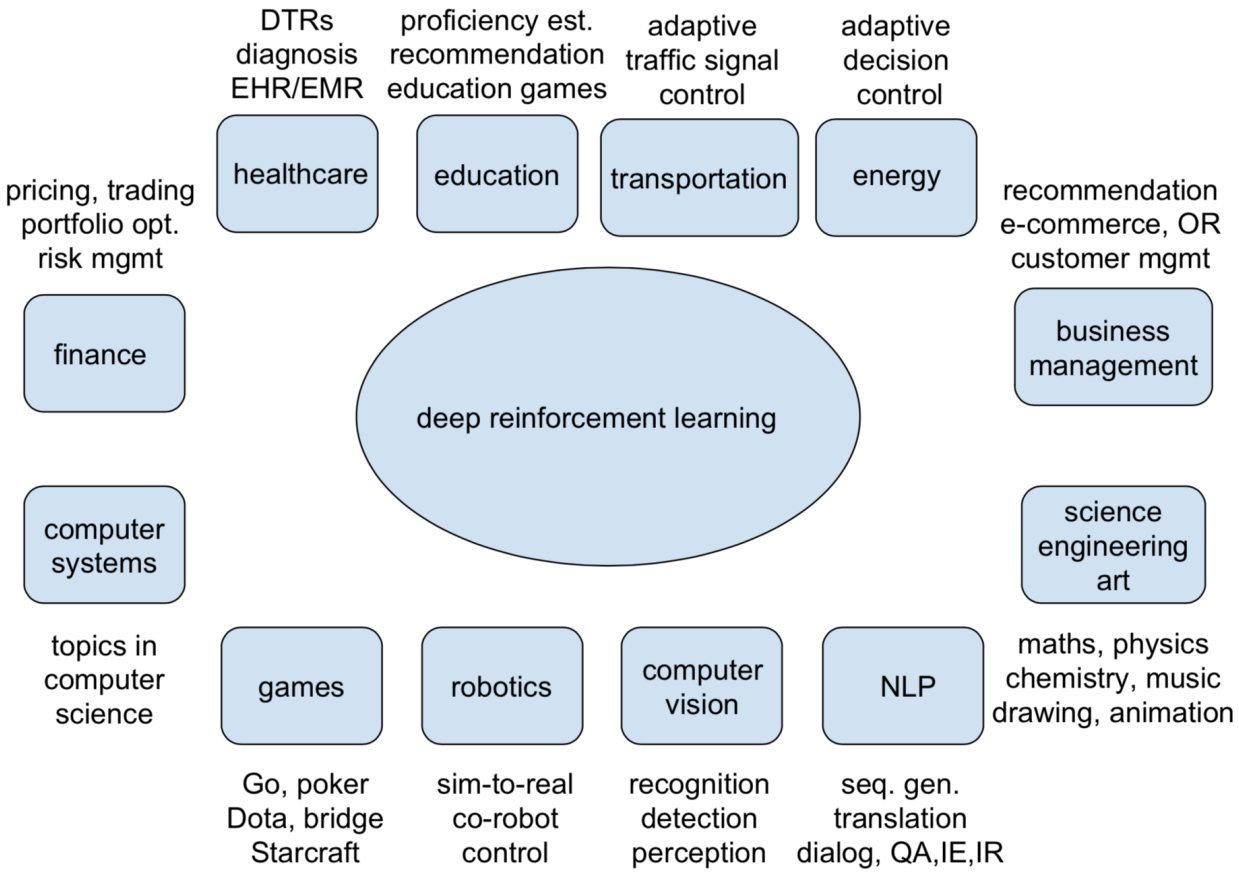


Number of seconds that the robot keeps balancing the bicycle vs. the number of trials

Record of path of first 151 trials. The longest path is 7 meters

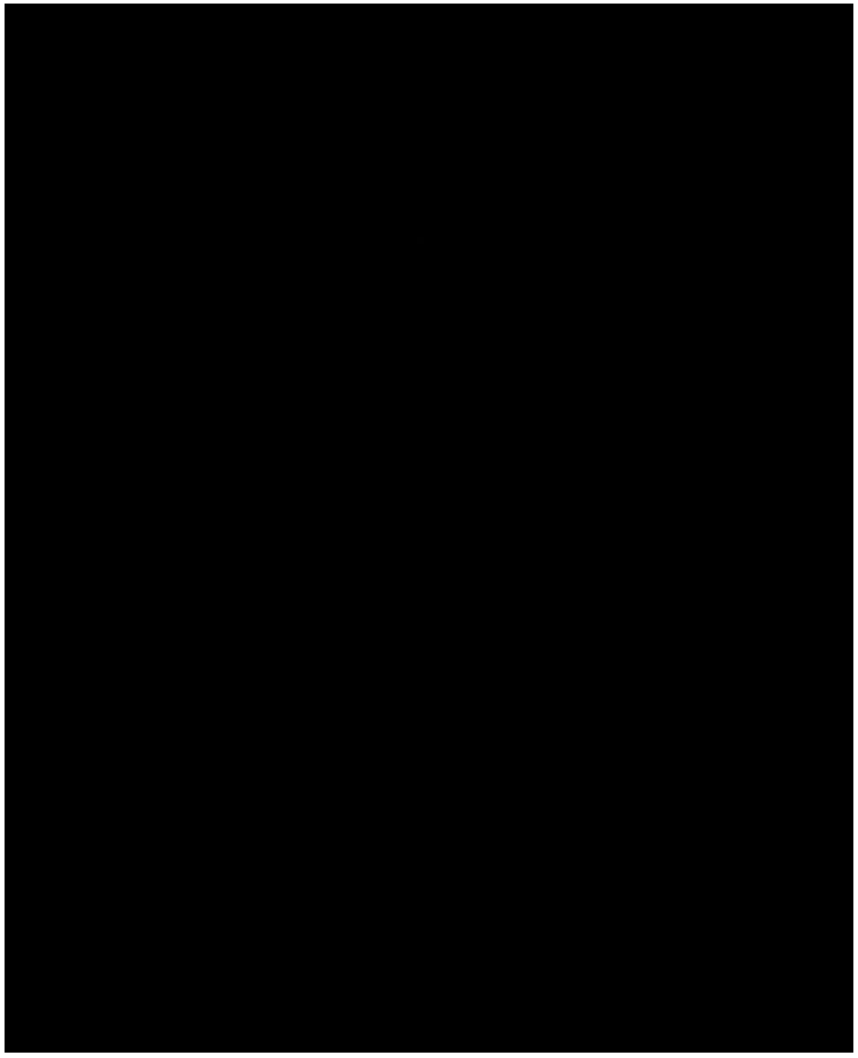
J. Randlv and P. Alstrm. Learning to derive a bicycle using reinforcement learning and shaping. ICML, 1998

Applications



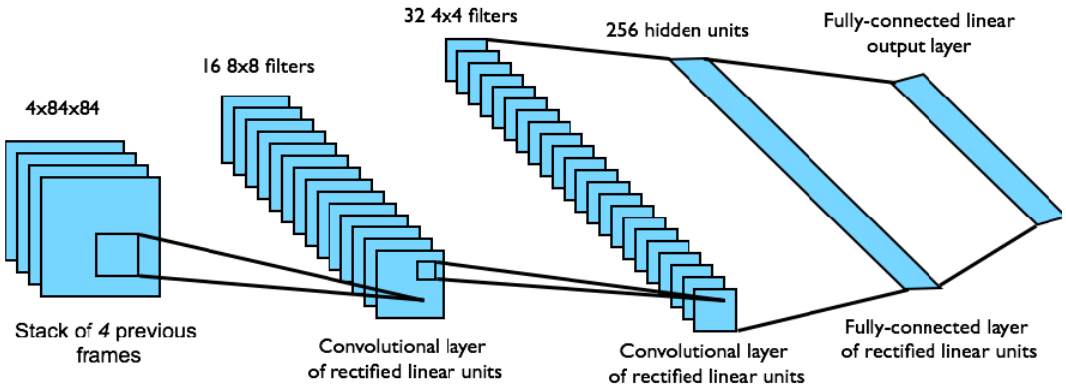
Yuxi Li, Deep Reinforcement Learning, arXiv, 2018

Google’s Deep Q-learning playing Atari Breakout



Deep Q-Networks (DQN) in Atari Games

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters do not change across 2000+ games

Human-level control through deep reinforcement learning, Nature, 2015

A Crawling Robot: Q-Learning

AI learns how to walk


NAKAMOTO
&TURING
LABS