L16: Deep Reinforcement Learning

Shan Wang

Lingnan College, Sun Yat-sen University

2020 Data Mining and Machine Learning LN3119
https://wangshan731.github.io/DM-ML/

Last lecture

- Introduction to RL
- Model-based RL
  - MDP Bellman Equation

$$V^{\pi}(s) = R(s) + \gamma \sum_{s\prime \in S} P_{s\pi(s)}(s')\, V^{\pi}(s')$$

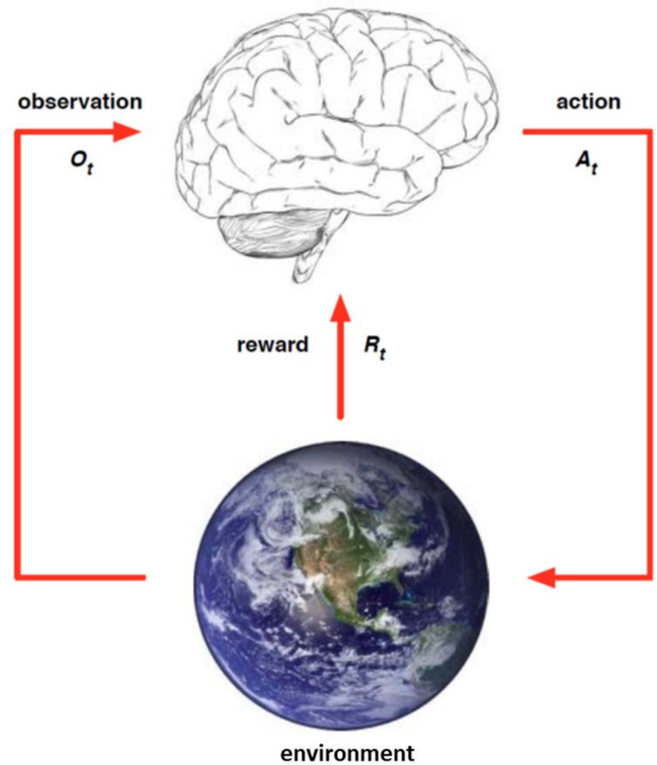
  - Value iteration

$$V(s) \; = R(s) + \gamma \max_{a \in A} \sum_{s\prime \in S} P_{sa}(s') V(s')$$

  - Policy iteration

$$\pi(s) = \operatorname*{argmax}_{a \in A} \sum_{s\prime \in S} P_{sa}(s') V^{\pi}(s')$$

  - Estimate P_{sa} and $R(s)$



Course Outline

- Supervised learning
  - Linear regression
  - Logistic regression
  - SVM and kernel
  - Tree models
- Unsupervised learning
  - Clustering
  - PCA
  - EM
- Deep learning
  - Neural networks
  - Convolutional NN
  - Recurrent NN
- Reinforcement learning
  - MDP
  - ADP
  - Deep Q-Network

This lecture

- Model free reinforcement learning
- Approximation in reinforcement learning
- Introduction to deep reinforcement learning
- Value-based DRL
  - DQN

Reference: CS 420, Weinan Zhang (SJTU)

Model-free RL

Model-free Prediction

Model-free Reinforcement Learning

- In realistic problems, often the state transition and reward function are not explicitly given
- Model-free RL is to directly learn value & policy from experience without building an MDP
- Key steps: (1) estimate value function; (2) optimize policy

Value Function Estimation

- In model-based RL (MDP), the value function is calculated by dynamic programming

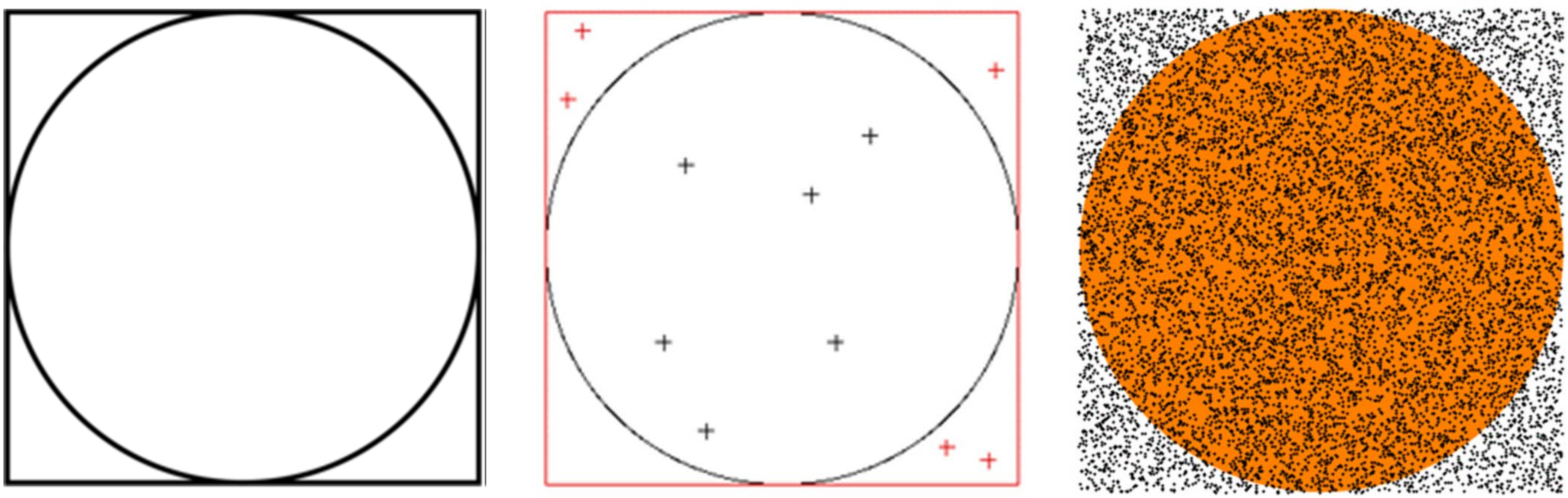
$$V^{\pi}(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi]$$

$$= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

- Now in model-free RL
  - We cannot directly know P_{sa} and R
  - But we have a list of experiences to estimate the values

Monte-Carlo Methods

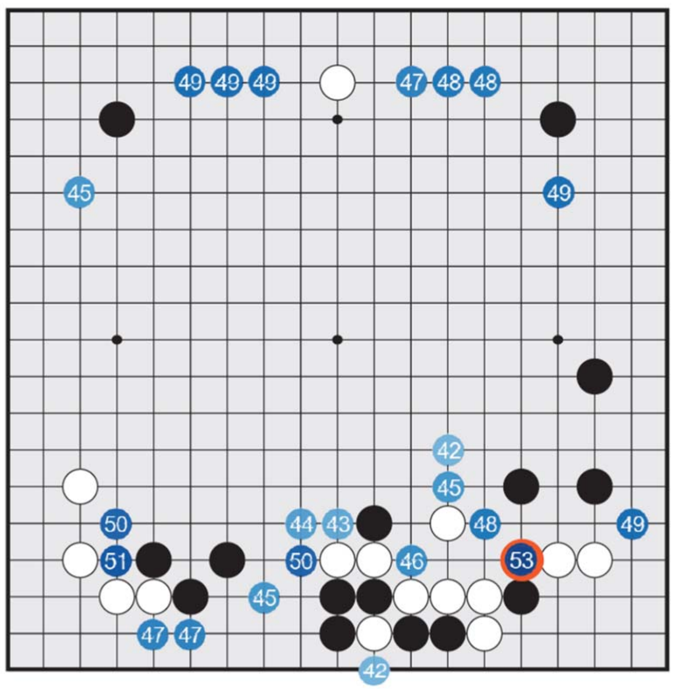
- Monte-Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.
- Example, to calculate the circle's surface

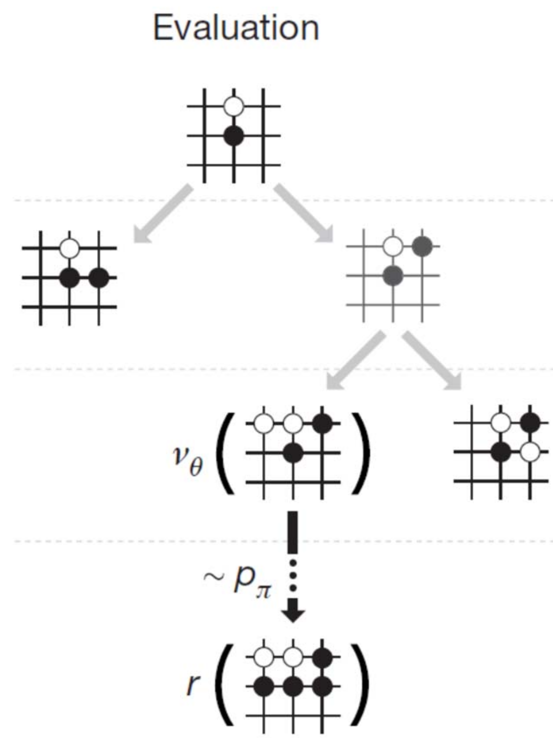


$$\text{Circle Surface} = \text{Square Surface} \times \frac{\#\text{points in circle}}{\#\text{points in total}}$$

Monte-Carlo Methods

- Go: to estimate the winning rate given the current state





$$\text{Win Rate }(s) = \frac{\#\text{win simulation cases started from } s}{\#\text{simulation cases started from } s \text{ in total}}$$

Monte-Carlo Value Estimation

- Goal: learn V^π from experience under policy π
- Recall that the return is the total discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \gamma^{T-1} R_T$$

- Recall that the value function is the expected return

$$V^\pi(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi]$$
$$= \mathbb{E}[G_t | s_t = s, \pi]$$
$$\simeq \frac{1}{N} \sum_{i=1}^{N} G_t^{(i)}$$

- Sample N episodes from state s using policy π
- Calculate the average of cumulative reward
- $G_t^{(i)}$ is the G_t of ith sample

- Monte-Carlo policy evaluation uses empirical mean return instead of expected return

Monte-Carlo Value Estimation

- Implementation
  - Sample episodes under policy π

$$s_0^{(i)} \xrightarrow[R_1^{(i)}]{a_0^{(i)}} s_1^{(i)} \xrightarrow[R_2^{(i)}]{a_1^{(i)}} s_2^{(i)} \xrightarrow[R_3^{(i)}]{a_2^{(i)}} s_3^{(i)} \cdots s_T^{(i)} \sim \pi$$

- Every time-step t that state s is visited in an episode
  - Increment counter $N(s) = N(s) + 1$
  - Increment total return $S(s) = S(s) + G_t$
  - Value is estimated by mean return $V(s) = S(s)/N(s)$
  - By law of large numbers

$$V(s) \longrightarrow V^{\pi(S)} \text{ as } N(s) \longrightarrow \infty$$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after each episode
- For each state S_t with cumulative return G_t

$$N(s_t) = N(s_t) + 1$$

$$V(s_t) = V(s_t) + \frac{1}{N(s_t)}(G_t - V(s_t))$$

- For non-stationary problems (i.e. the environment could be varying over time), it can be useful to track a running mean, i.e. forget old episodes

$$V(s_t) = V(s_t) + \alpha(G_t - V(s_t))$$

Monte-Carlo Value Estimation

Idea: $V(s) = \frac{1}{N}\sum_{i=1}^{N} G_t^{(i)}$

Implementation: $V(S_t) = V(S_t) + \alpha(G_t - V(S_t))$

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

Temporal-Difference Learning

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = R_{t+1} + \gamma V(S_{t+1})$$

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

Observed reward   Guess of future

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

State Value and Action Value

- $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$
- State value
  - The state-value function $V^\pi(s)$ of an MDP is the expected return starting from state s and then following policy π

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- Action value
  - The action-value function $Q^\pi(s,a)$ of an MDP is the expected return starting from state s, taking action a, and then following policy π

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Bellman Expectation Equation

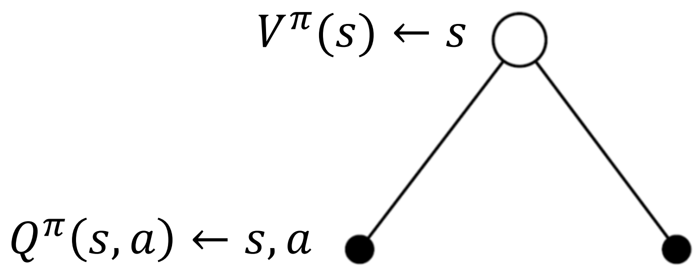
- The state-value function $V^{\pi}(s)$ can be decomposed into immediate reward plus discounted value of successor state

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma V^{\pi}(S_{t+1}) \, | S_t = s]$$

- The action-value function $Q^{\pi}(s,a)$ can similarly be decomposed

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma Q^{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

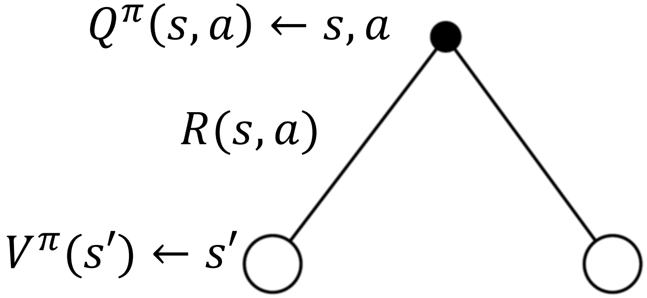
State Value and Action Value



Action value of *(s,a)*

$$V^\pi(s) = \sum_{a \in A} \pi(a|s) Q^\pi(s,a)$$

Probability of taking action *a* at state *s*



Probability of next state *s'*

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P_{sa}(s') V^\pi(s')$$

Current reward of action *a* at state *s*

State of *s'*

Temporal-Difference (TD) Learning

- For state value

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- For action value

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

- Learn it step by step
- Faster than MC
- Small variance
- But biased

Approximation in RL

Value and Policy Approximation

- In previous models, we have created a lookup table to maintain a variable $V(s)$ for each state or $Q(s, a)$ for each state-action
- What if we have a large MDP, i.e.
  - the state or state-action space is too large
  - or the state or action space is continuous

to maintain $V(s)$ for each state or $Q(s, a)$ for each state-action?

- For example
  - Game of Go (10170 states)
  - Helicopter, autonomous car (continuous state space)

Parametric Value Function Approximation

- Create parametric (thus learnable) functions to approximate the value function

$$V_\theta(s) \simeq V^\pi(s)$$

$$Q_\theta(s,a) \simeq Q^\pi(s,a)$$

- θ is the parameters of the approximation function, which can be updated by reinforcement learning
- Generalize from seen states to unseen states

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$V_\theta(s) = \theta' x(s)$$

- Objective function is quadratic in parameters θ

$$J(\theta) = \mathbb{E}_\pi[\frac{1}{2}(V^\pi(s) - \theta' x(s))^2]$$

- Update θ

$$\theta \leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$$
$$= \theta + \alpha(V^\pi(s) - V_\theta(s))x(s)$$

TD Learning with Value Function Approx.

$$\theta \leftarrow \theta + \alpha(V^{\pi}(s) - V_{\theta}(s))x(s)$$

- TD target is a biased sample $R_{t+1} + \gamma V_{\theta}(S_{t+1})$ of true target value $V^{\pi}(s)$
- Supervised learning from "training data"
  $$\langle S_1, R_2 + \gamma V_{\theta}(S_2)\rangle, \langle S_2, R_3 + \gamma V_{\theta}(S_3)\rangle, \ldots$$
- For each data instance $\langle S_t, R_{t+1} + \gamma V_{\theta}(S_{t+1})\rangle$
  $$\theta \leftarrow \theta + \alpha(R_{t+1} + \gamma V_{\theta}(S_{t+1}) - V_{\theta}(S_t))x(S_t)$$
- Linear TD converges (close) to global optimum

Linear Action-Value Function Approx.

- Parametric Q function, e.g., the linear case

$$Q_\theta(s,a) = \theta' x(s,a)$$

- Stochastic gradient descent update

$$\theta \leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$$

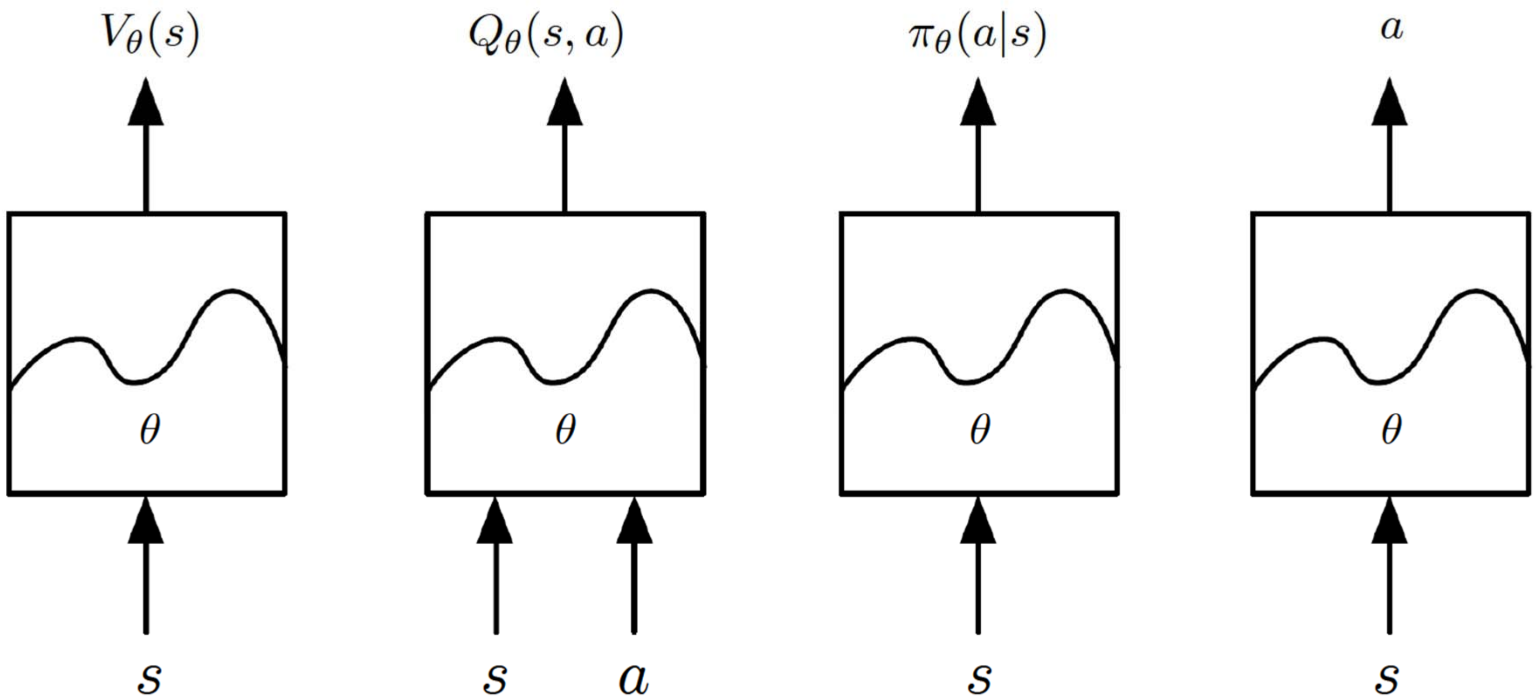
$$= \theta + \alpha(Q^\pi(s,a) - Q_\theta(s,a))x(s,a)$$

- TD Learning with Value Function Approx.
  - The target is $R_{t+1} + \gamma Q_\theta(S_{t+1}, A_{t+1})$

$$\theta \leftarrow \theta + \alpha(R_{t+1} + \gamma Q_\theta(S_{t+1}, A_{t+1}) - Q_\theta(S_t, A_t))x(S_t, A_t)$$

Introduction to Deep RL

Value and Policy Approximation



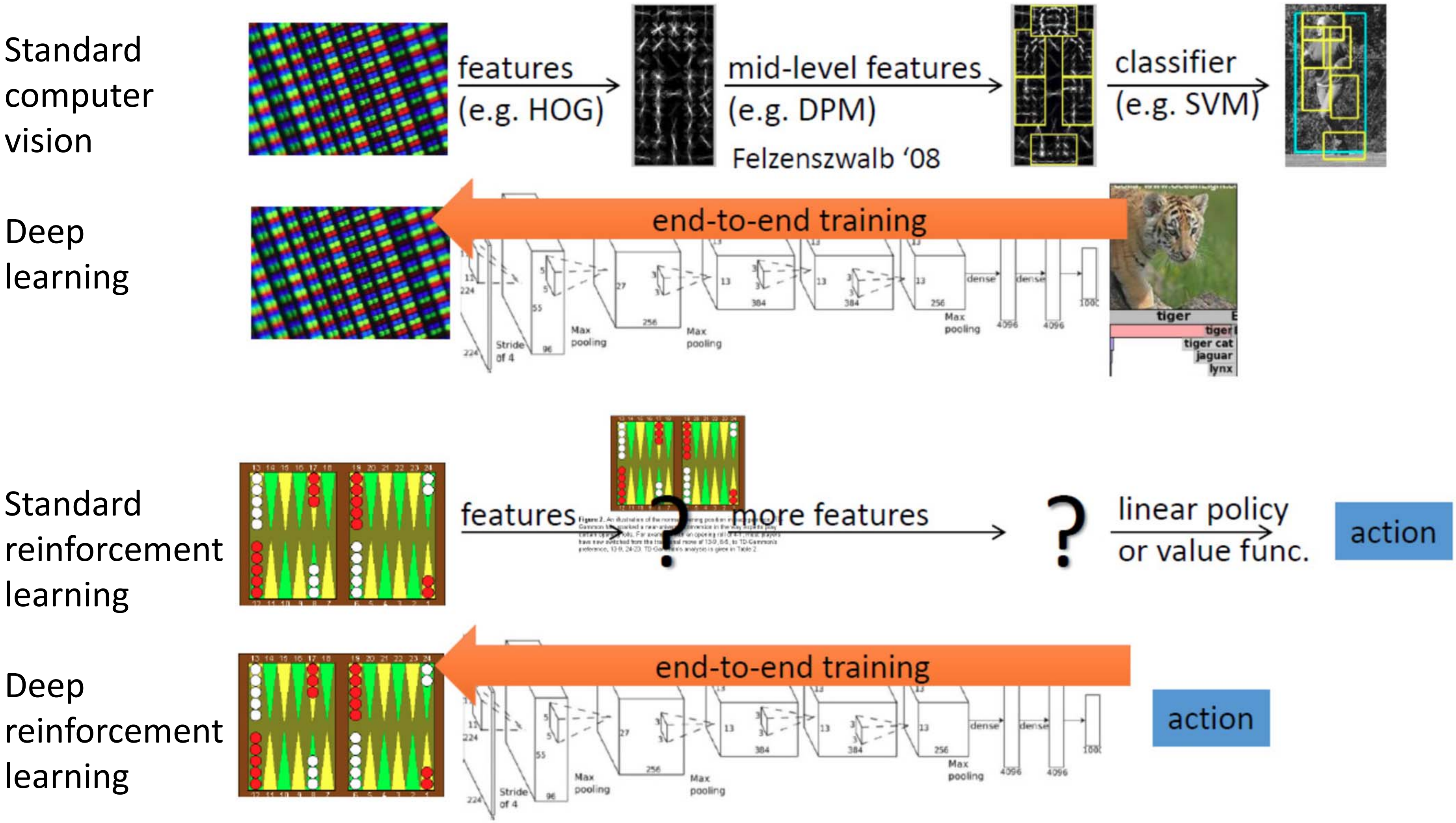
State-value and action-value approx.

Stochastic policy approx.

Deterministic policy approx.

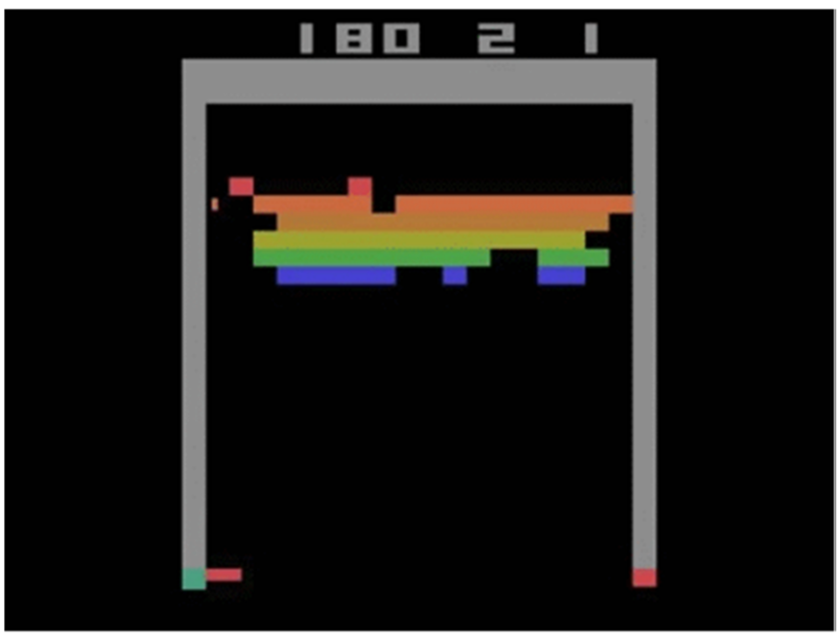
What if we directly build these approximate function with deep neural networks?

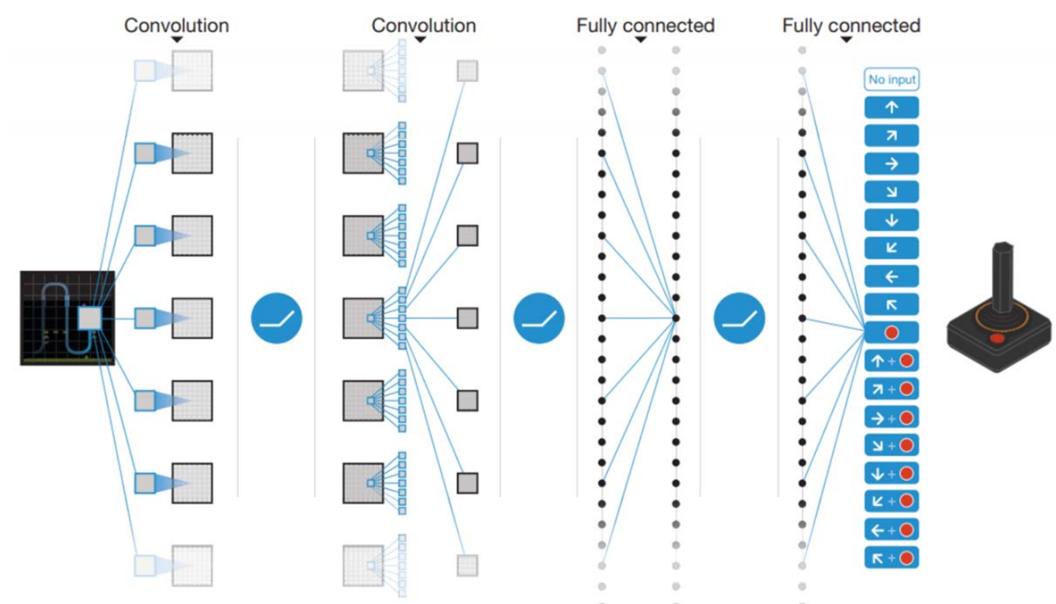
End-to-End Reinforcement Learning



Deep Reinforcement Learning

- Deep Reinforcement Learning
  - leverages deep neural networks for value functions and policies approximation
  - so as to allow RL algorithms to solve complex problems in an end-to-end manner





Deep Reinforcement Learning Trends

- What will happen when combining DL and RL?
  - Value functions and policies are now deep neural nets
  - Very high-dimensional parameter space
  - Hard to train stably
  - Easy to overfit
  - Need a large amount of data
  - Need high performance computing
  - Balance between CPUs (for collecting experience data) and GPUs (for training neural networks)
  - …
- These new problems motivates novel algorithms for DRL

Deep Reinforcement Learning Categories

- Value-based methods
  - Deep Q-network
- Stochastic policy-based methods
  - Policy gradients with NNs, natural policy gradient, trust-region policy optimization, proximal policy optimization, A3C
- Deterministic policy-based methods
  - Deterministic policy gradient, DDPG

DQN

Deep Q-network

Q-Learning and Policy Control

- Recall TD learning for action value *Q*(*s*,*a*)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

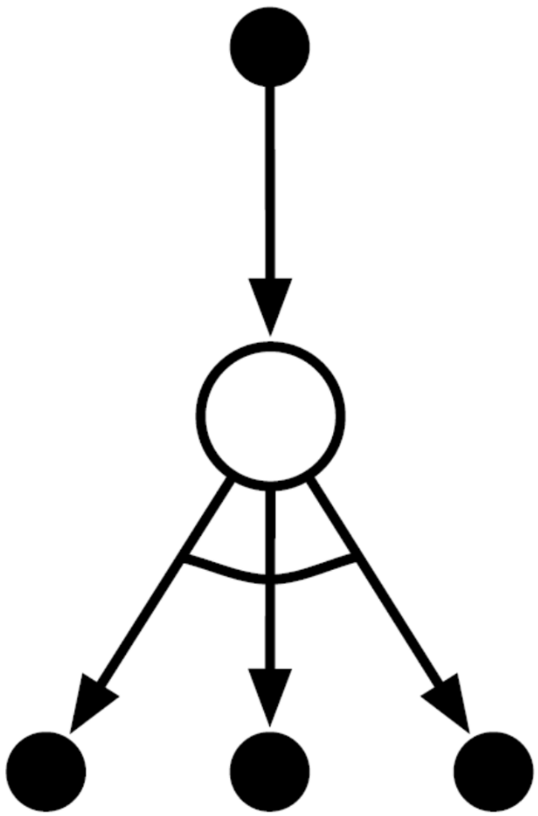
- The next action is chosen using policy

$$a_{t+1} \sim \pi(\cdot \mid S_{t+1})$$

- What if we consider another good action a'?
- And update $Q(s_t, a_t)$ towards value of a'

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \underbrace{\max_{a' \in A} Q(S_{t+1}, a')}_{\text{Policy Control}} - Q(S_t, A_t))$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_{a' \in A} Q(S_{t+1}, a') - Q(S_t, A_t)\right)$$



At state *s*, take action *a*

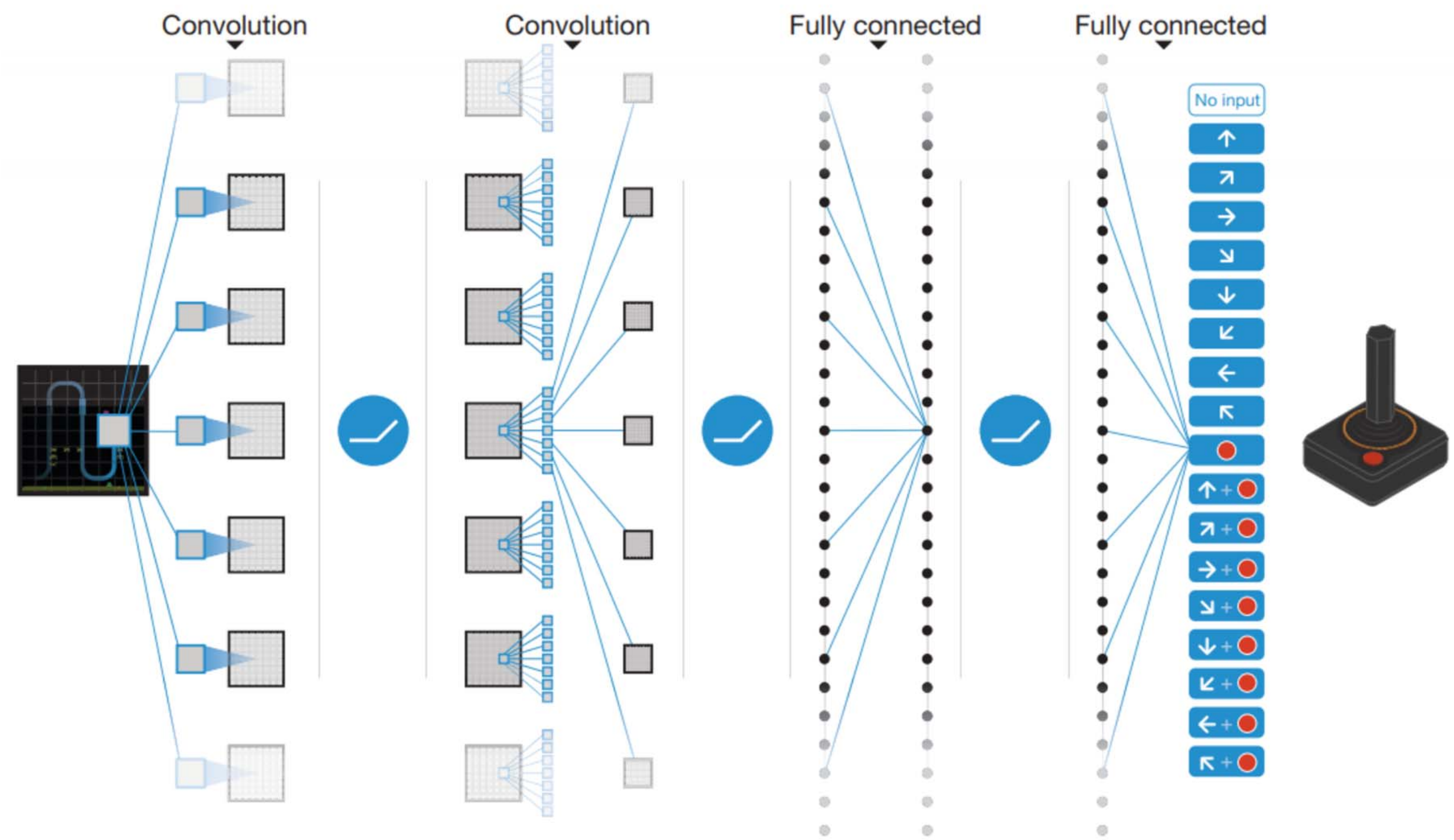
Observe reward *R*

Transit to the next state *s*’

At state *s*’, take action argmax *Q*(s’,*a*’)

Deep Q-Network (DQN)

- Implement Q function with deep neural network
  - Input a state, output Q values for all actions



Deep Q-Network (DQN)

- The loss function at iteration *i*

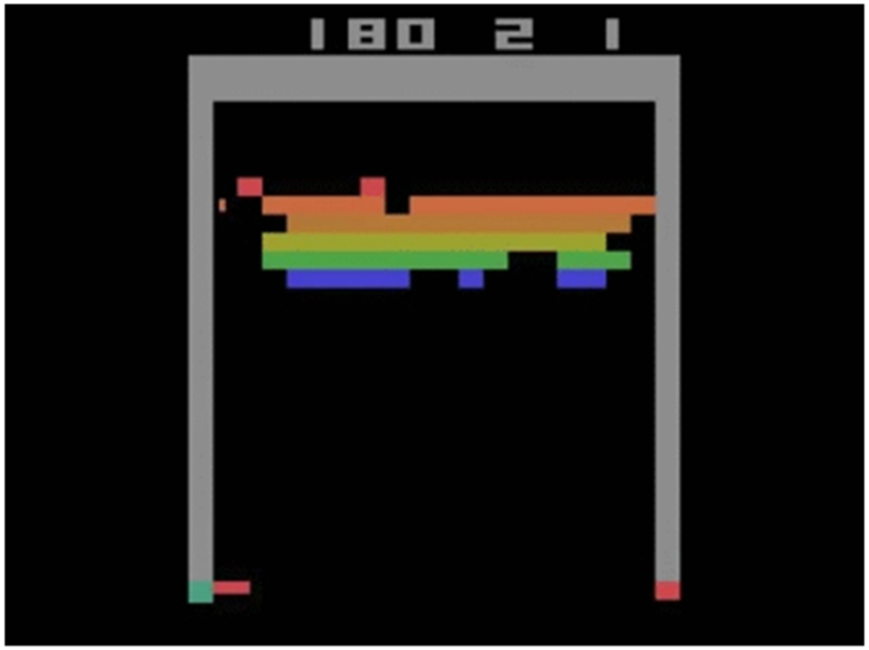
$$L_i(\theta_i) = \mathbb{E}_{(s,a,R,s')\sim U(D)}\left[\left(\boxed{R + \gamma \max_{a'} Q(s',a';\theta_i^-)} - \boxed{Q(s,a;\theta_i)}\right)^2\right]$$

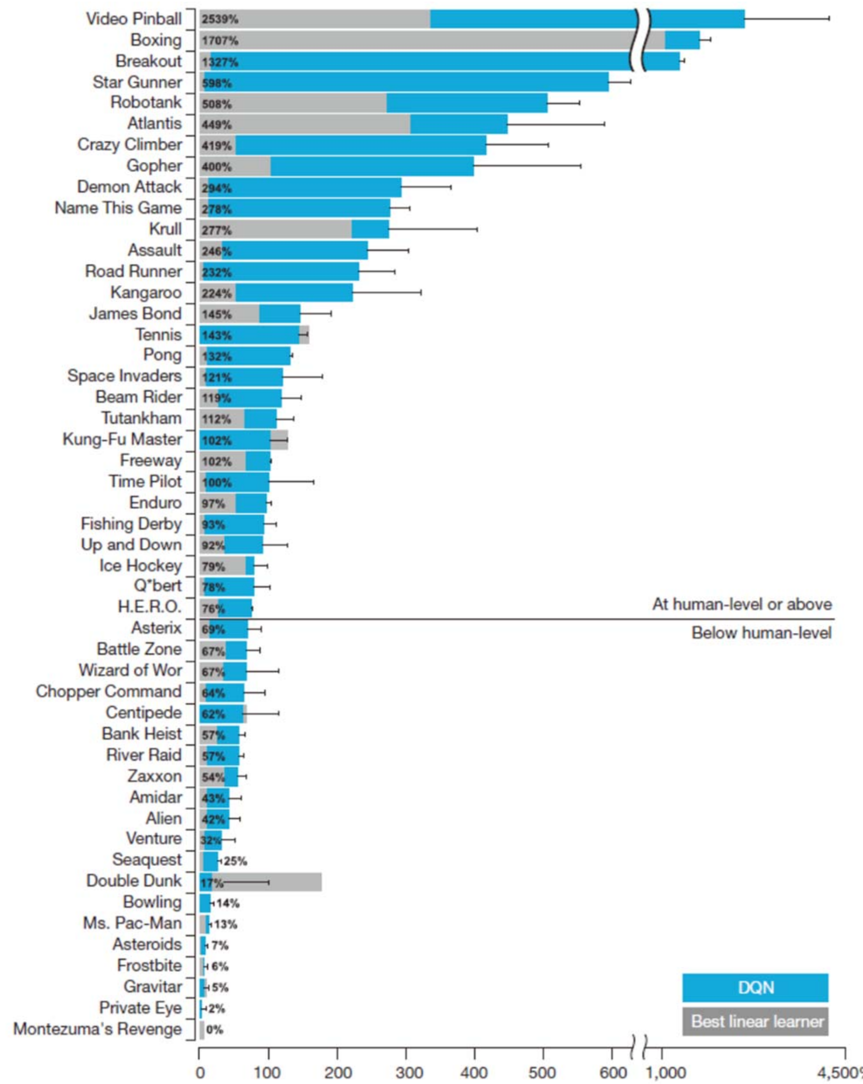
Target Q-value
(Target output)

Estimated Q-value
(Estimated output)

- θ_i are the network parameters to be updated at iteration *i*
  - Updated with standard back-propagation algorithms
- θ_i^- are the target network parameters
  - Only updated with ϑi for every *C* steps
- $(s,a,R,s')\sim U(D)$: the samples are uniformly drawn from the experience pool *D*
  - Thus to avoid the overfitting to the recent experiences

Deep Q-Network (DQN)





DQN (NIPS 2013) is the beginning of the entire deep reinforcement learning subarea.

Lecture 16 Wrap-up

- ✓ Model free reinforcement learning
- ✓ Approximation in reinforcement learning
- ✓ Introduction to deep reinforcement learning
- ✓ Value-based DRL
  - ✓ DQN

2020 Data Mining and Machine Learning LN3119

https://wangshan731.github.io/DM-ML/

Questions?

Shan Wang (王杉)

https://wangshan731.github.io/