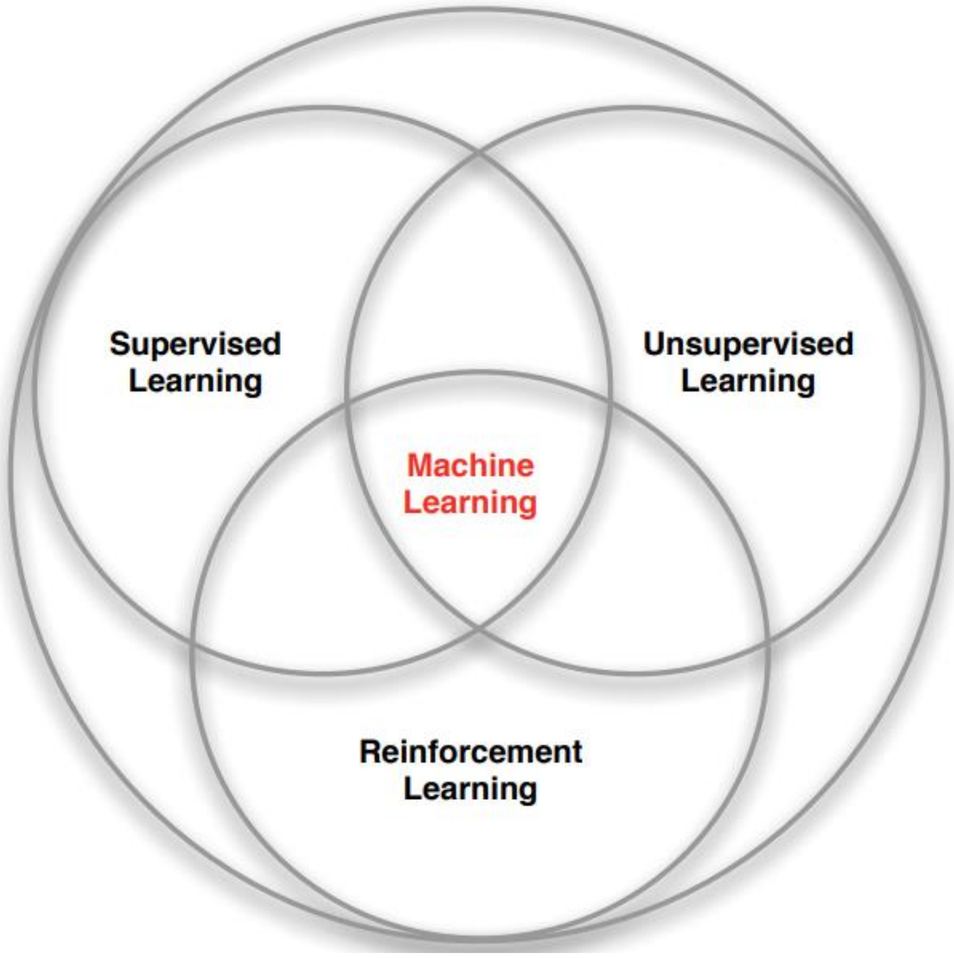
Day 6

Reinforcement Learning

Jun Zhu
dcszj@mail.tsinghua.edu.cn
Department of Computer Science and Technology
Tsinghua University

Tsinghua Deep Learning Summer School, Beijing, 2019

Branches of Machine Learning



Supervised Learning

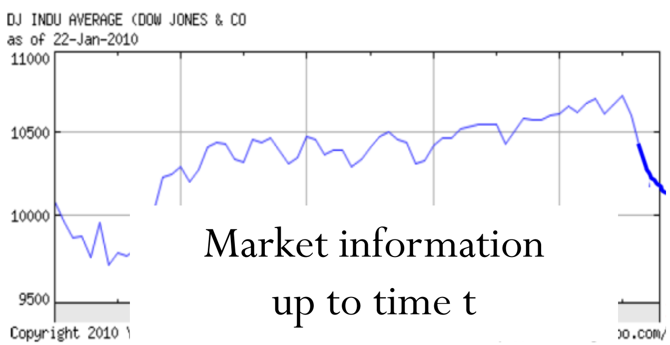
- Task: learn a predictive function $h: \mathcal{X} \rightarrow \mathcal{Y}$

Feature space $\mathcal{X}$

Label space $\mathcal{Y}$

Words in documents

“Sports”
“News”
“Politics”
…



Market information up to time t

Share price
“$ 20.50”

- “Experience” or training data:

$$\{< x_d, y_d >\}_{d=1}^{D}, \; x_d \in \mathcal{X}, y_d \in \mathcal{Y}$$

Unsupervised Learning

- Task: learn an explanatory function $f(x), \ x \in \mathcal{X}$
- Aka "Learning without a teacher"

Feature space $\mathcal{X}$

Words in documents

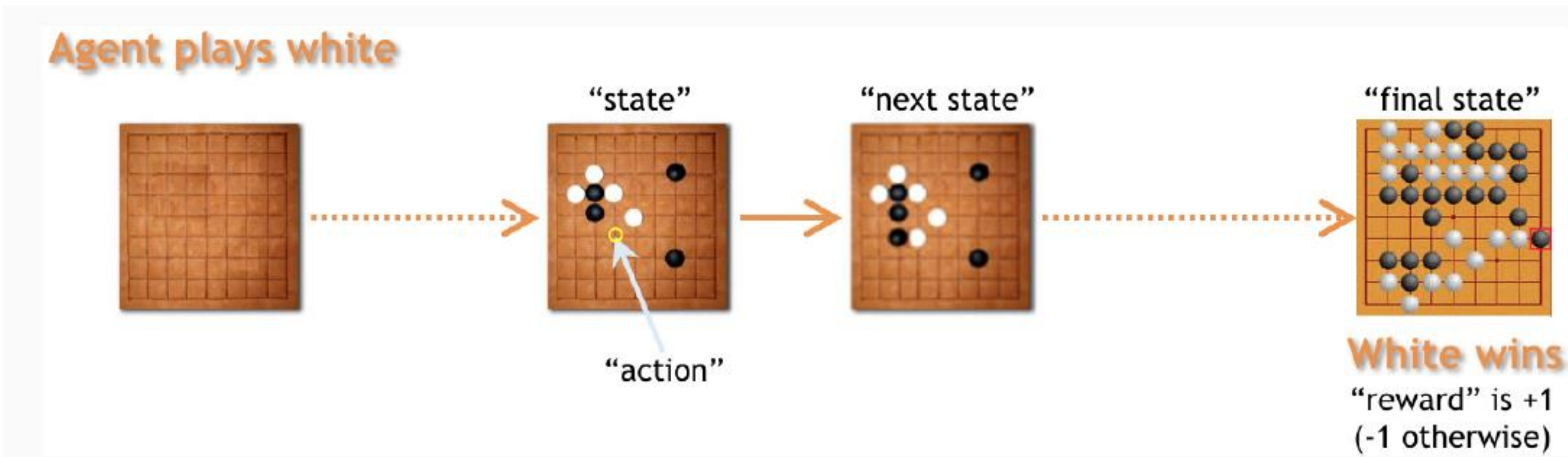
Word distribution
(probability of a word)

- No training/test split

Outline

- RL preliminaries
- RL basics
- RL advanced

Sequential decision making as RL



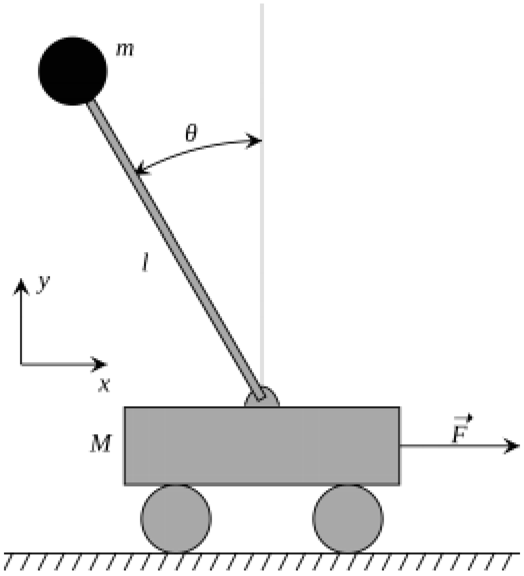
Problem Statement

Terminology

- Trajectory/episode: $s_1, a_1, r_1, \quad s_2, a_2, r_2, \quad \cdots$
- Policy: $\pi(s_t) \rightarrow a_t$
- Objective: choose a_t to maximize return $R_t := r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$

Some Examples

Cart-Pole Problem

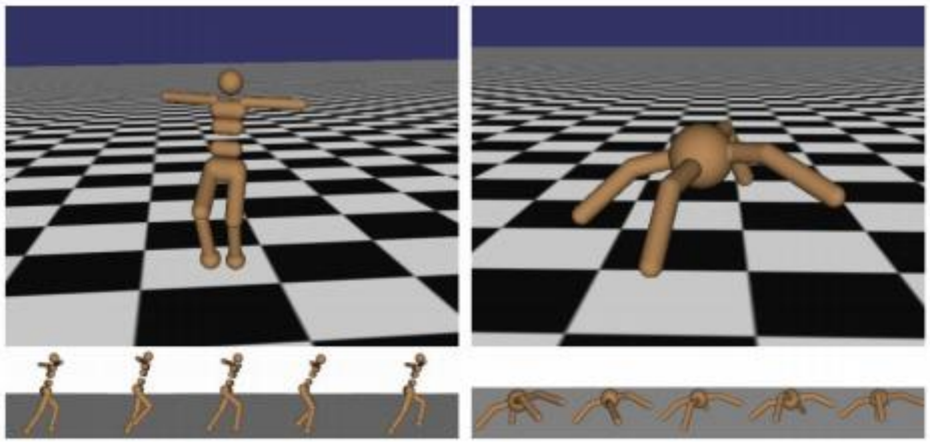


Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

Some Examples

Robot Locomotion

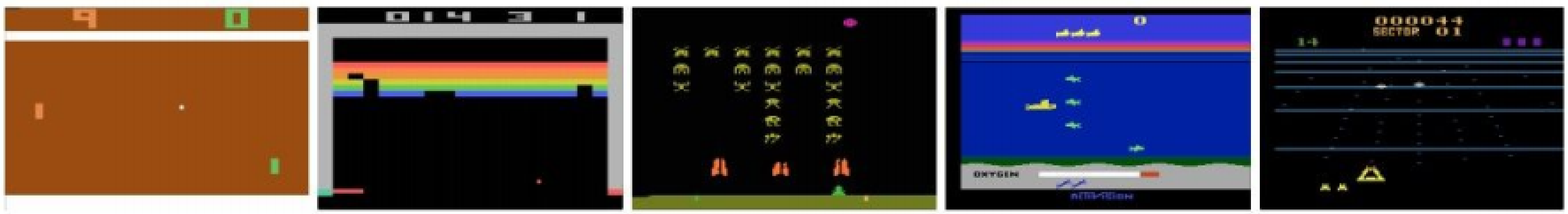


Objective: Make the robot move forward

State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

Some Examples

Atari Games

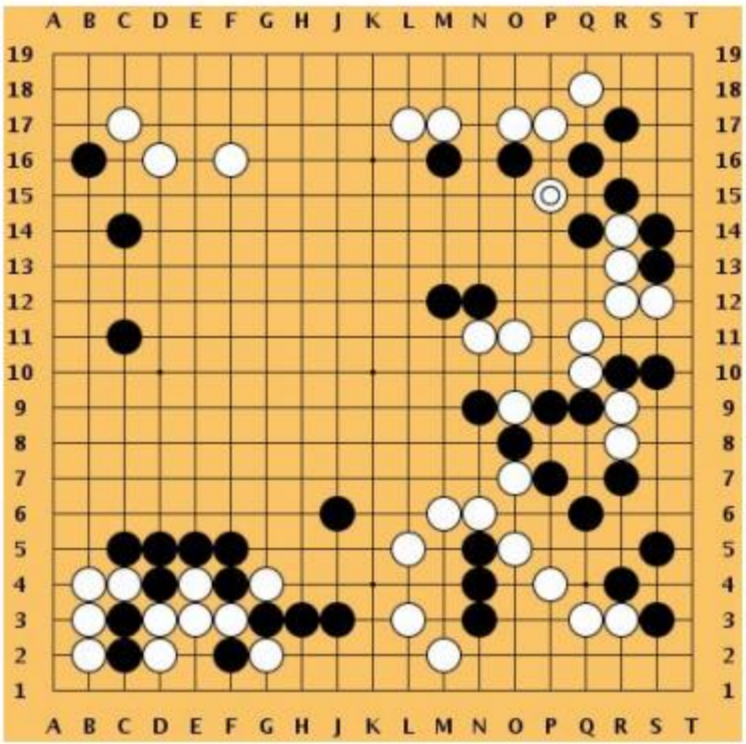


Objective: Complete the game with the highest score

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

Some Examples

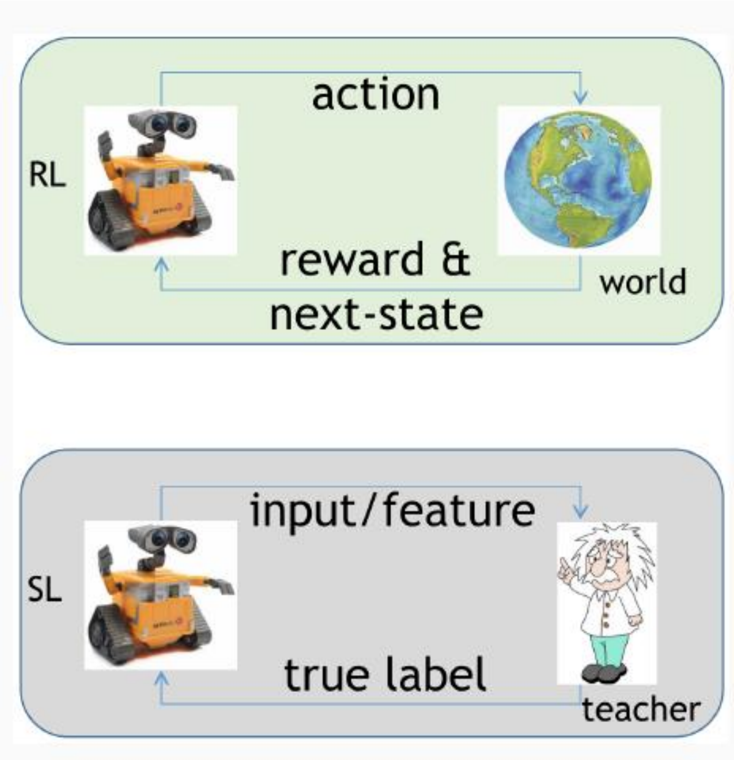
Go



Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

RL vs. SL (supervised learning)



Differences from SL

- Learn by trial-and-error ("experimenting")
  - Need exploration/exploitation trade-off
- Optimize long-term reward $(r_1 + \gamma r_2 + \cdots)$
  - Need temporal credit assignment

Similarities to SL

- Representation
- Generalization
- Hierarchical problem solving
- ...

RL or SL?

Problem: detect whether an email is spam or not.

Labeled training data

Your favorite
ML algorithm

Supervised learning

RL or SL?

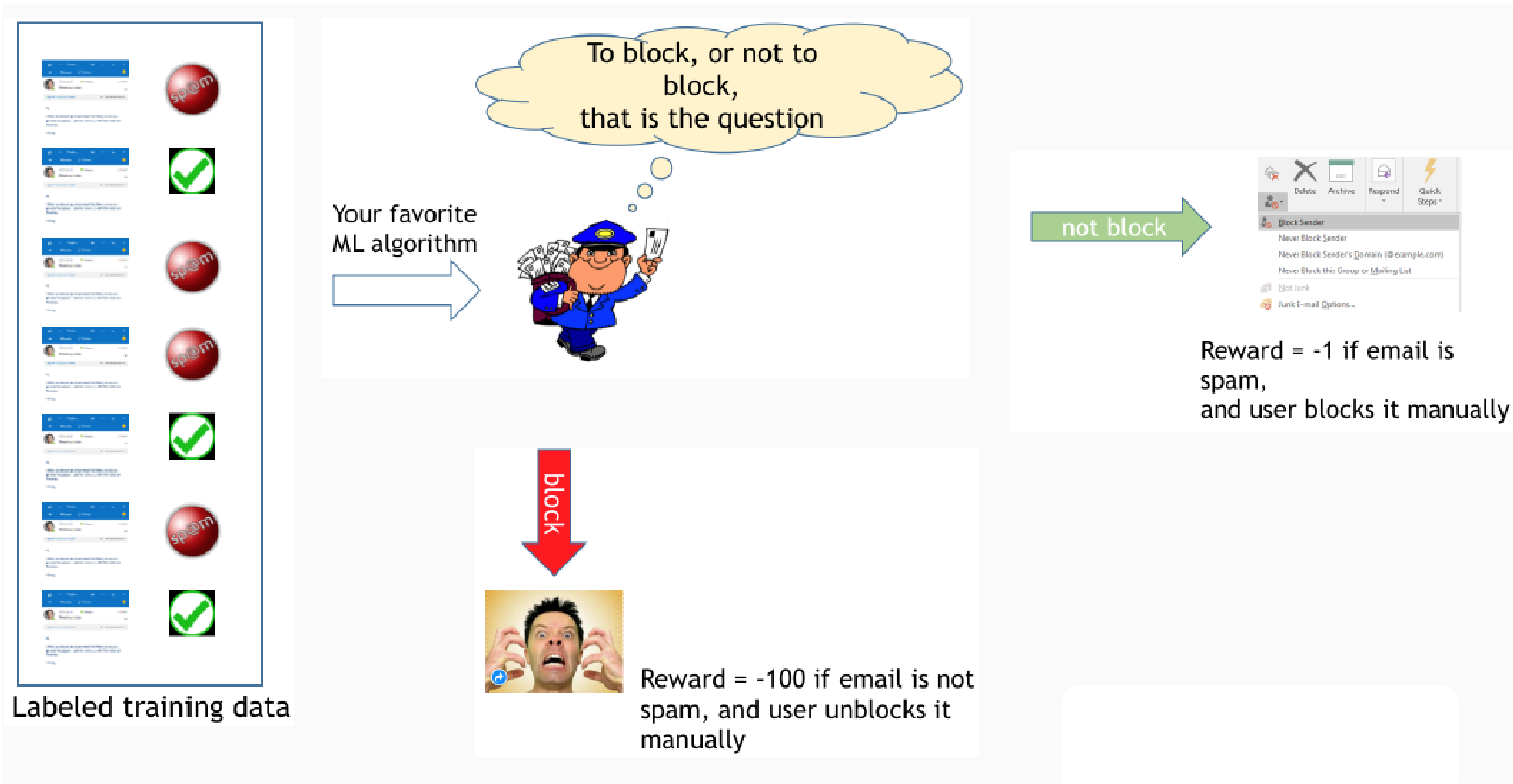
Problem: build a self-driving car

Reinforcement learning

Supervised learning

RL or SL?

- Email spam (again)



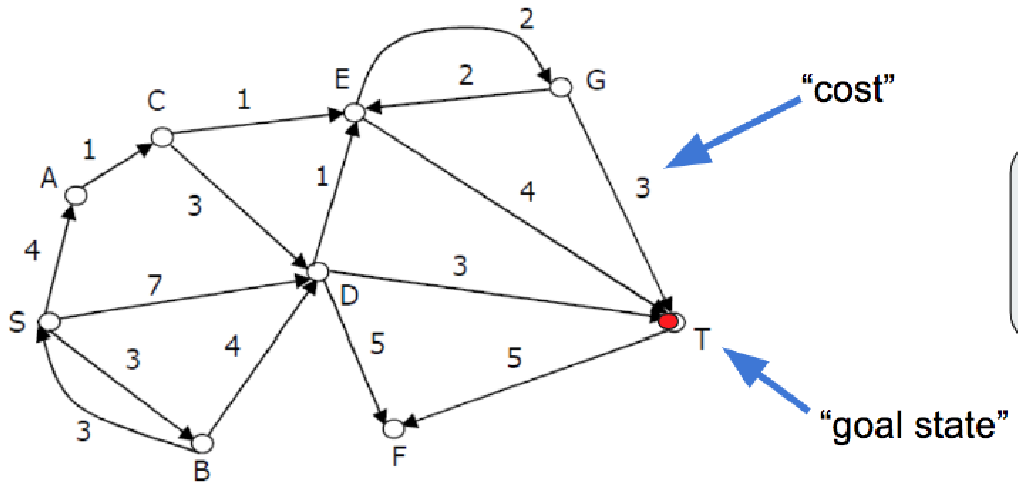
Markov Decision Process (MDP)

- Described by $M = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$:
  - $\mathcal{S}$: set of states
  - $\mathcal{A}$: set of actions
  - $P(s'|s,a)$: state-transition probabilities
  - $R(s,a) \in [0,1]$: average immediate (one-step) reward
  - $\gamma \in [0,1)$: discount factor
- Policy $\pi : \mathcal{S} \to \mathcal{A}$ (deterministic) or $\mathcal{S} \to \Delta(\mathcal{A})$ (stochastic)
  ($\Delta(\mathcal{A})$ is the set of distributions over $\mathcal{A}$)
- Goal of RL: find optimal policy π^* to maximize long-term reward:

$$\pi^* \in \arg\max_{\pi} \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | \pi\right]$$

- States are Markovian
  - States are sufficient statistics of history
  - Future is independent from history, conditioned on current state

Simple MDP: Shortest path problem



node → state
edge → action
cost → (negative) reward

$$\forall i: \quad \text{CostToGo}(i) = \min_{j \in \text{Neighbor}(i)} \{\text{cost}(i \to j) + \text{CostToGo}(j)\}$$

Principle of Optimality (Richard Bellman, 1957)

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Bellman Equations for MDPs

- General case:

Deterministic shortest path

$$\text{CostToGo}(i) = \min_{j \in \text{Neighbor}(i)} \{\text{cost}(i \to j) + \text{CostToGo}(j)\}$$

Markov decision process

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V^*(s')] \right\}$$

(maximum long-term reward starting from s)

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q^*(s',a') \right]$$

(maximum long-term reward after choosing a from s)

V^* and Q^* are called optimal value functions

Bellman Operator

- Bellman operator $\mathcal{B}$ transforms Q to another function $\mathcal{B}Q$ on $\mathcal{S} \times \mathcal{A}$:

  $$\mathcal{B}Q(s,a) := R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[\max_{a'} Q(s',a')]$$

  Special case with a fixed policy π:

  $$\mathcal{B}^{\pi}Q(s,a) := R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[Q(s',\pi(s'))]$$

  Similarly for $V(s)$
- Bellman equations re-expressed as: $Q^* = \mathcal{B}Q^*$ (or $Q^{\pi} = \mathcal{B}^{\pi}Q^{\pi}$) ($Q^*$ and Q^{π} are called "fixed points" of $\mathcal{B}$ and $\mathcal{B}^{\pi}$)
- Some key properties of $\mathcal{B}$:
  - Q^* leads to an optimal policy

    $$\pi^*(s) := \max_a Q^*(s,a)$$

  - More generally, near-optimal Q leads to a near-optimal policy [4]
  - Monotonicity: $Q_1 \geq Q_2$ implies $\mathcal{B}Q_1 \geq \mathcal{B}Q_2$
  - It is a contraction
  - Its fixed point exists and is unique

Value Iteration (VI)

Algorithm

- Initialize Q_0 arbitrarily (e.g., $Q_0 \equiv 0$)
- For $k = 1, 2, \ldots$ (until termination condition)
  - $Q_k \leftarrow \mathcal{B} Q_{k-1}$

- Convergence:
  - So V.I. converges to Q^* exponentially fast
  - Can be used to decide termination condition for VI

Policy Iteration (PI)

Algorithm

- Start with arbitrary policy $\pi_0 : \mathcal{S} \to \mathcal{A}$
- For $k = 0, 1, 2, \ldots$
  - Policy evaluation: solve for Q_k that satisfies

$$\forall (s, a) : Q_k(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) Q_k(s', \pi_k(s'))$$

(Just solving Bellman equation $\mathcal{B}^{\pi_k} Q = Q$)
(Just a system of linear equations)

  - Policy improvement:

$$\pi_{k+1}(s) \leftarrow \arg\max_a Q_k(s, a)$$

(π_{k+1} is called a greedy policy w.r.t. Q_k)

Policy Improvement Theorem

Theorem

In PI, either π_{k+1} is strictly better than π_k, or π_k is optimal.

VI vs. PI

- VI works in value function space (asymptotic convergence)

$$Q_0 \to Q_1 \to Q_2 \to \cdots \to Q^*$$

- PI does hill-climbing in policy space (finite-time convergence)

$$\pi_0 \xrightarrow{Q_0} \pi_1 \xrightarrow{Q_1} \pi_2 \cdots \xrightarrow{Q_{K-1}} \pi_K = \pi^*$$

RL Basics

RL in MDP

Typical life of an RL agent

- Observe initial state s_1
- For $t = 1, 2, 3, \ldots$
  - Choose action a_t based on s_t and current policy
  - Observe reward r_t and next state s_{t+1}
  - Update policy using new information (s_t, a_t, r_t, s_{t+1})
- Episode length may be finite or infinite
- Agent can have multiple episodes starting from new initial states

What we have learned so far

- Given MDP model ($R(s, a)$ and $P(s'|s, a)$), we can compute an optimal policy, using value iteration, policy iteration, etc.
- What if R and P are unknown?
  - This is what reinforcement learning is about!

Q-Learning

- Value iteration assumes knowledge of R and P:

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

- We may use data to approximate unknown quantities
- Given $\mathcal{D} = (s_1, a_1, r_1, \quad s_2, a_2, r_2, \quad \ldots)$, we want Q so that

$$\forall t : Q(s_t, a_t) \approx \underbrace{r_t + \gamma \max_{a'} Q(s_{t+1}, a')}_{\text{“bootstrapping”: use own estimate to create learning target}}$$

- This inspires Q-learning update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \underbrace{\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)}_{\text{“temporal difference” or “TD error”}}$$

where $\alpha_t \in (0,1)$ is a stepsize.

- Botstrapping: enables learning **without a teacher** (unlike SL)

Convergence of Q-Learning

Observations

- Q-learning updates are local: only $Q(s_t, a_t)$ changes at step t
- Bootstrapped value is a locally unbiased estimate of $\mathcal{B}$

$$\mathbb{E}_{r_t, s_{t+1} | s_t, a_t}\left[r_t + \gamma \max_{a'} Q(s_{t+1}, a')\right] = \mathcal{B}Q(s_t, a_t)$$

Theorem (from Stochastic Approximation theory [5])

Q-learning running on trajectory $\mathcal{D} = (s_1, a_1, r_1, s_2, a_2, r_2, s_3, \ldots)$ converges to Q^* almost surely, if the following holds for all (s, a):

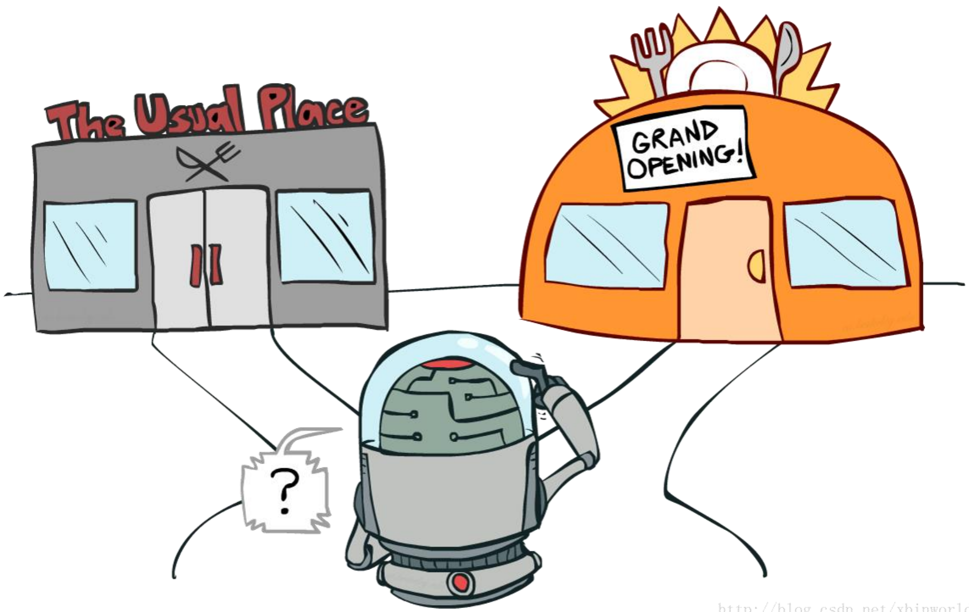
$$\sum_{t=1}^{\infty} \alpha_t \cdot \mathbb{I}\{s_t = s, a_t = a\} = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 \cdot \mathbb{I}\{s_t = s, a_t = a\} < \infty\,.$$

The condition implies that every (s, a) must occur infinitely often in $\mathcal{D}$.

Need for sufficient exploration

- Choose the best restaurant
  - 10 restaurants in total
  - You have tried 3 restaurants, and the best one is 清芬园, with score 8
  - What's your choice next time in order to have best dishes?



Q-learning with Ɛ–greedy exploration

Input: $\epsilon \in (0,1)$, $\{\alpha_t\}$
Initialize $Q(s,a) \leftarrow 0$ for all (s,a)
Observe initial state s_1
for $t = 1, 2, 3, \ldots$ **do**
    Choose action with ϵ-greedy exploration:

$$a_t \leftarrow \begin{cases} \arg\max_a Q(s_t, a), & \text{with prob. } 1-\epsilon \\ \text{random action}, & \text{with prob. } \epsilon \end{cases}$$

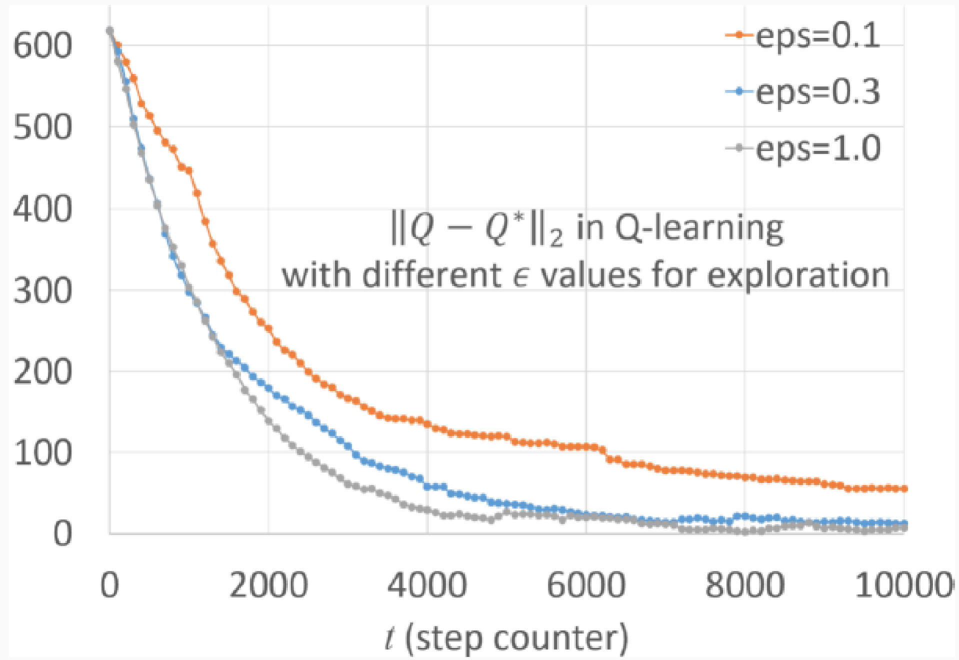
    Observe reward r_t and next-state s_{t+1}
    Update Q-function

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right)$$

end for

Some Examples

- Slowly decaying learning rate from 0.05 to 0.0001
- Different ϵ values
- Convergence is always guaranteed, but at different speed



Main lesson

Need efficient exploration/exploitation trade-off!

Exploration: ε–greedy

Selection rule

At every step, select a random action with small probability, and the current best-guess otherwise:

$$a_t = \begin{cases} \arg\max_a Q(s_t, a), & \text{with prob. } 1-\epsilon \\ \text{random action}, & \text{with prob. } \epsilon \end{cases}$$

- Larger ϵ, more exploration
    - Uniform random exploration when $\epsilon = 1$
- Very simple to implement; can be quite effective in practice
    - Often use a sequence of decaying ϵ values, reducing amount of exploration over time.
- Related strategies: ϵ-first, softmax/Boltzmann exploration
- Theoretically flawed; can get stuck in certain cases

Exploration Bonus

Selection rule

$$a_t = \arg\max_a \left\{ Q_(s_t, a) + B(s_t, a) \right\}$$

for some pre-defined exploration bonus function $B(s_t, a)$.

- $B(s, a)$ measures the need for exploration, e.g., some *decreasing* function of $C(s, a)$ — number of times a has been chosen in s
- Popular choice: $B(s, a) = \alpha / \sqrt{C(s, a)}$ with parameter $\alpha > 0$.
  - Also known as Upper Confidence Bound (UCB)
- An example of guided exploration (as opposed to ϵ-greedy)

Other Exploration Strategies

- Unguided exploration: ϵ-greedy, ϵ-first, forced exploration, ...
- Boltzmann/softmax exploration:
- Exponentiated exploration rules [1]
- Thompson sampling [6, 26]
- Exploration bonus: UCB, pseudocounts, curiosity driven, ...
- ...

RL Advanced

Approximation and Generalization

So far we have focused on finite state/action MDPs where all calculations (such as Bellman operator) can be done exactly.

- Computationally, how to handle large MDPs?
  - Number of legal states in Go is 2×10^{170}
  - In some problems (e.g., autonomous driving), state/actions are even continuous

No relation is assumed between two distinct states $s', s'' \in \mathcal{S}$.

- Statistically, how to generalize information to unseen states to reduce sample complexity?
  - No one is able to exhaust all states before mastering Go

That is why we need good representation in RL

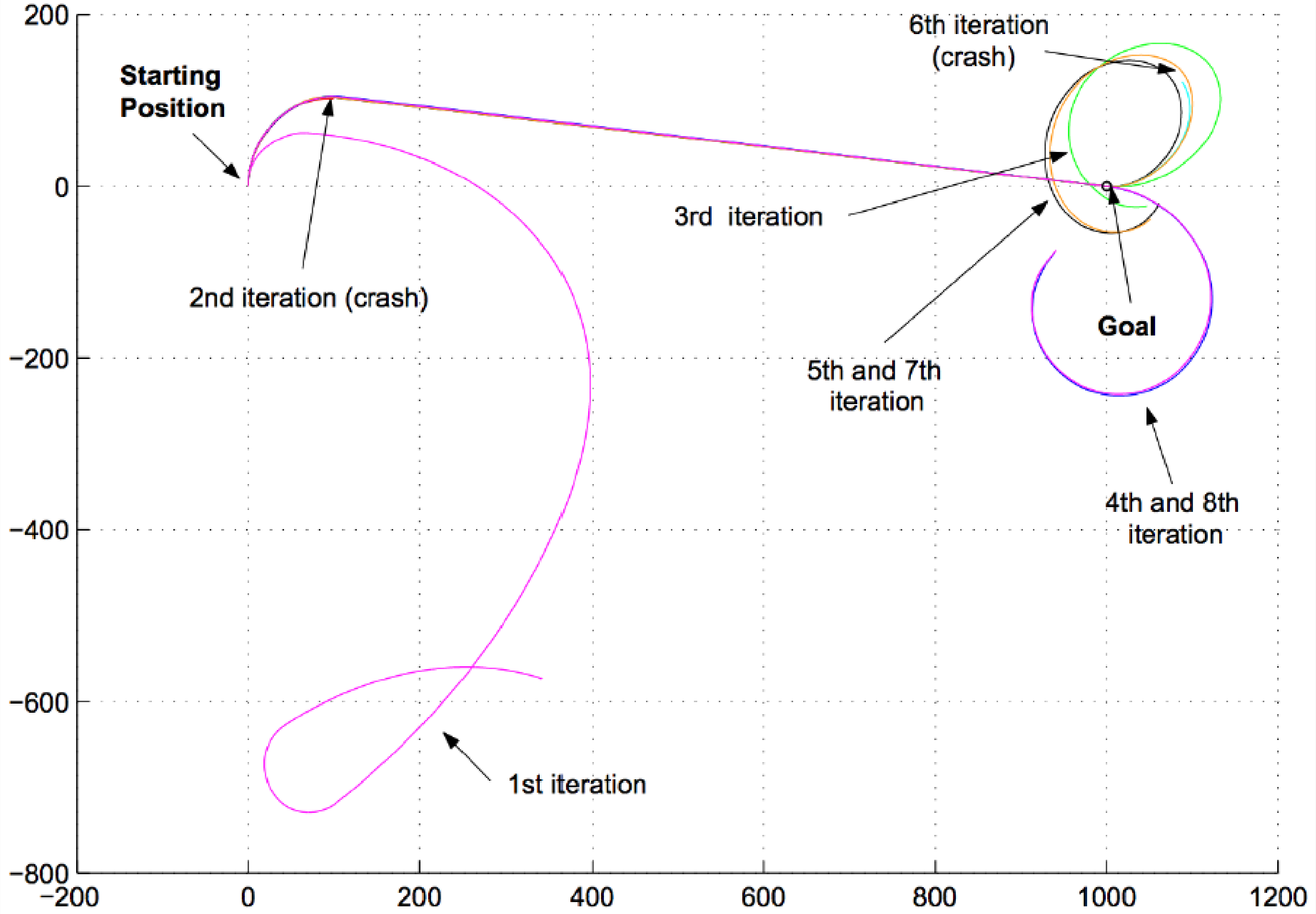
Function Approximation (FA) in RL

- Tabular/exact representations (covered so far); do not scale well
  - Q, V and π represented as $|\mathcal{S}|$- or $|\mathcal{S} \times \mathcal{A}|$-dimensional vectors
- Aggregation/abstraction/discretization: treating multiple states/actions as the same [20]
- Linearly parametric approximation with given features (a.k.a. basis functions):

$$Q(s, a; \theta) = \theta^\top \phi(s, a), \qquad \text{where } \theta, \phi(s, a) \in \Re^d$$

- Nonlinear parametric approximation, e.g., neural nets
- Non-parametric representations: kernel, decision trees, ...

Learning to Ride Bicycle



Results of Least-Squares Policy Iteration (linear FA) [16]

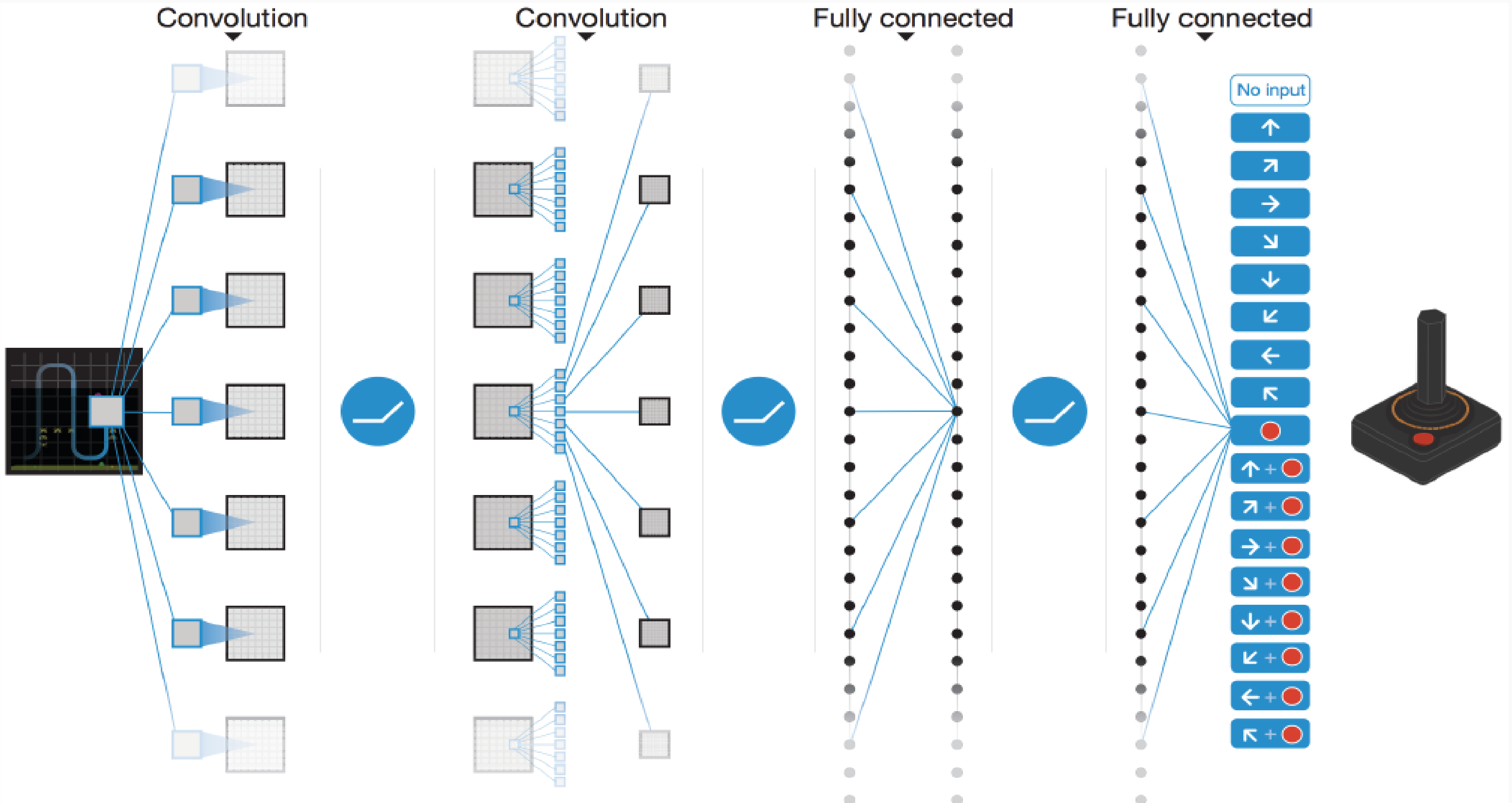
Learning to Ride Bicycle: States & Features

- State $s = (\theta, \dot{\theta}, \omega, \dot{\omega}, \ddot{\omega}, \psi) \in \Re^6$
  - θ: angle of handlebar
  - ω: vertical angle of the bicycle
  - ψ: angle of the bicycle to the goal
- Action $a = (\tau, \nu) \in \Re^2$
  - $\tau \in \{0, \pm 2\}$: torque applied to the handlebar
  - $\nu \in \{0, \pm 0.02\}$: displacement of the rider
- Popular featurization techniques: polynomial basis, radial basis, Fourier basis, coarse coding, ...
- Previous slide's results are based on degree-2 polynomial features:

$$Q(s, a; \theta) = \theta_a^\top \phi(s), \quad \text{where } \theta = (\theta_1, \ldots, \theta_{|\mathcal{A}|}) \in \Re^{|\phi| \times |\mathcal{A}|}$$

- Challenge: how to construct features automatically [24]
- Use neural networks to learn representations ("deep RL")

Nonlinear FA with Neural Nets



The DQN architecture that enables an RL agent to excel at a wide range of video games [23]. The use of neural networks in RL dates back to the 80s, with a great success of TD-Gammon that beated human champion in the game of back-gammon [36].

Q-Learning with Function Approximation

- Recall tabular Q-learning update on transition (s_t, a_t, r_t, s_{t+1})

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \underbrace{\left(\overbrace{r_t + \gamma \max_a Q(s_{t+1}, a)}^{\text{regression target}} - Q(s_t, a_t) \right)}_{\text{TD error, denoted by } \delta_t}$$

- Q-learning with differentialbe parametric representation $Q(s, a; \theta)$

$$\theta \leftarrow \theta + \alpha_t \cdot \delta_t \cdot \nabla_\theta Q(s_t, a_t; \theta)$$

- Linear FA special case with $Q(s, a; \theta) = \theta^\top \phi(s, a)$

$$\theta \leftarrow \theta + \alpha_t \cdot \delta_t \cdot \phi(s_t, a_t)$$

Q-learning with FA Pseudo-code

Input: θ_0, $\epsilon \in (0,1)$, $\{\alpha_t\}$
Initialize $\theta \leftarrow \theta_0 \in \Re^d$
Observe initial state s_1
for $t = 1, 2, 3, \ldots$ **do**
    Choose action with ϵ-greedy exploration:

$$a_t \leftarrow \begin{cases} \text{arg max}_a \, Q(s_t, a; \theta), & \text{with prob. } 1-\epsilon \\ \text{random action}, & \text{with prob. } \epsilon \end{cases}$$

    Observe reward r_t and next-state s_{t+1}
    Update Q-function

$$\theta \leftarrow \theta + \alpha_t \left(r_t + \gamma \max_a Q(s_{t+1}, a; \theta) - Q(s_t, a_t; \theta) \right) \nabla_\theta Q(s_t, a_t; \theta)$$

end for

Heuristics that Sometimes Work

Q-learning with FA can easily be unstable in practice.

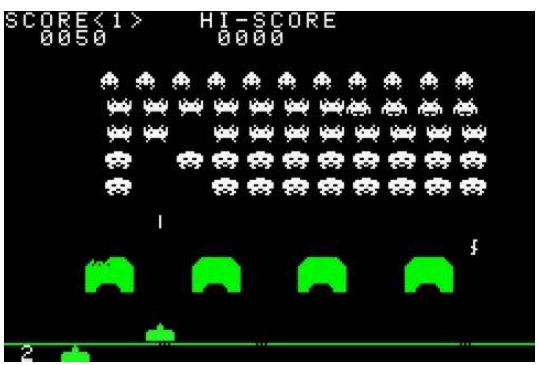
Some potentially helpful heuristics:

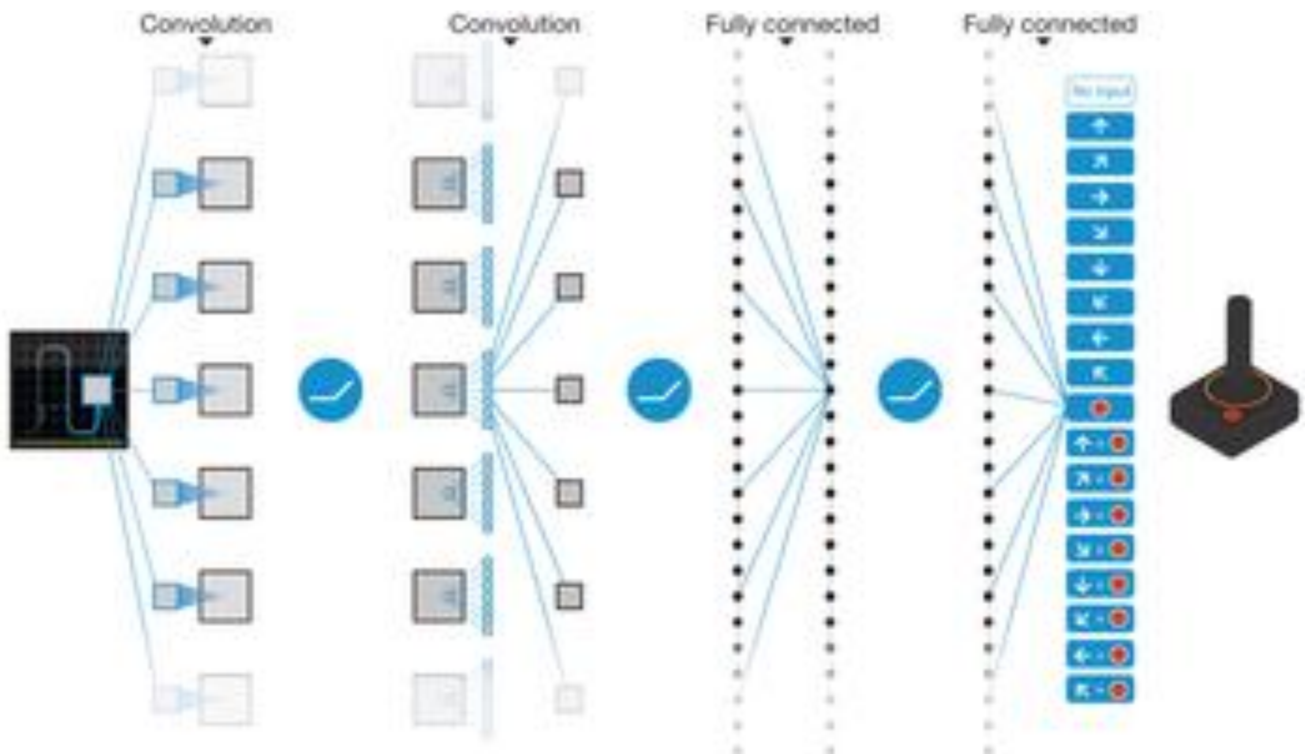
- Richer/safer representation [30]
- Two-network implementation [23]
- Experience replay [21, 23]
- Use an on-policy alternative such as Sarsa [30]
- ...

Are there provably stable algorithmic alternatives?

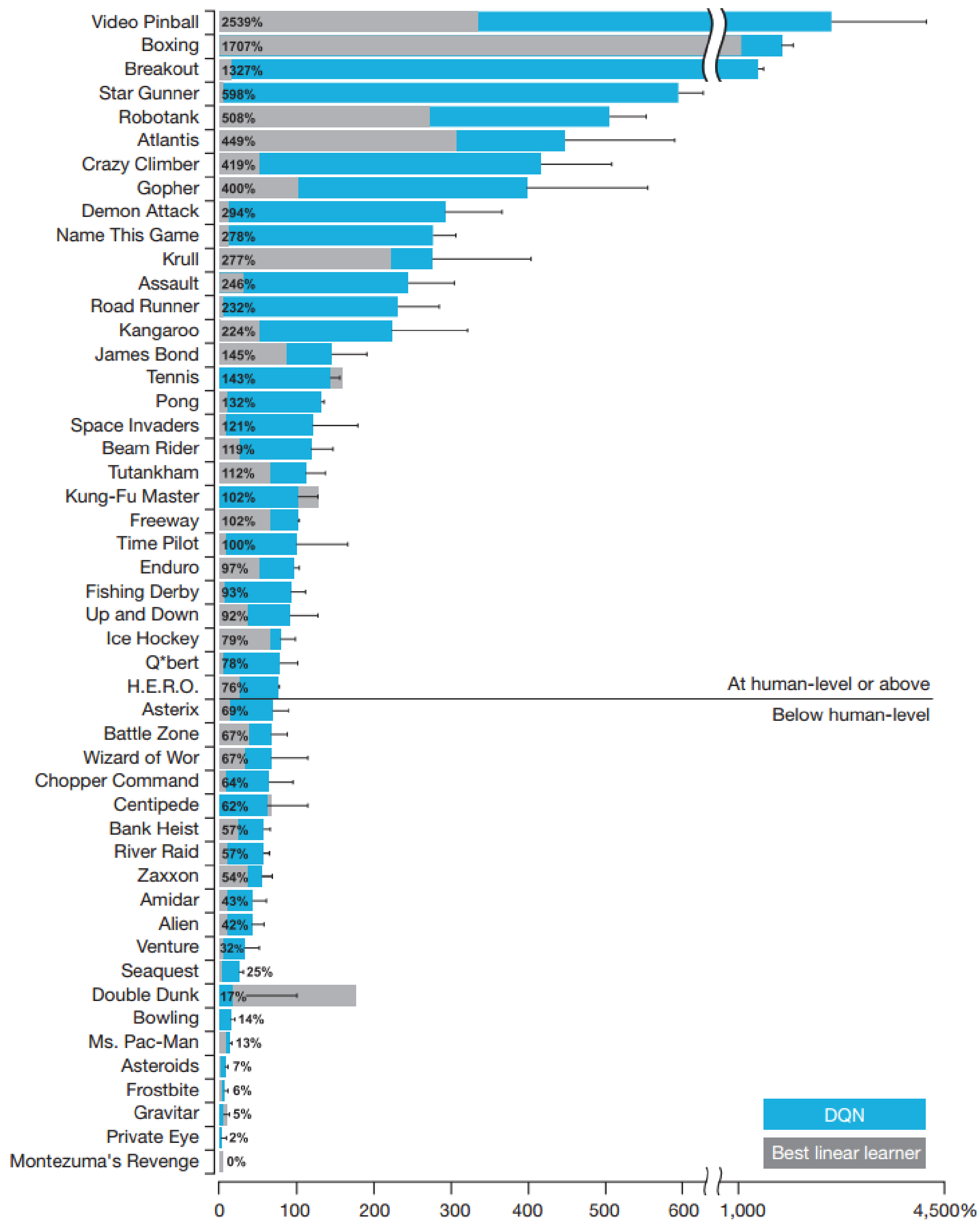
Human-Level Control via Deep RL

- Deep Q-network with human-level performance on Atari games





[Mnih et al., Nature 518, 529–533, 2015]



Video Pinball
Boxing
Breakout
Star Gunner
Robotank
Atlantis
Crazy Climber
Gopher
Demon Attack
Name This Game
Krull
Assault
Road Runner
Kangaroo
James Bond
Tennis
Pong
Space Invaders
Beam Rider
Tutankham
Kung-Fu Master
Freeway
Time Pilot
Enduro
Fishing Derby
Up and Down
Ice Hockey
Q*bert
H.E.R.O.
Asterix
Battle Zone
Wizard of Wor
Chopper Command
Centipede
Bank Heist
River Raid
Zaxxon
Amidar
Alien
Venture
Seaquest
Double Dunk
Bowling
Ms. Pac-Man
Asteroids
Frostbite
Gravitar
Private Eye
Montezuma's Revenge
2539%
1707%
1327%
598%
508%
449%
419%
400%
294%
278%
277%
246%
232%
224%
145%
143%
132%
121%
119%
112%
102%
102%
100%
97%
93%
92%
79%
78%
76%
69%
67%
67%
64%
62%
57%
57%
54%
43%
42%
32%
25%
17%
14%
13%
7%
6%
5%
2%
0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

Demo

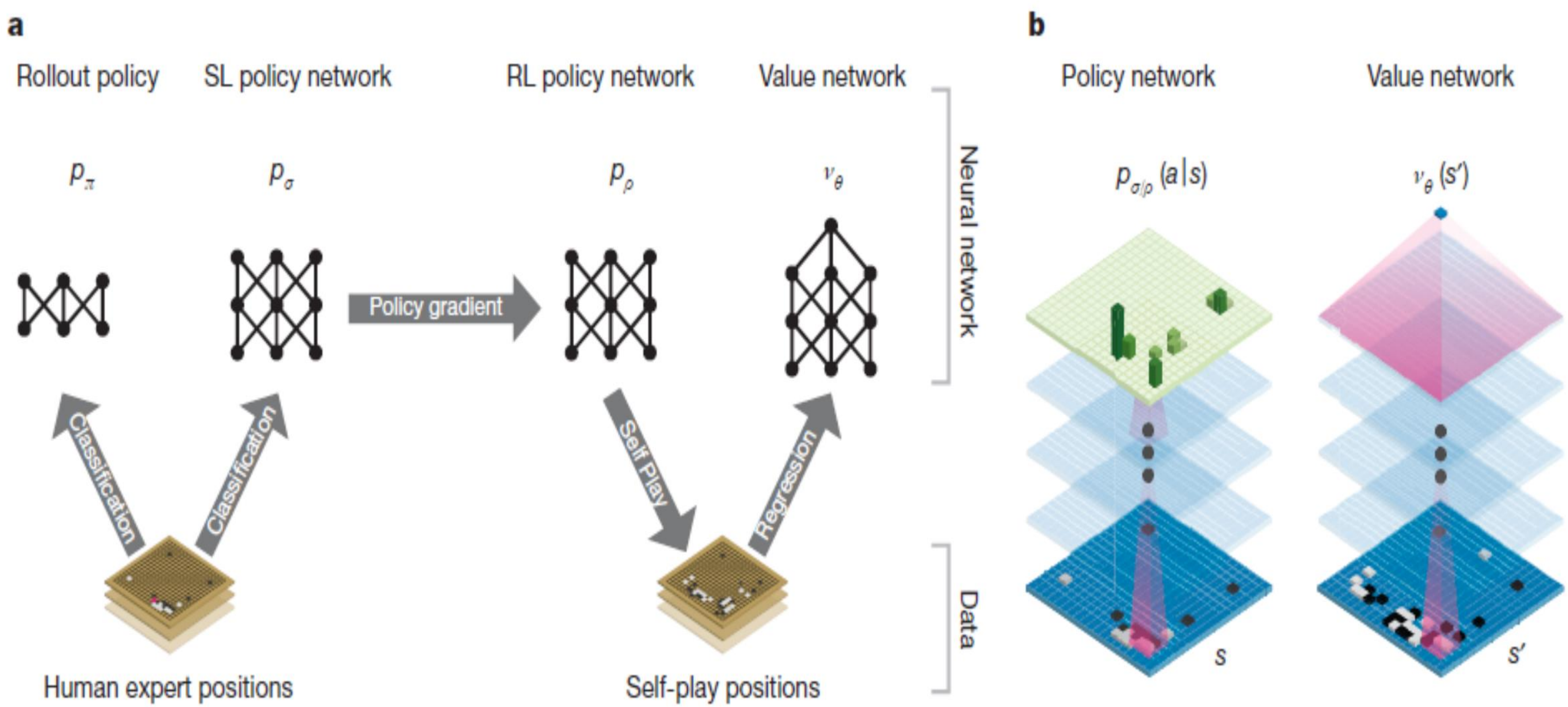
- Google DeepMind's DQN playing Atari Breakout

Demo

- Google DeepMind's DQN playing Atari Pacman

AlphaGo

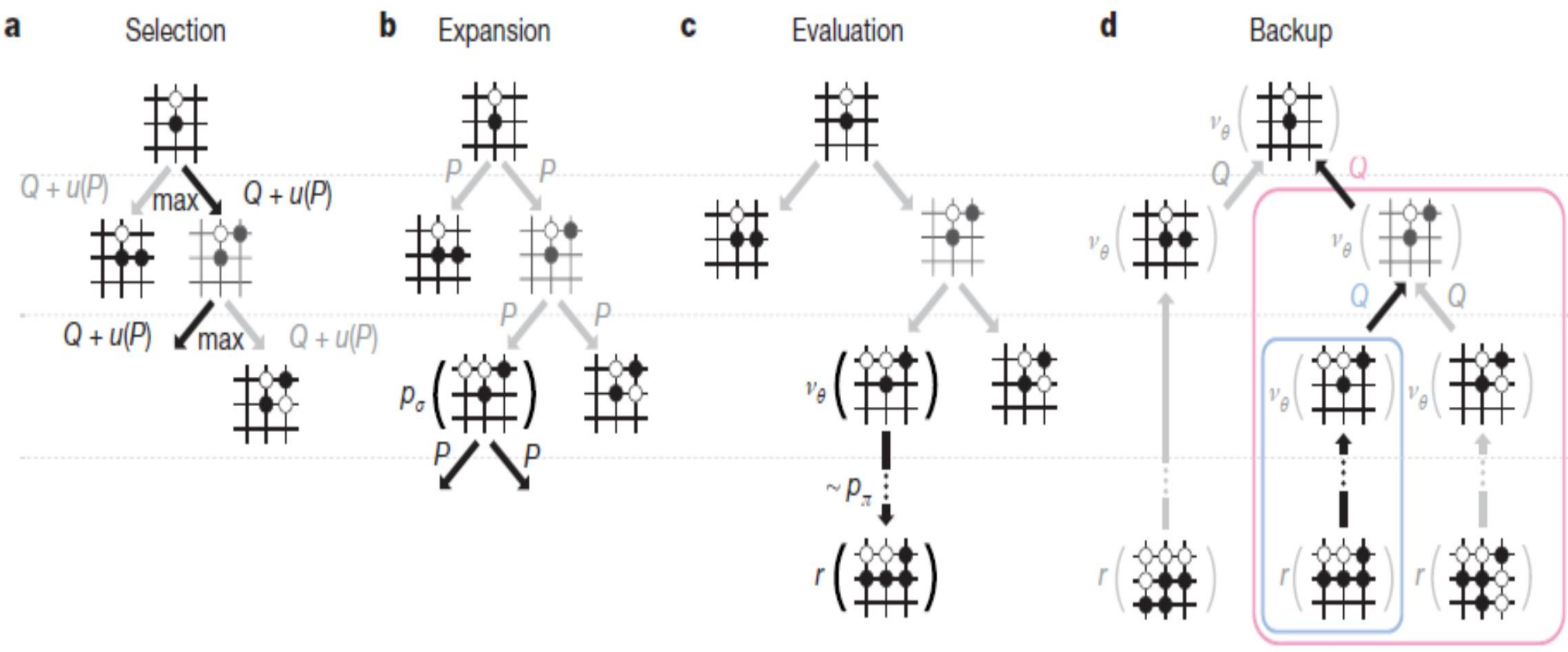
- Neural network training pipeline and architecture



[Silver et al., Mastering the game of Go with deep neural networks and tree search. Nature, 484(529), 2016]

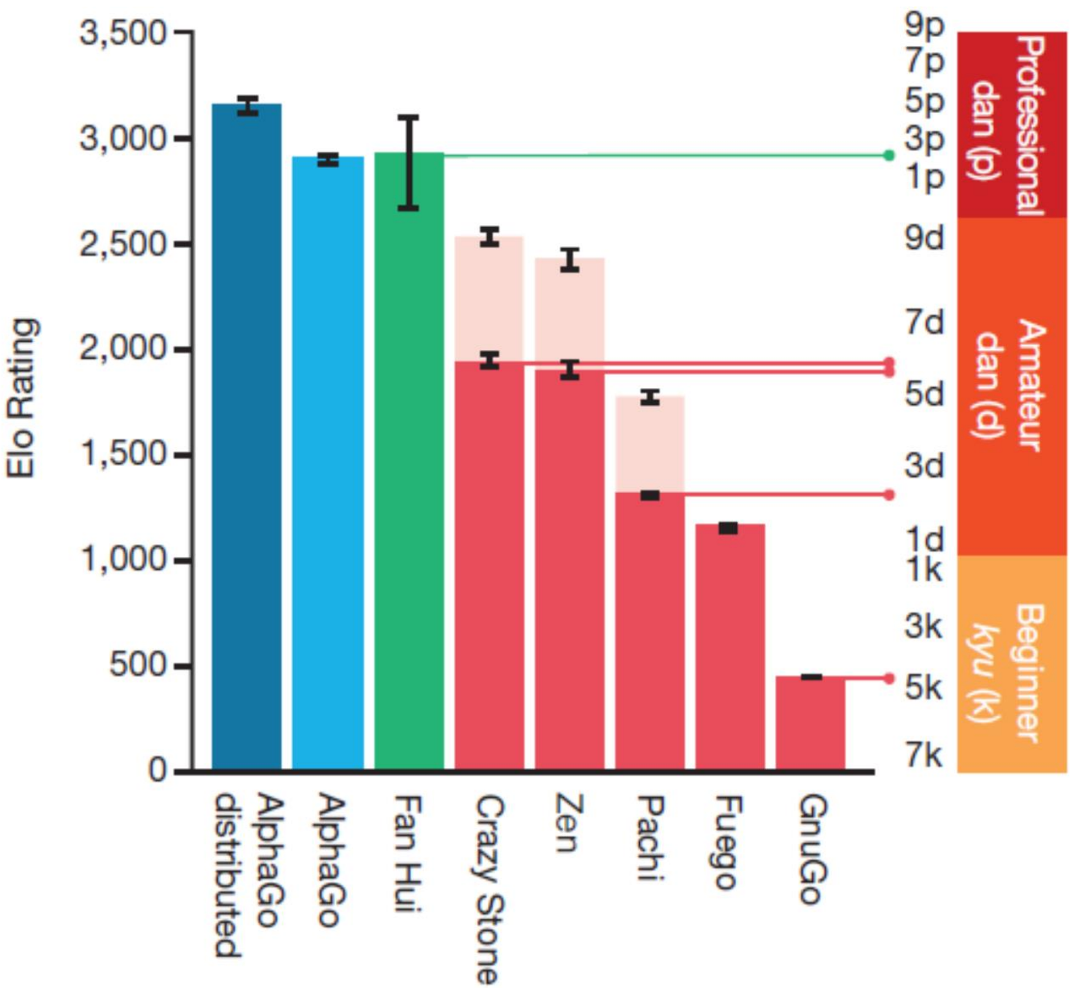
AlphaGo

- Monte Carlo tree search

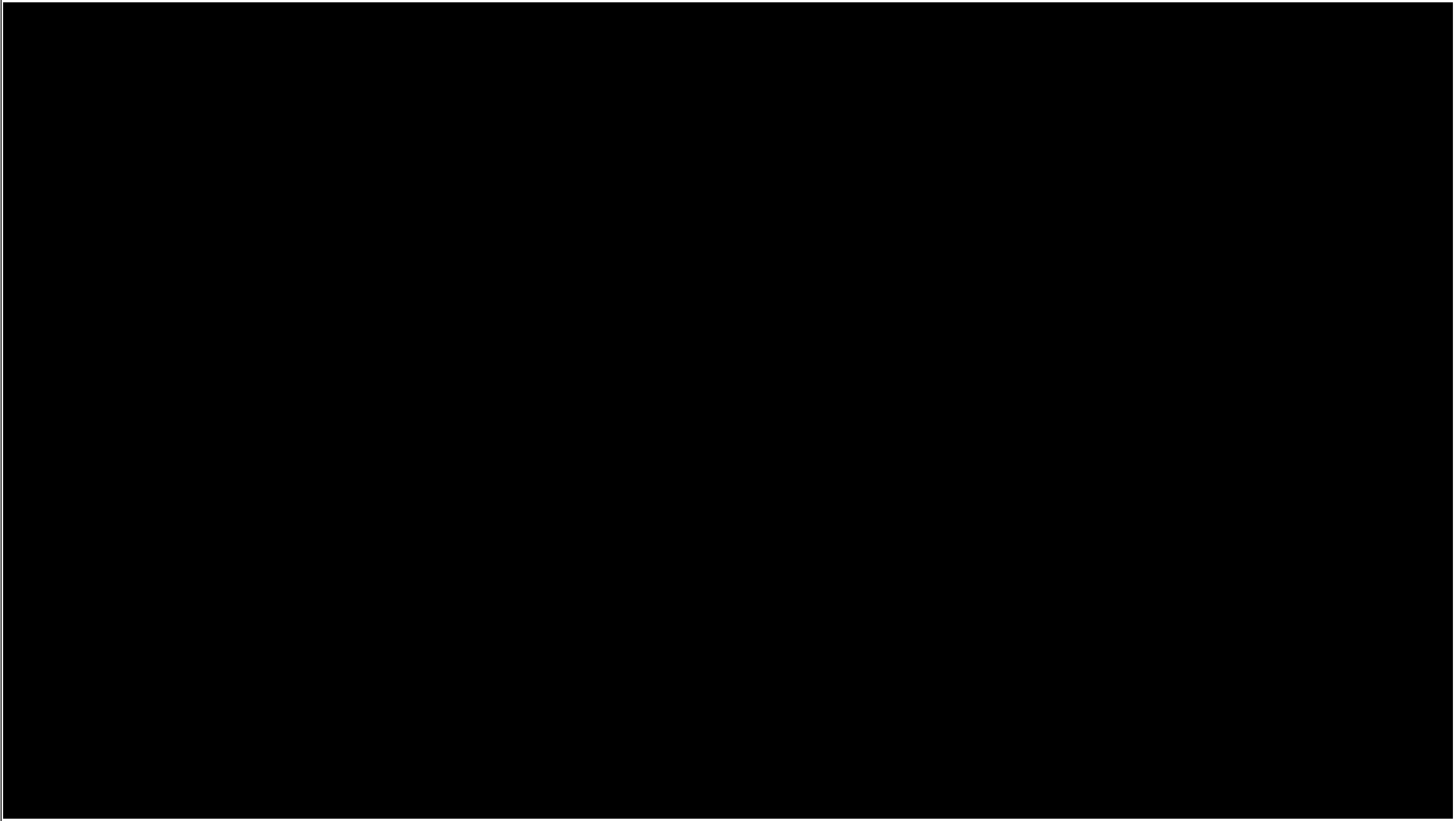


[Silver et al., Mastering the game of Go with deep neural networks and tree search. Nature, 484(529), 2016]

AlphaGo



[Silver et al., Mastering the game of Go with deep neural networks and tree search. Nature, 484(529), 2016]



RL Algorithms: A Simplified Overview

	Model (P & R) is known (aka "planning")	Model is unknown (full RL)
Value-function methods	value iteration policy iteration linear program	approx. VI (TD, Q-learning) approx. PI (LSPI, ...) approx. LP Monte Carlo estimation
Policy-search methods	Monte Carlo tree search Pegasus	Policy gradient (actor-critic)

- A third dimension — representation
  - Tabular, aggregation, linear, kernel, decision trees, neural nets, ...
- Other topics not covered so far
  - Model-based RL
  - Hierarchical RL
  - Multi-task RL, transfer RL, continual RL
  - Multi-agent RL

Further Readings

- Textbooks/monographs on RL in general: [4, 25, 31, 35, 40]
- Value function approximation: [5, 9, 37]
- Exploration: [3, 18, 26]
- Off-policy value estimation: [13, Chapter 4] [14]
- Hierarchical RL: [34]

References

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire.
The nonstochastic multiarmed bandit problem.
SIAM Journal on Computing, 32(1):48–77, 2002.

Leemon C. Baird.
Residual algorithms: Reinforcement learning with function approximation.
In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pages 30–37, 1995.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos.
Unifying count-based exploration and intrinsic motivation.
In *Advances in Neural Information Processing Systems (NIPS-16)*, pages 1471–1479, 2016.

Dimitri P. Bertsekas and John N. Tsitsiklis.
Neuro-Dynamic Programming.
Athena Scientific, September 1996.

Vivek S. Borkar.
Stochastic Approximation: A Dynamical Systems Viewpoint.
Cambridge University Press, 2008.

Olivier Chapelle and Lihong Li.
An empirical evaluation of Thompson sampling.
In *Advances in Neural Information Processing Systems 24 (NIPS-11)*, pages 2249–2257, 2012.

Yichen Chen, Lihong Li, and Mengdi Wang.
Scalable bilinear π-learning using state and action features, 2018.
In preparation.

References

Bo Dai, Albert Shaw, Niao He, Lihong Li, and Le Song.
Boosting the actor with dual critic.
In *Proceedings of the Sixth International Conference on Learning Representations (ICLR-18)*, 2018.

Bo Dai, Albert Shaw, Lihong Li, Lin Xiao, Niao He, Zhen Liu, Jianshu Chen, and Le Song.
SBEED learning: Convergent control with nonlinear function approximation, 2018.
In preparation.

Daniela Pucci de Farias and Benjamin Van Roy.
The linear programming approach to approximate dynamic programming.
Operations Research, 51(6):850–865, 2003.

Miroslav Dudík, John Langford, and Lihong Li.
Doubly robust policy evaluation and learning.
In *Proceedings of the Twenty-Eighth International Conference on Machine Learning (ICML-11)*, pages 1097–1104, 2011.
CoRR abs/1103.4601.

Geoffrey J. Gordon.
Stable function approximation in dynamic programming.
In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pages 261–268, 1995.

Katja Hofmann, Lihong Li, and Filip Radlinski.
Online evaluation for information retrieval.
Foundations and Trends in Information Retrieval, 10(1):1–117, 2016.

Nan Jiang and Lihong Li.
Doubly robust off-policy evaluation for reinforcement learning.
In *Proceedings of the Thirty-Third International Conference on Machine Learning (ICML-16)*, 2016.

References

Levente Kocsis and Csaba Szepesvári.
Bandit based Monte-Carlo planning.
In *Proceedings of the Seventeenth European Conference on Machine Learning (ECML-06)*, pages 282–293, 2006.

Michail G. Lagoudakis and Ronald Parr.
Least-squares policy iteration.
Journal of Machine Learning Research, 4:1107–1149, 2003.

Chandrashekar Lakshminarayanan, Shalabh Bhatnagar, and Csaba Szepesvári.
A linearly relaxed approximate linear program for Markov decision processes.
IEEE Transactions on Automatic Control, 63(4):1185–1191, 2017.

Lihong Li.
Sample complexity bounds of exploration.
In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State of the Art*, pages 175–204. Springer Verlag, 2012.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire.
A contextual-bandit approach to personalized news article recommendation.
In *Proceedings of the Nineteenth International Conference on World Wide Web (WWW-10)*, pages 661–670, 2010.

Lihong Li, Thomas J. Walsh, and Michael L. Littman.
Towards a unified theory of state abstraction for MDPs.
In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics (ISAIM-06)*, 2006.

Long-Ji Lin.
Self-improving reactive agents based on reinforcement learning, planning and teaching.
Machine Learning, 8(3–4):293–321, 1992.

References

Hamid Reza Maei, Csaba Szepesvári, Shalabh Bhatnagar, and Richard S. Sutton.
Toward off-policy learning control with function approximation.
In *ICML*, pages 719–726, 2010.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis.
Human-level control through deep reinforcement learning.
Nature, 518:529–533, 2015.

Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael L. Littman.
Analyzing feature generation for value-function approximation.
In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML-07)*, pages 737–744, 2007.

Martin L. Puterman.
Markov Decision Processes: Discrete Stochastic Dynamic Programming.
Wiley-Interscience, New York, 1994.

Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen.
A tutorial on Thompson sampling, 2018.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever.
Evolution strategies as a scalable alternative to reinforcement learning, 2017.
arXiv:1703.03864.

Paul J. Schweitzer and Abraham Seidmann.
Generalized polynomial approximations in Markovian decision processes.
Journal of Mathematical Analysis and Applications, 110(2):568–582, 1985.

References

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis.
Mastering the game of Go with deep neural networks and tree search.
Nature, 529:484–489, 2016.

Richard S. Sutton.
Generalization in reinforcement learning: Successful examples using sparse coarse coding.
In *Advances in Neural Information Processing Systems 8 (NIPS-95)*, pages 1038–1044, 1996.

Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning: An Introduction.
MIT Press, Cambridge, MA, March 1998.

Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning: An Introduction.
MIT Press, 2nd (draft) edition, 2018.
https://drive.google.com/file/d/1xeUDVGWGUUv1-ccUMAZHJLej2C7aAFWY/view?usp=sharing.

Richard S. Sutton, Hamid Maei, Doina Precup, Shalab Bhatnagar, Csaba Szepesvári, and Eric Wiewiora.
Fast gradient-descent methods for temporal-difference learning with linear function approximation.
In *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML-09)*, pages 993–1000, 2009.

Richard S. Sutton, Doina Precup, and Satinder P. Singh.
Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning.
Artificial Intelligence, 112(1–2):181–211, 1999.
An earlier version appeared as Technical Report 98-74, Department of Computer Science, University of Massachusetts, Amherst, MA 01003. April, 1998.

References

Csaba Szepesvri.
Algorithms for Reinforcement Learning.
Morgan & Claypool, 2010.

Gerald Tesauro.
Temporal difference learning and TD-Gammon.
Communications of the ACM, 38(3):58–68, March 1995.

John N. Tsitsiklis and Benjamin Van Roy.
An analysis of temporal-difference learning with function approximation.
IEEE Transactions on Automatic Control, 42:674–690, 1997.

Mengdi Wang.
Primal-dual π learning: Sample complexity and sublinear run time for ergodic Markov decision problems, 2017.
CoRR abs/1710.06100.

Shimon Whiteson and Peter Stone.
Evolutionary function approximation for reinforcement learning.
Journal of Machine Learning Research, 7:877–917, 2006.

Marco Wiering and Martijn van Otterlo.
Reinforcement Learning: State of the Art.
Springer, 2012.

Yinyu Ye.
The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate.
Mathematics of Operations Research, 36(4):593–603, 2011.

- Thanks to Dr. Lihong Li for the slides content.