Lecture 14: Learning 4

http://cs.nju.edu.cn/yuy/course_ai18.ashx

How to train a dog?

PHASE 1

DOWN

How to train a dog?

hear “down”

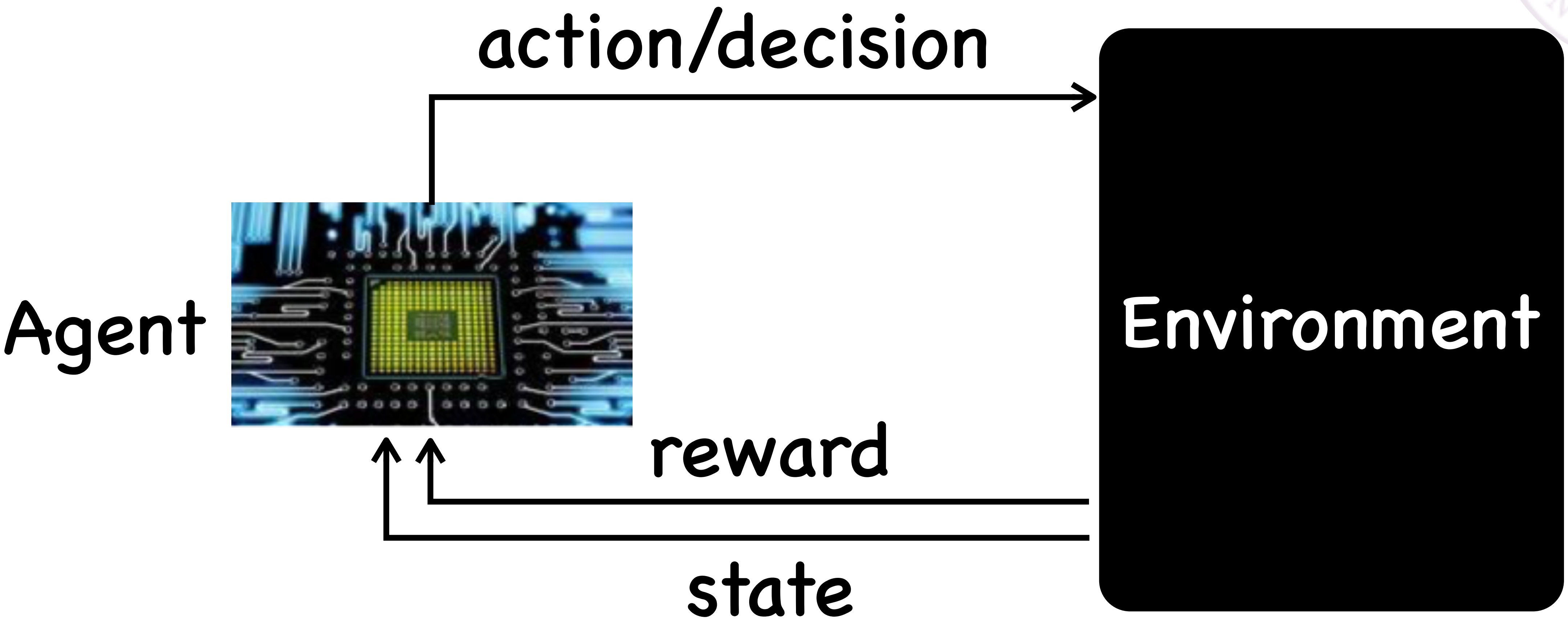
reward

action

dog learns from rewards to adapt to the environment

can computers do similarly?

Reinforcement learning setting



$$<A,\ S,\ R,\ P>$$

Action space: A

State space: S

Reward: $R : S \times A \times S \to \mathbb{R}$

Transition: $P : S \times A \to S$

Reinforcement learning setting

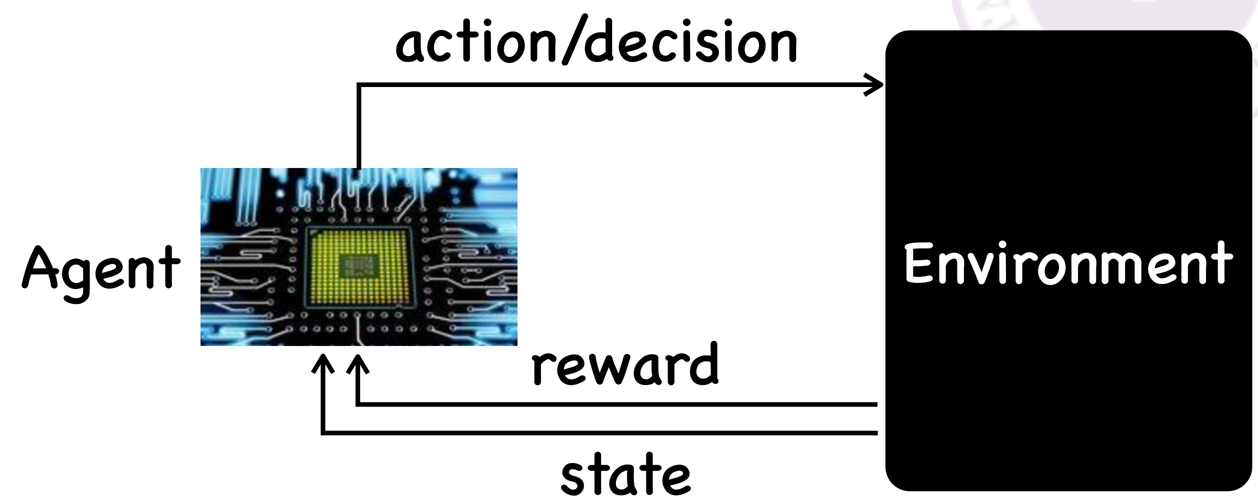
$<A,\ S,\ R,\ P>$

Action space: A

State space: S

Reward: $R : S \times A \times S \to \mathbb{R}$

Transition: $P : S \times A \to S$



Agent:

Policy: $\pi : S \times A \to \mathbb{R}, \quad \sum_{a \in A} \pi(a|s) = 1$

Policy (deterministic): $\pi : S \to A$

Agent's view: $s_0,\ a_0,\ r_1, s_1,\ a_2,\ r_2, s_2,\ a_3,\ r_3, s_3, \ldots$

$\pi(s_0)$ $\pi(s_1)$ $\pi(s_2)$

Reinforcement learning setting

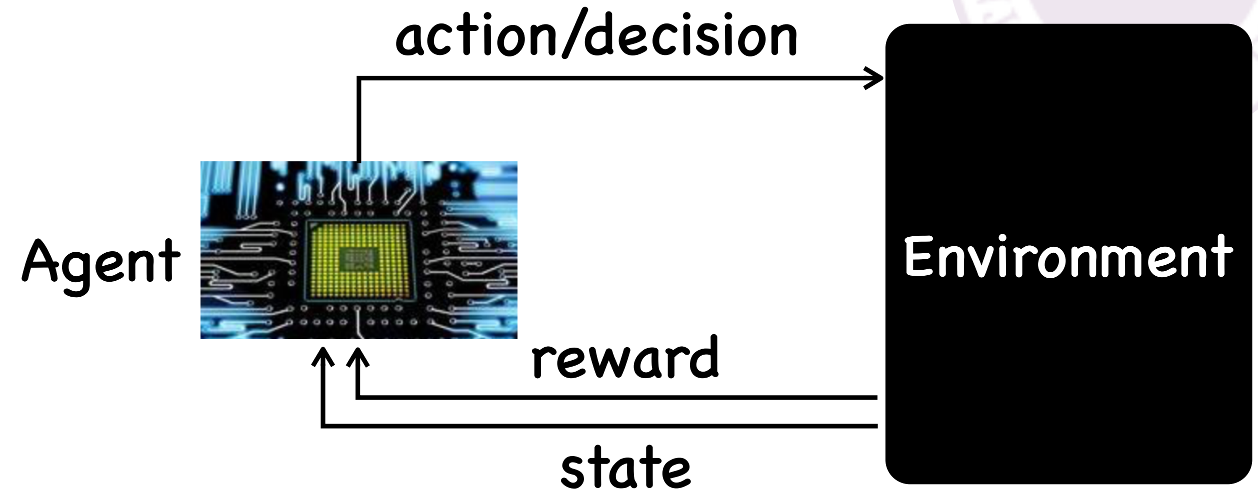
$<A,\ S,\ R,\ P>$

Action space: A

State space: S

Reward: $R : S \times A \times S \to \mathbb{R}$

Transition: $P : S \times A \to S$



Agent: Policy: $\pi : S \times A \to \mathbb{R}, \quad \sum_{a \in A} \pi(a|s) = 1$

Policy (deterministic): $\pi : S \to A$

Agent's goal:

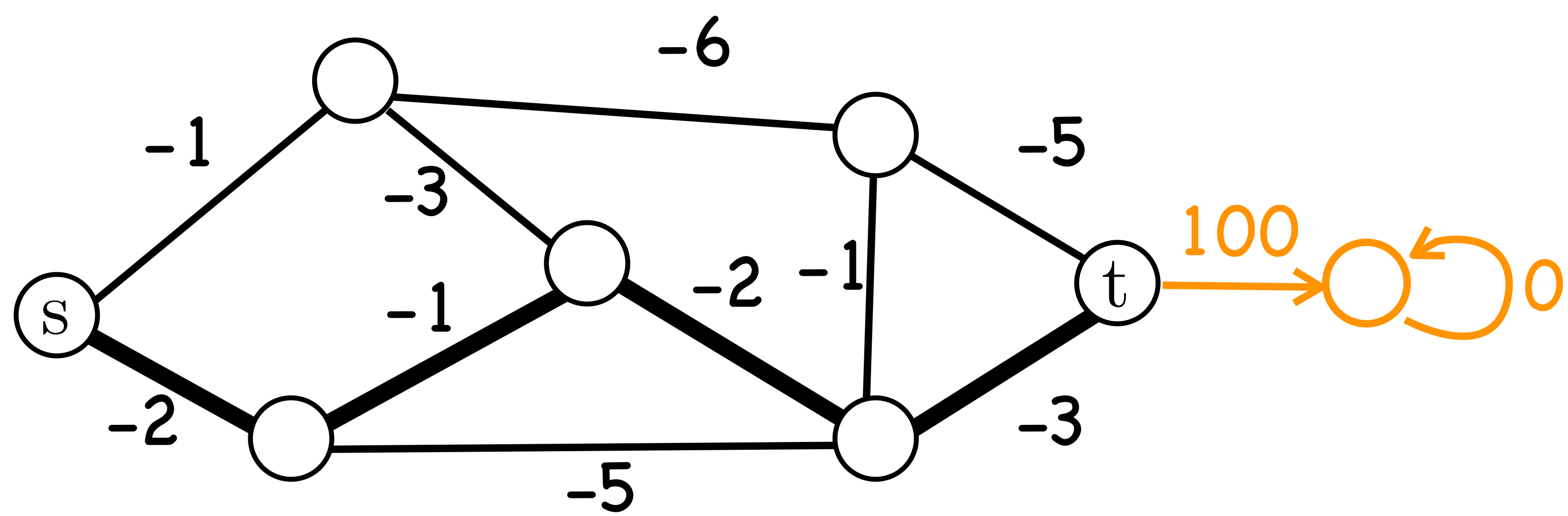
learn a policy to maximize long-term total reward

T-step: $\sum_{t=1}^{T} r_t$ discounted: $\sum_{t=1}^{\infty} \gamma^t r_t$

all RL tasks can be defined by maximizing total reward

Reward examples

shortest path:



- every node is a state, an action is an edge out
- reward function = the negative edge weight
- optimal policy leads to the shortest path

Difference between RL and planning?

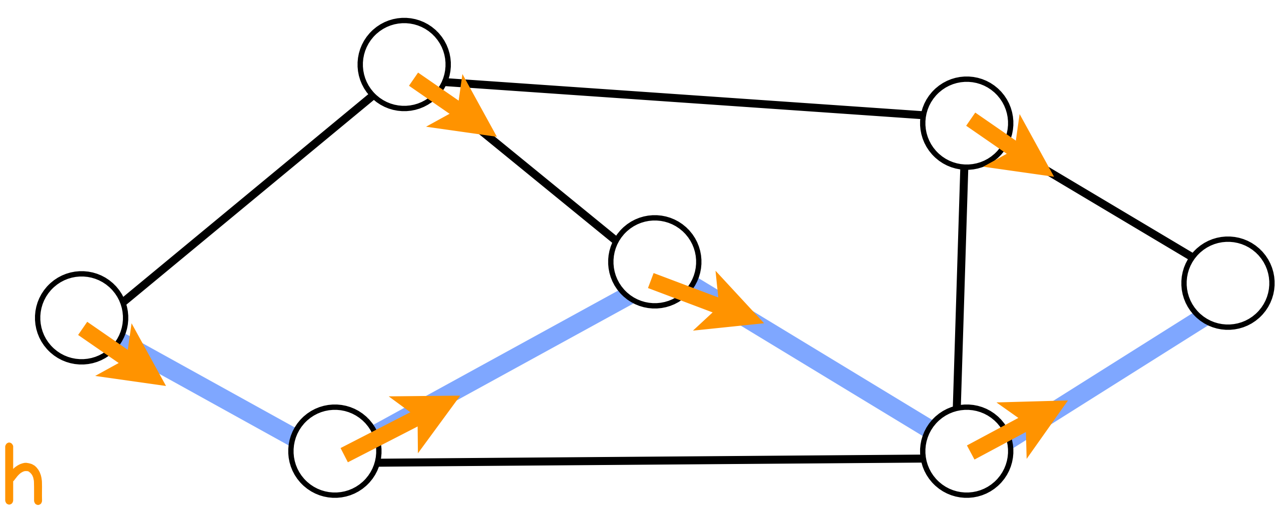
what if we use planning/search methods to find actions that maximize total reward

Planing: find an optimal solution

RL: find an optimal policy from samples

planning: shortest-path

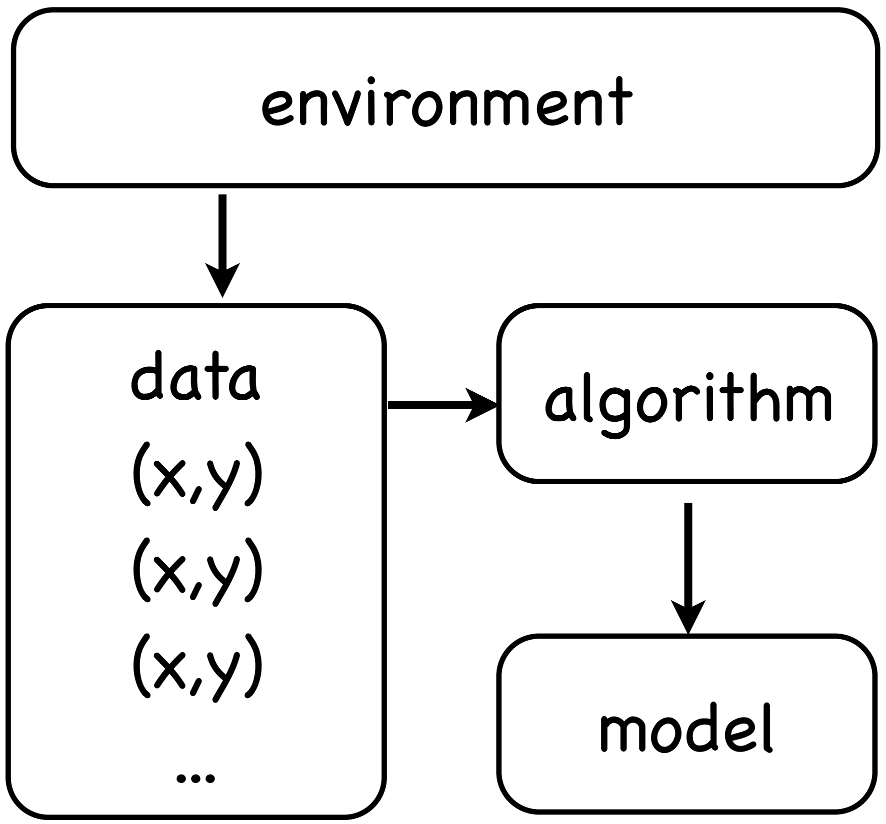
RL: shortest-path policy without knowing the graph



Difference between RL and SL?

supervised learning also learns a model ...

supervised learning

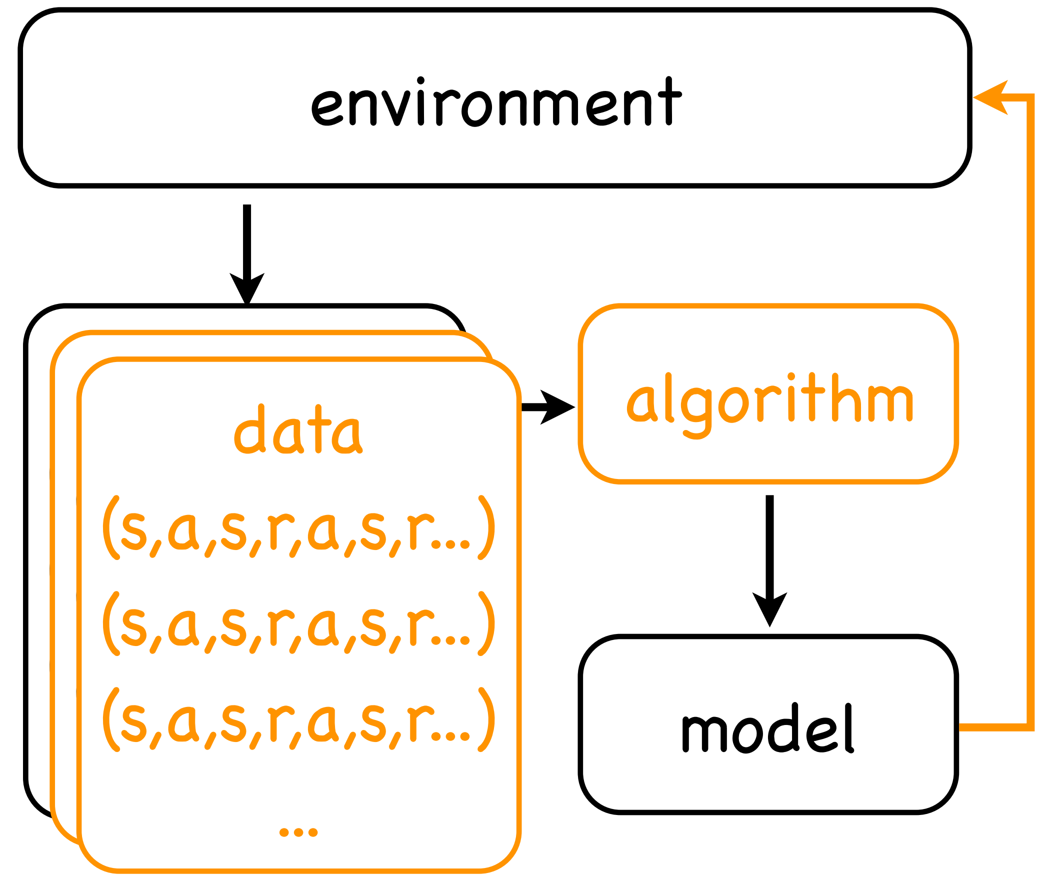


learning from labeled data

open loop

passive data

reinforcement learning



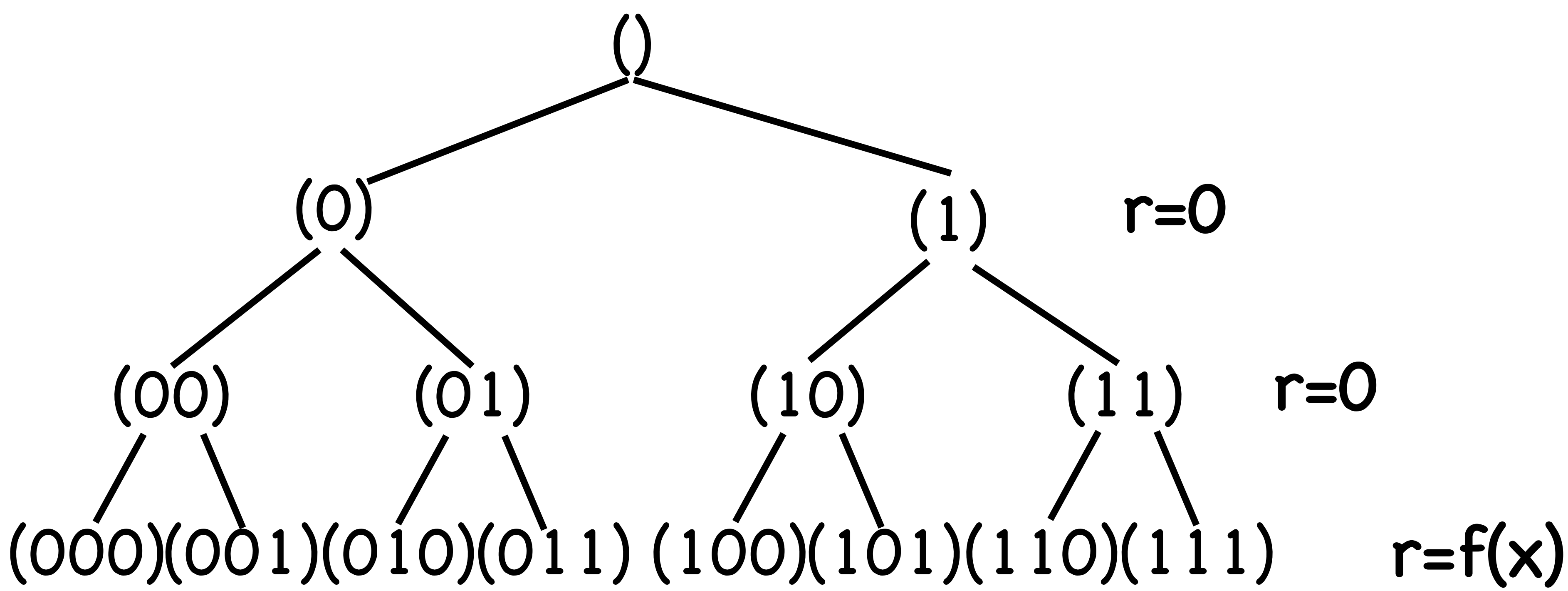
learning from delayed reward

closed loop

explore environment

Reward examples

general binary space problem $\max_{x \in \{0,1\}^n} f(x)$

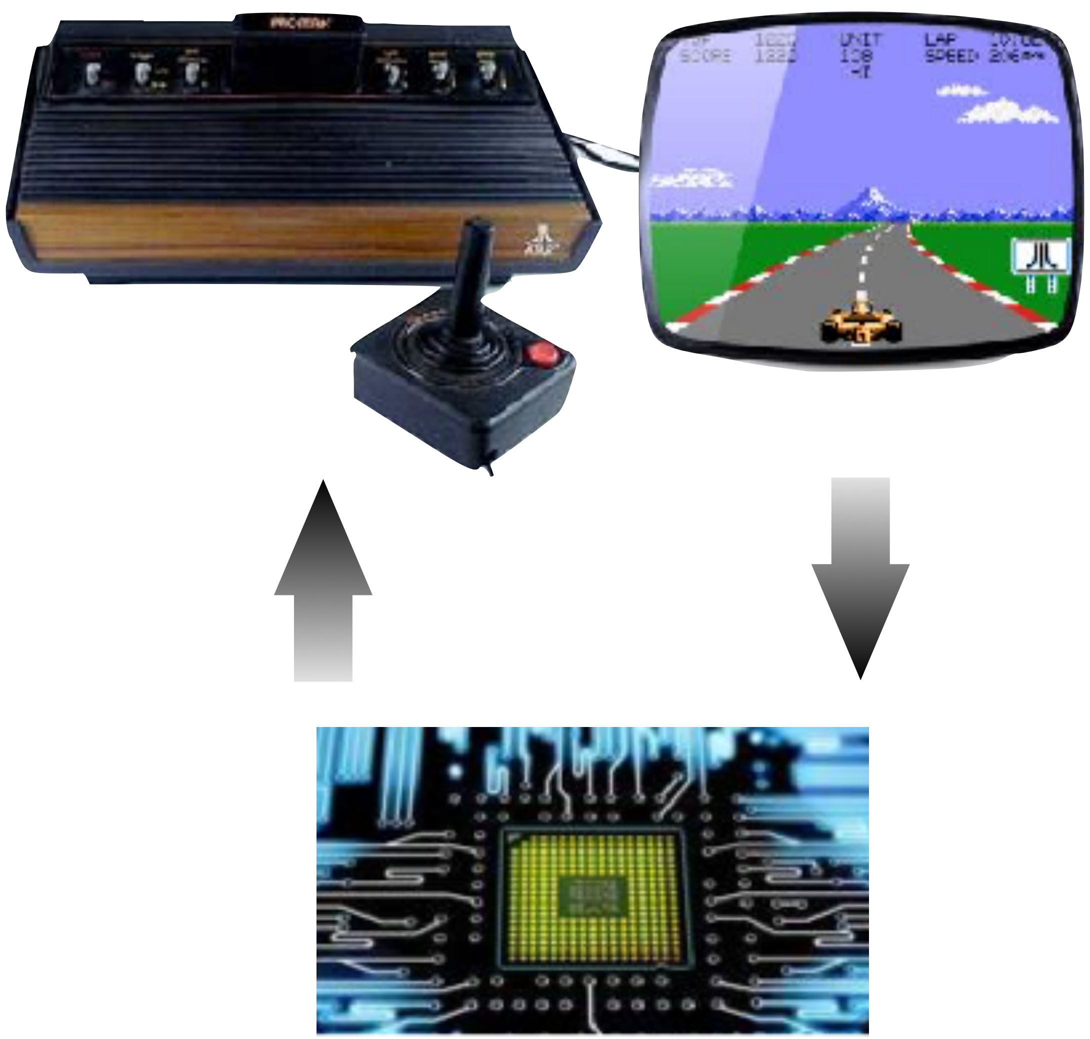


solving the optimal policy is NP-hard!

Applications

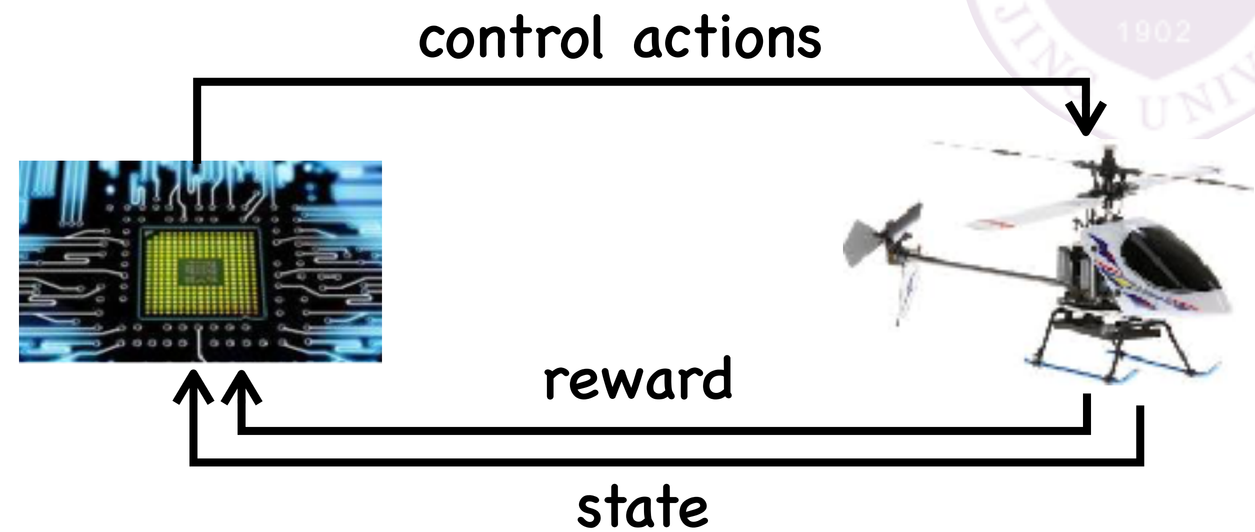
Deepmind Deep Q-learning on Atari

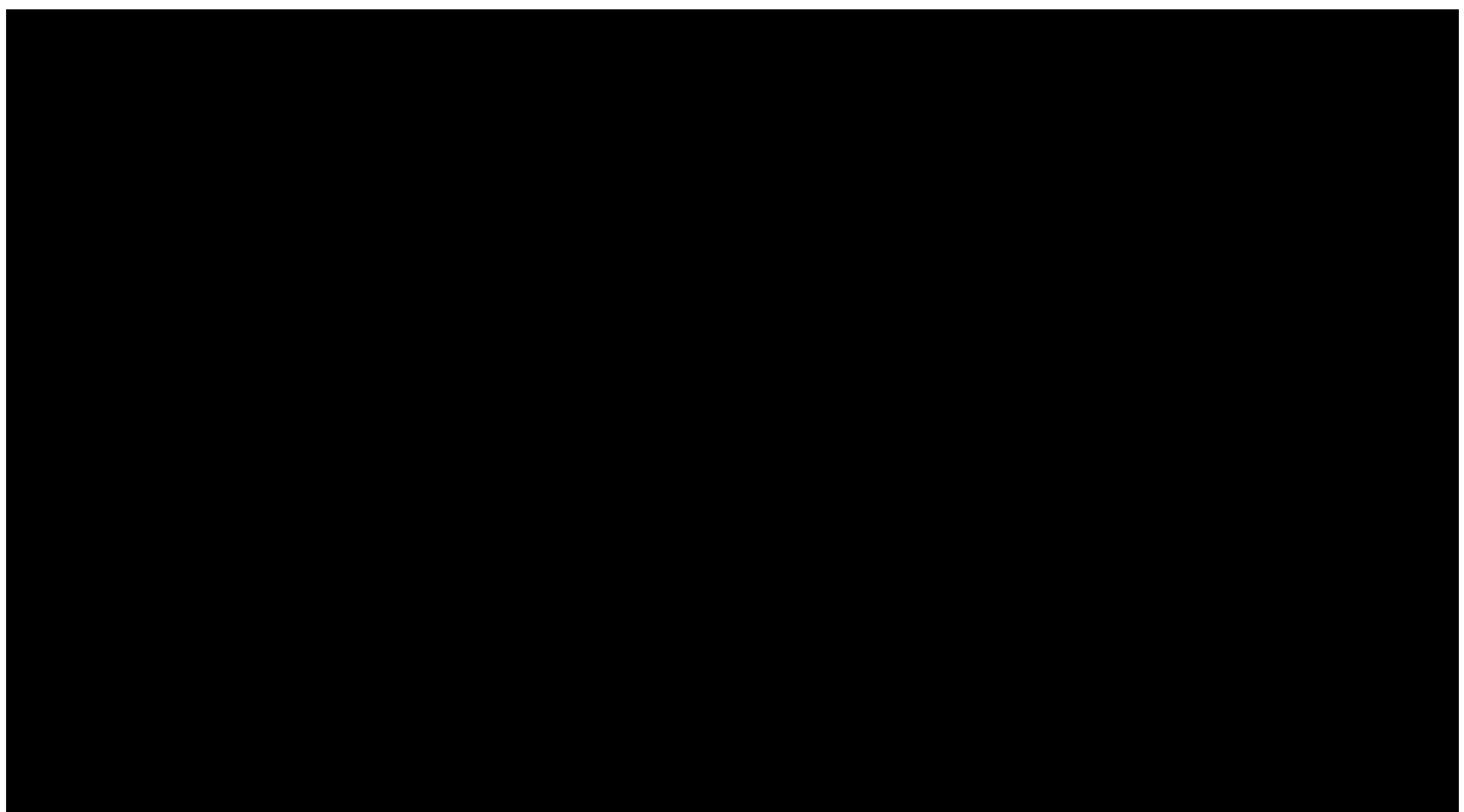
[Mnih *et al*. Human-level control through deep reinforcement learning. Nature, 518(7540): 529-533, 2015]



Applications

learning robot skills





https://www.youtube.com/watch?v=VCdxqn0fcnE

More applications

Search

Recommendation system

Stock prediction

...

every decision changes the world

Markov Decision Process

essential mathematical model for RL

Markov Process

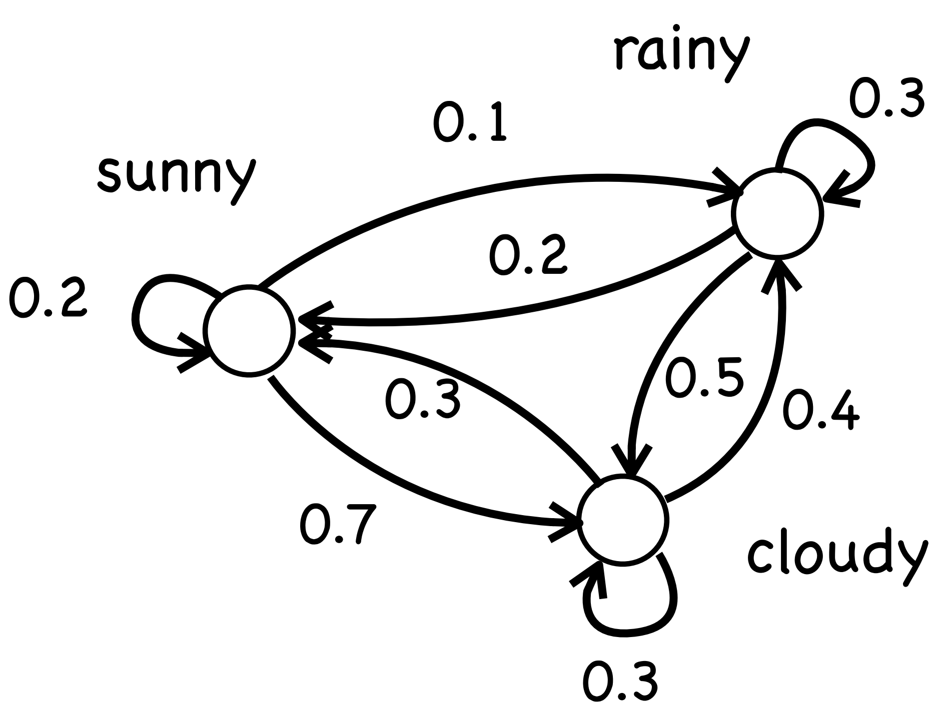
(finite) state space S, transition matrix P

a process $s_0, s_1, \ldots$ is Markov if

has no memory

$$P(s_{t+1} \mid s_t, \ldots, s_0) = P(s_{t+1} \mid s_t)$$

discrete S -> Markov chain



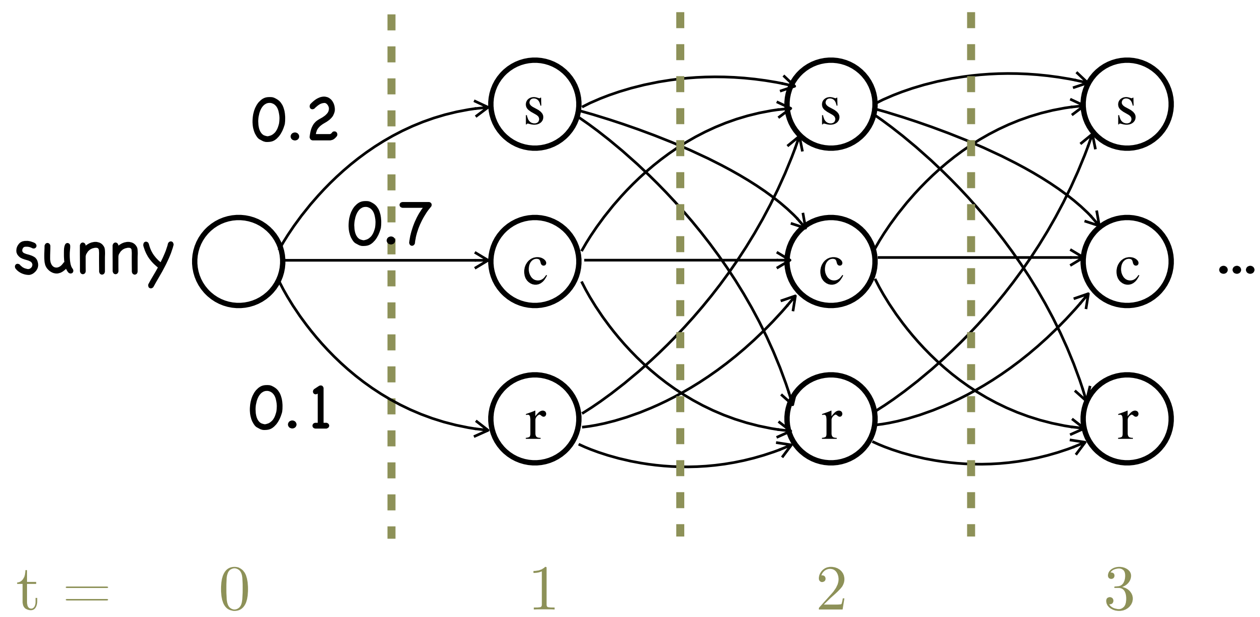
$P =$

	s	c	r
sunny	0.2	0.7	0.1
cloudy	0.3	0.3	0.4
rainy	0.2	0.5	0.3

$$\boldsymbol{s}_{t+1} = \boldsymbol{s}_t P = \boldsymbol{s}_0 P^{t+1}$$

Markov Process

horizontal view



stationary distribution: $s == sP$

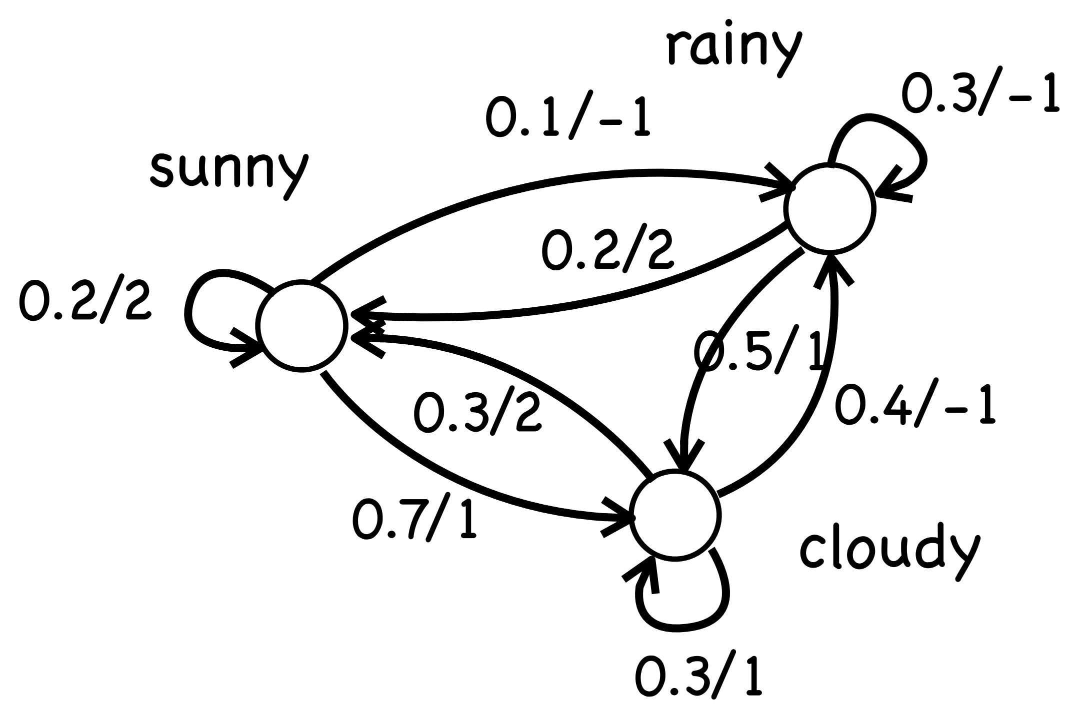
sampling from a Markov process:

s, c, c, r ...

s, c, s, c ...

Markov Reward Process

introduce reward function R



how to calculate the long-term total reward?

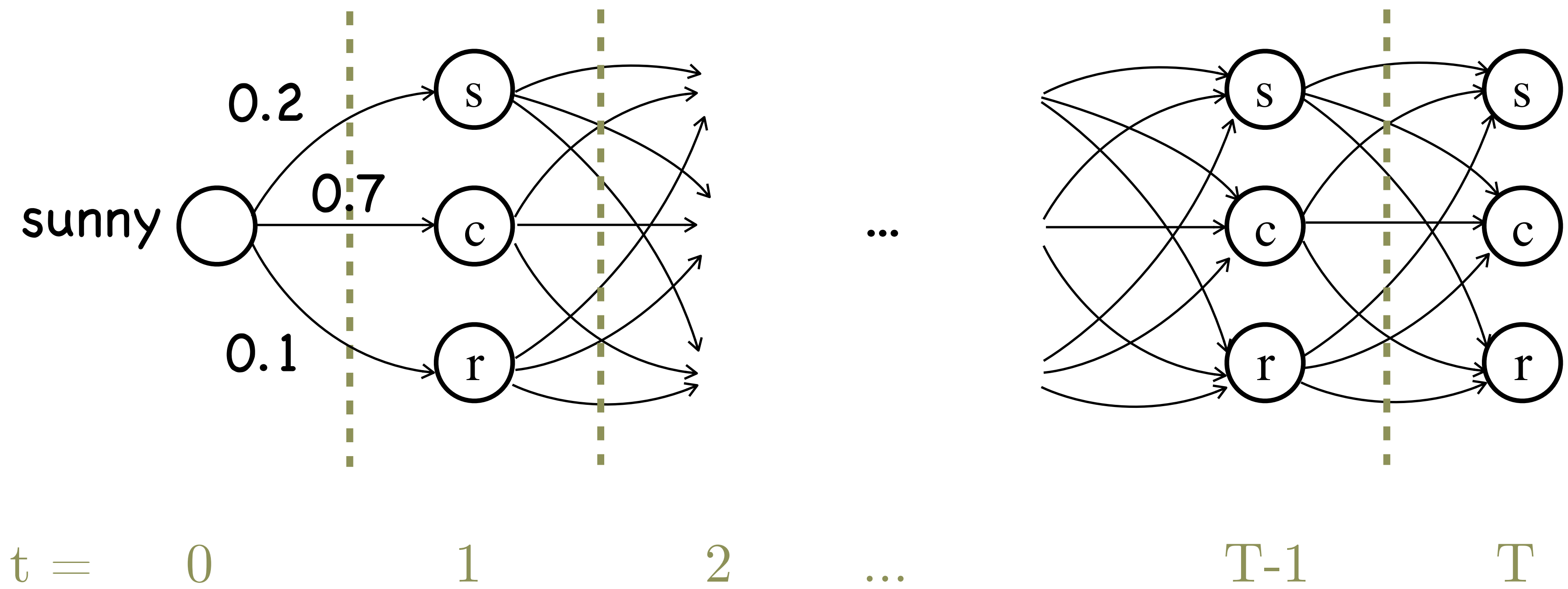
$$V(\text{sunny}) = E[\sum_{t=1}^{T} r_t | s_0 = \text{sunny}]$$

$$V(\text{sunny}) = E[\sum_{t=1}^{\infty} \gamma^t r_t | s_0 = \text{sunny}]$$

value function

Markov Reward Process

horizontal view: consider T steps



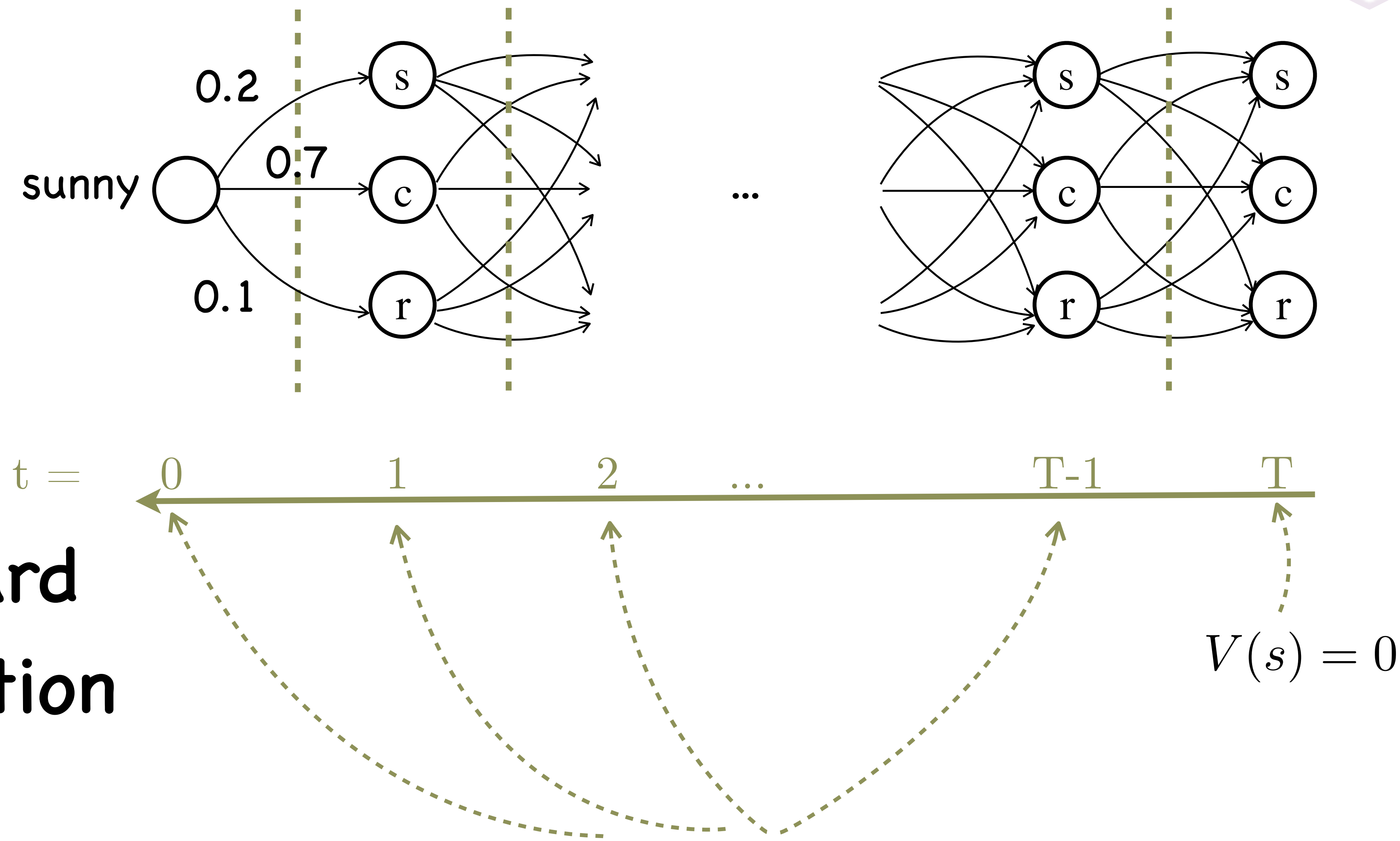
recursive definition:

$$V(\text{sunny}) = P(\text{s}|\text{s})[R(\text{s}) + V(\text{s})] + P(\text{c}|\text{s})[R(\text{c}) + V(\text{c})] + P(\text{r}|\text{s})[R(\text{r}) + V(\text{r})]$$

$$= \sum_{s} P(s|\text{sunny})\big(R(s) + V(s)\big)$$

Markov Reward Process

horizontal view: consider T steps

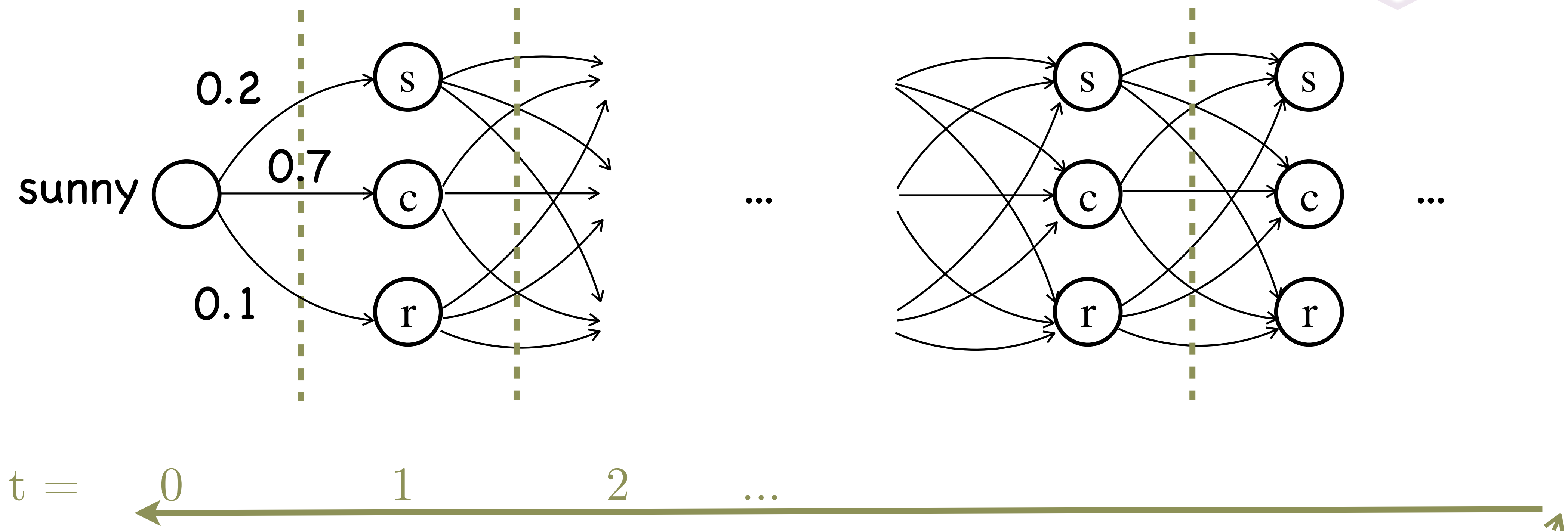


backward calculation

$$V(s) = \sum_{s'} P(s'|s)\big(R(s') + V(s')\big)$$

Markov Reward Process

horizontal view: consider discounted infinite steps



$V(s) = 0$

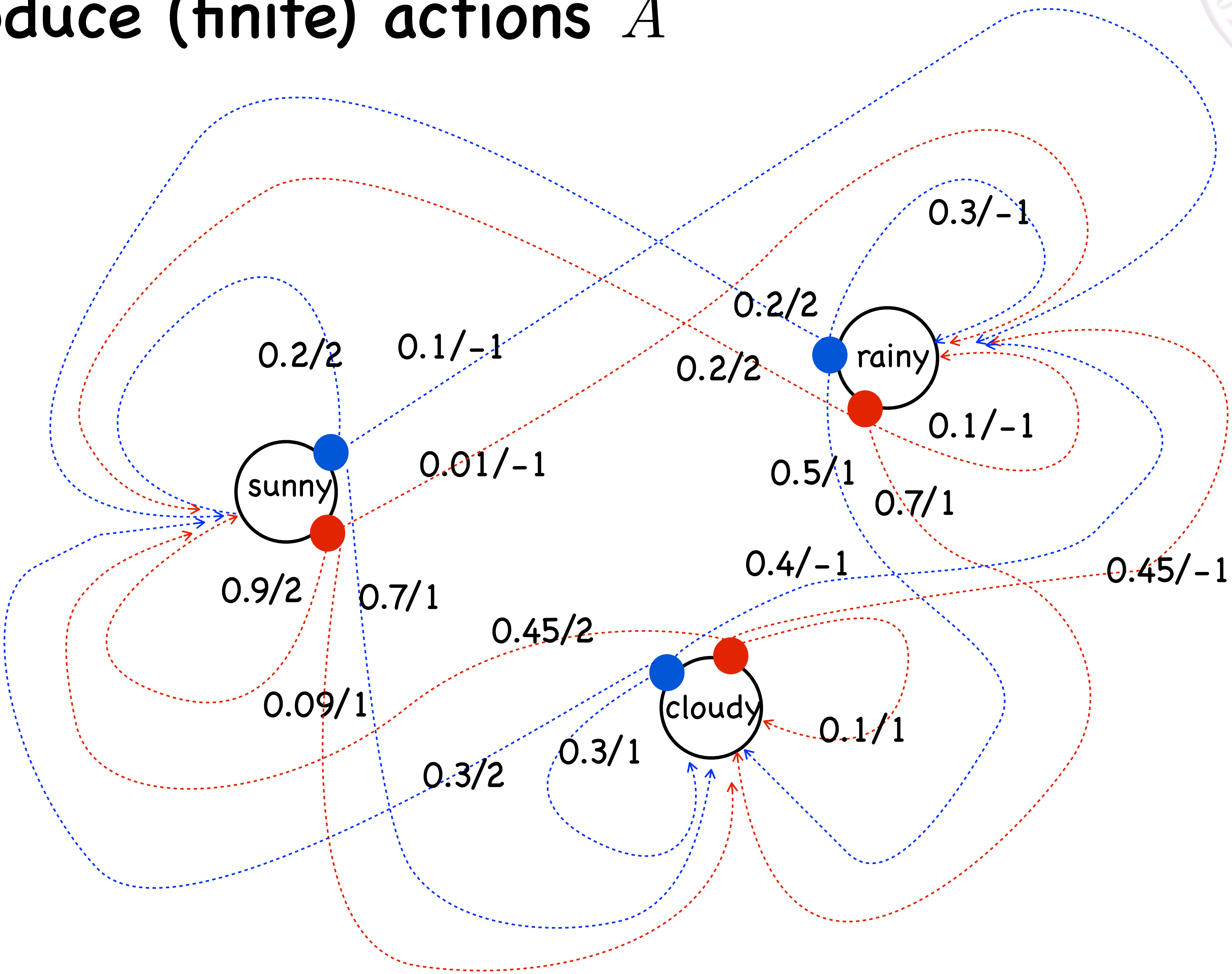
backward calculation

repeat until converges

$$V(s) = \sum_{s'} P(s'|s)\big(R(s') + \gamma V(s')\big)$$

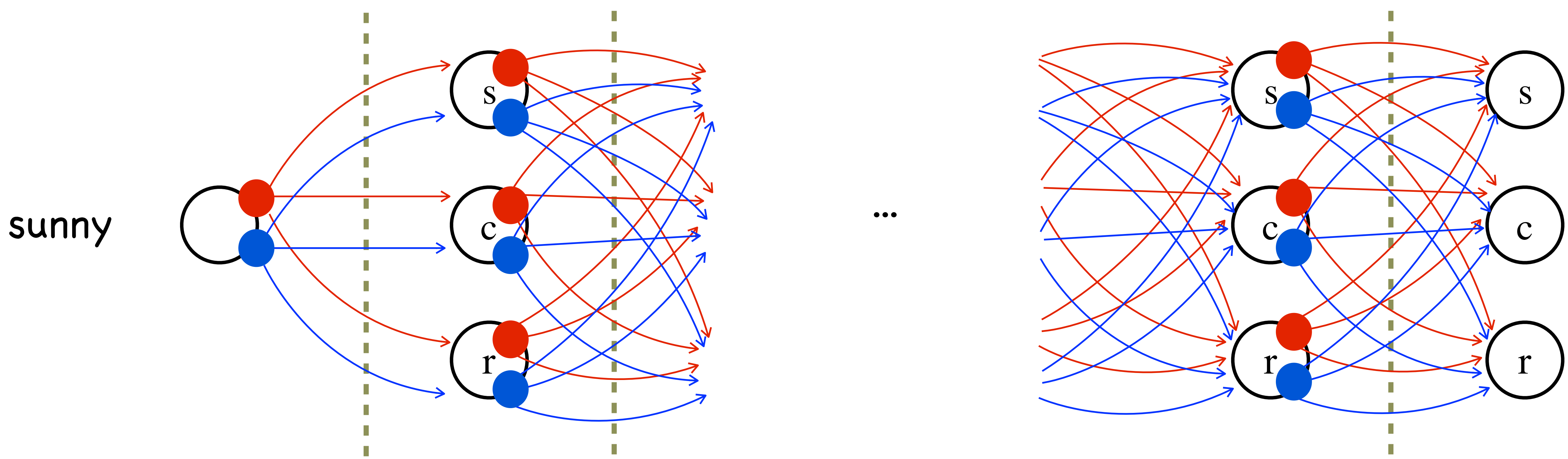
Markov Decision Process

introduce (finite) actions A



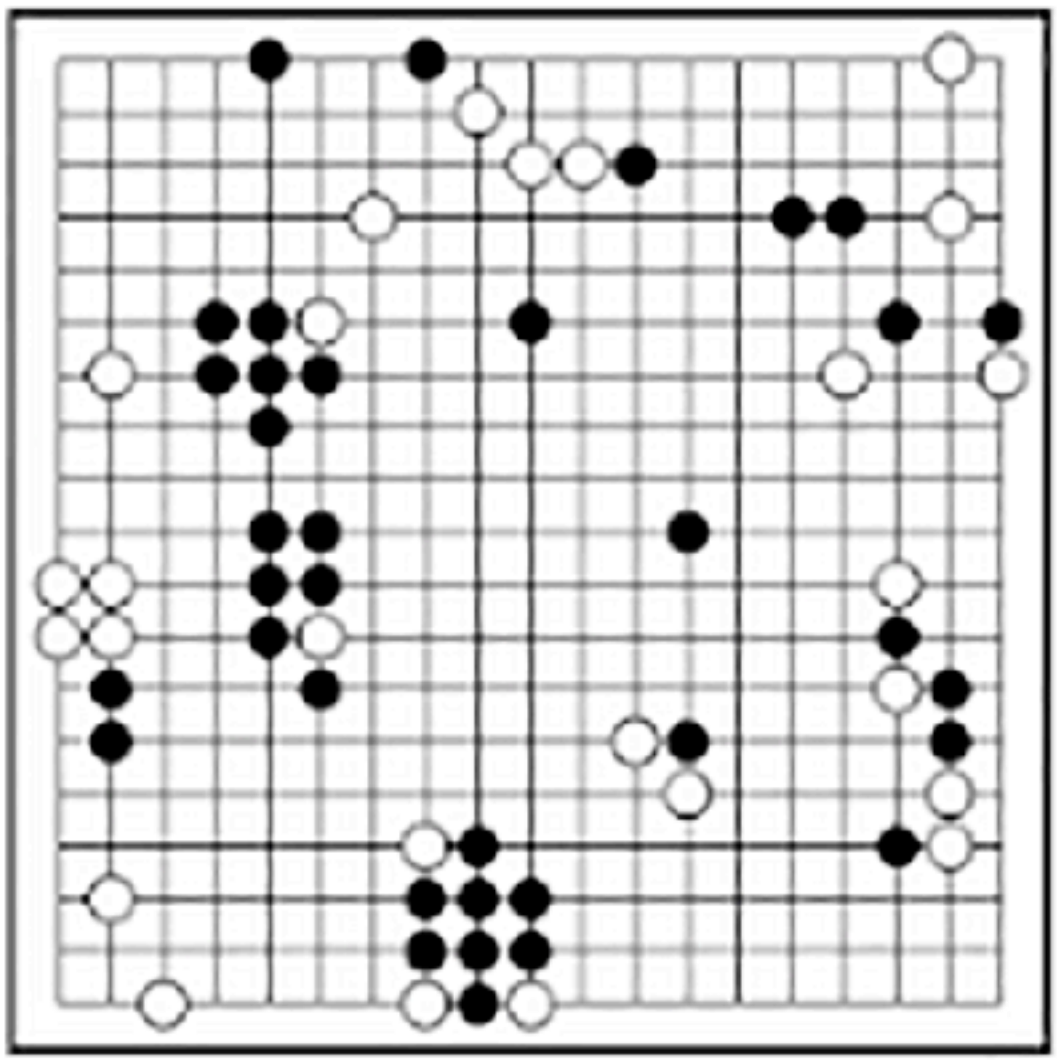
Markov Decision Process

horizontal view

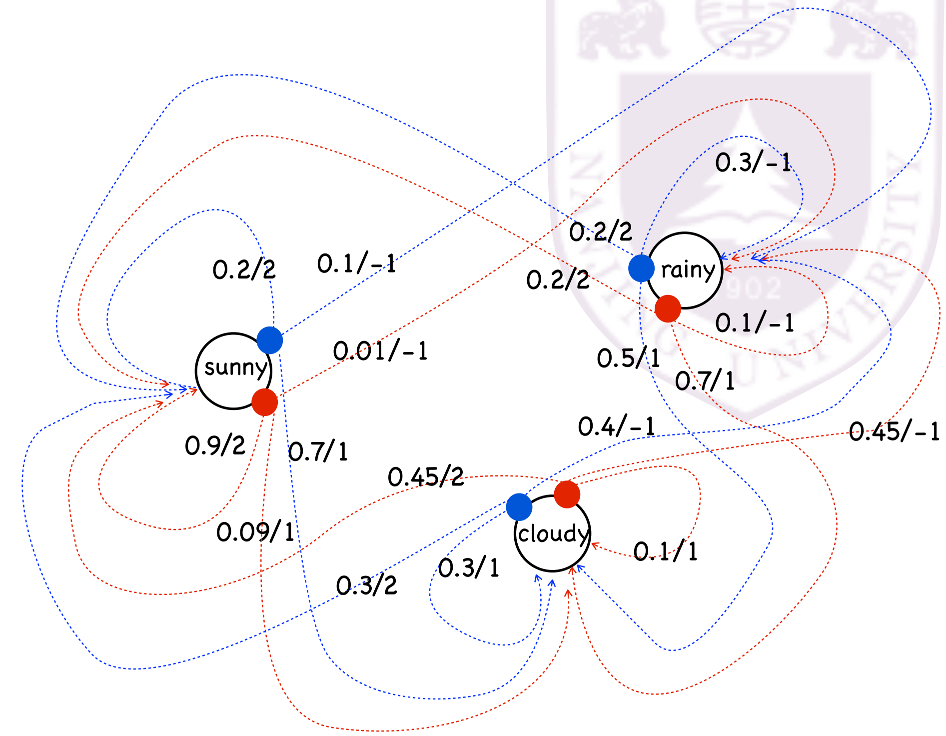


Markov Decision Process

horizontal view of the game of Go



Markov Decision Process



MDP $<S, A, R, P>$ (often with γ)

essential model for RL

but not all of RL

policy

stochastic

$$\pi(a|s) = P(a|s)$$

deterministic

$$\pi(s) = \arg\max_a P(a|s)$$

$|A|^{|S|}$ deterministic policies

tabular representation

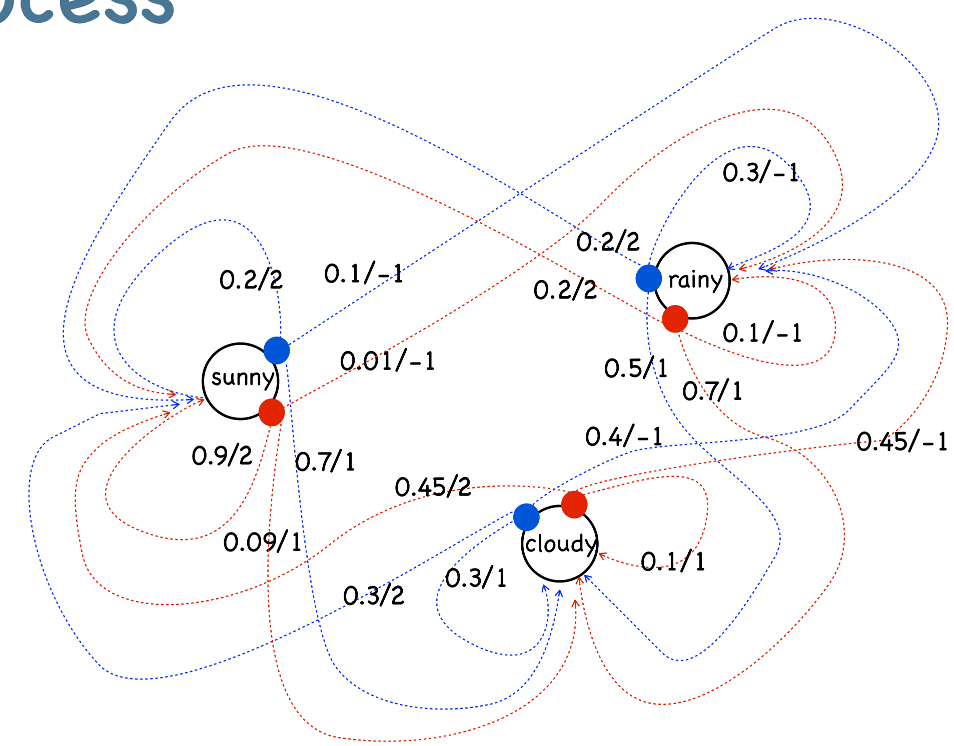
$\pi =$

s	0	0.3
	1	0.7
c	0	0.6
	1	0.4
r	0	0.1
	1	0.9

Expected return

how to calculate the expected total reward of a policy?

similar with the Markov Reward Process



MRP:

$$V(s) = \sum_{s'} P(s'|s)\big(R(s') + V(s')\big)$$

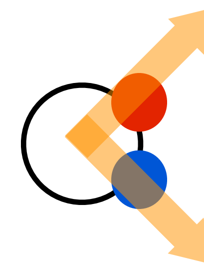
MDP:

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)\big(R(s,a,s') + V^{\pi}(s')\big)$$

expectation over actions
with respect to the policy

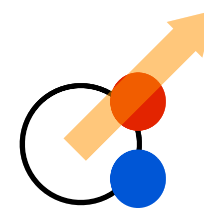
Q-function

state value function



$$V^\pi(s) = E[\sum_{t=1}^{T} r_t | s]$$

state-action value function



$$Q^\pi(s,a) = E[\sum_{t=1}^{T} r_t | s, a] = \sum_{s'} P(s'|s,a)\big(R(s,a,s') + V^\pi(s')\big)$$

consequently,

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

Q-function => policy

Optimality

s	0	0.3
	1	0.7
c	0	0.6
	1	0.4
r	0	0.1
	1	0.9

there exists an optimal policy π^*

$$\forall \pi, \forall s, V^{\pi^*}(s) \geq V^{\pi}(s)$$

optimal value function

$$\forall s, V^*(s) = V^{\pi^*}(s)$$

$$\forall s, \forall a, Q^*(s,a) = Q^{\pi^*}(s,a)$$

Bellman optimality equations

s	0	0.3
	1	0.7
c	0	0.6
	1	0.4
r	0	0.1
	1	0.9

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

from the relation between V and Q

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V^*(s')\big)$$

we have

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma \max_a Q^*(s',a)\big)$$

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V^*(s')\big)$$

the unique fixed point is the optimal value function

Solve optimal policy in MDP

idea:

how is the current policy — policy evaluation

improve the current policy — policy improvement

policy evaluation: backward calculation

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V^\pi(s')\big)$$

policy improvement: from the Bellman optimality equation

$$V(s) \leftarrow \max_a Q^\pi(s,a)$$

Solve optimal policy in MDP

policy improvement: from the Bellman optimality equation

$$V(s) \leftarrow \max_a Q^\pi(s,a)$$

let π' be derived from this update

$$V^\pi(s) \leq Q^\pi(s, \pi'(s))$$
$$= \sum\nolimits_{s'} P(s'|s,\pi'(s))(R(s,\pi'(s),s') + \gamma V^\pi(s'))$$
$$\leq \sum\nolimits_{s'} P(s'|s,\pi'(s))(R(s,\pi'(s),s') + \gamma Q^\pi(s', \pi'(s)))$$
$$= \ldots$$
$$= V^{\pi'}$$

so the policy is improved

Solve optimal policy in MDP

Policy iteration algorithm:

loop until converges

  policy evaluation: calculate V

  policy improvement: choose the action greedily

$$\pi_{t+1}(s) = \arg\max_a Q^{\pi_t}(s,a)$$

converges: $V^{\pi_{t+1}}(s) = V^{\pi_t}(s)$

$$Q^{\pi_{t+1}}(s,a) = \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma \max_a Q^{\pi_t}(s',a)\big)$$

recall the optimal value function about Q

Solve optimal policy in MDP

embed the policy improvement in evaluation

Value iteration algorithm:

$V_0 = 0$

for t=0, 1, …

for all s <- synchronous v.s. asynchronous

$$V_{t+1}(s) = \max_a \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V_t(s)\big)$$

end for

break if $|| V_{t+1} - V_t ||_\infty$ is small enough

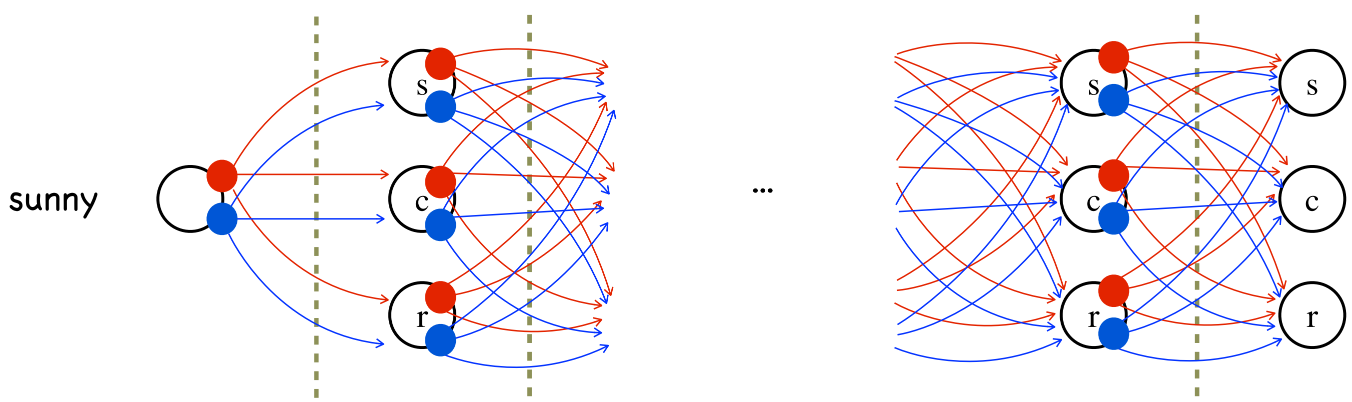
end for

recall the optimal value function about V

Solve optimal policy in MDP

$$Q^{\pi_{t+1}}(s,a) = \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma \max_a Q^{\pi_t}(s',a)\big)$$

$$V_{t+1}(s) = \max_a \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V_t(s')\big)$$



R. E. Bellman

1920-1984

Dynamic programming

Complexity

needs $\Theta(|S|\cdot|A|)$ iterations to converge on deterministic MDP

[O. Madani. Polynomial Value Iteration Algorithms for Deterministic MDPs. UAI'02]

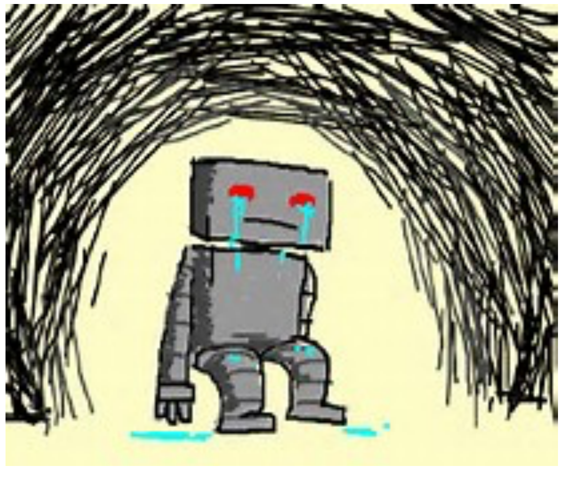
curse of dimensionality: Go board 19x19, $|S|=2.08\times10^{170}$

[https://github.com/tromp/golegal]

from MDP to reinforcement learning

MDP $<S, A, R, P>$

R and P are unknown



Methods

A: learn R and P,
then solve the MDP

model-based

B: learn policy without R or P

model-free

MDP is the model

Model-free RL

explore the environment and learn policy at the same time

Monte-Carlo method

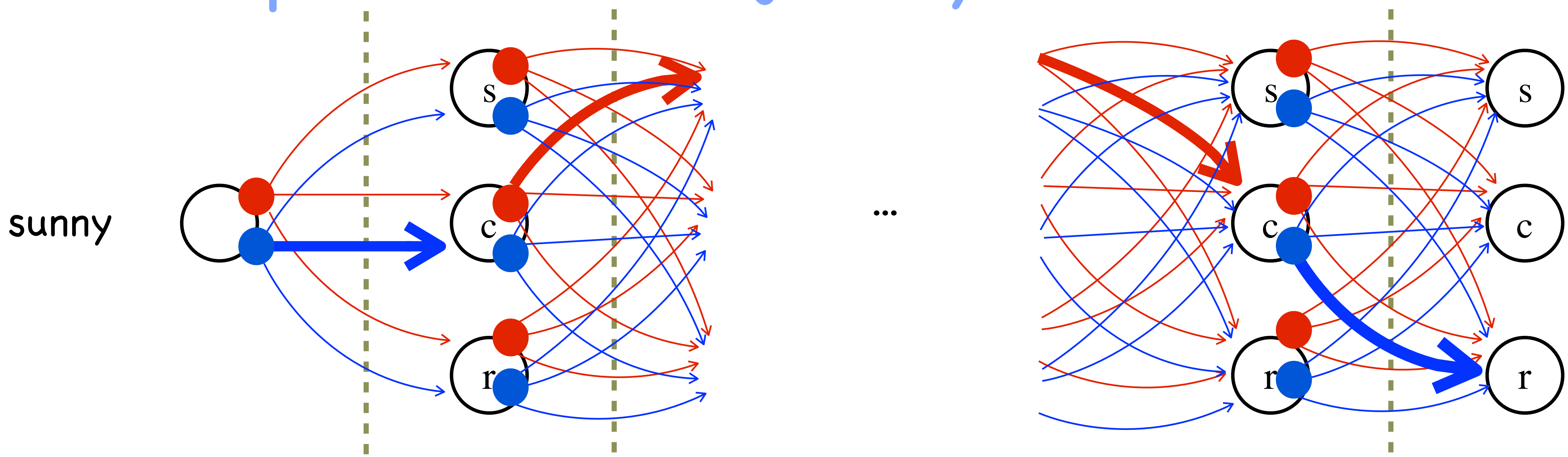
Temporal difference method

Monte Carlo RL - evaluation

Q, not V

expected total reward $Q^\pi(s,a) = E[\sum_{t=1}^{T} r_t | s, a]$

expectation of trajectory-wise rewards



sample trajectory m times,

approximate the expectation by average

$$Q^\pi(s,a) = \frac{1}{m}\sum_{i=1}^{m} R(\tau_i)$$ τ_i is sample by following π after s, a

Monte Carlo RL - evaluation+improvement

$Q_0 = 0$

for i=0, 1, ..., m

  generate trajectory $< s_0, a_0, r_1, s_1, ..., s_T >$

  for t=0, 1, ..., T-1

    R = sum of rewards from t to T

    $Q(s_t, a_t) = (\mathrm{c}(s_t, a_t)\, Q(s_t, a_t) + \mathrm{R}) / (\mathrm{c}(s_t, a_t) + 1)$

    $\mathrm{c}(s_t, a_t){+}{+}$

  end for

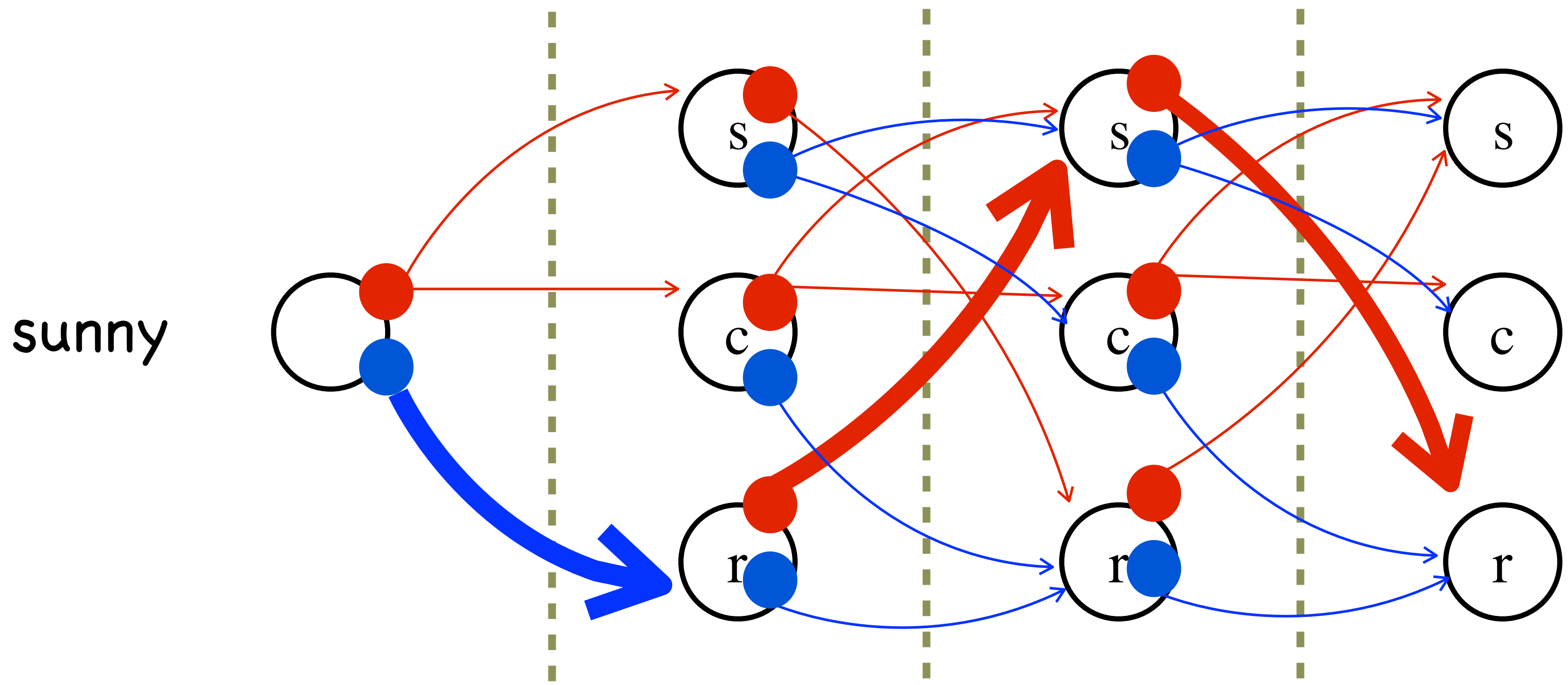
  update policy $\pi(s) = \arg\max_a Q(s, a)$

end for

improvement ?

Monte Carlo RL

problem: what if the policy takes only one path?



cannot improve the policy

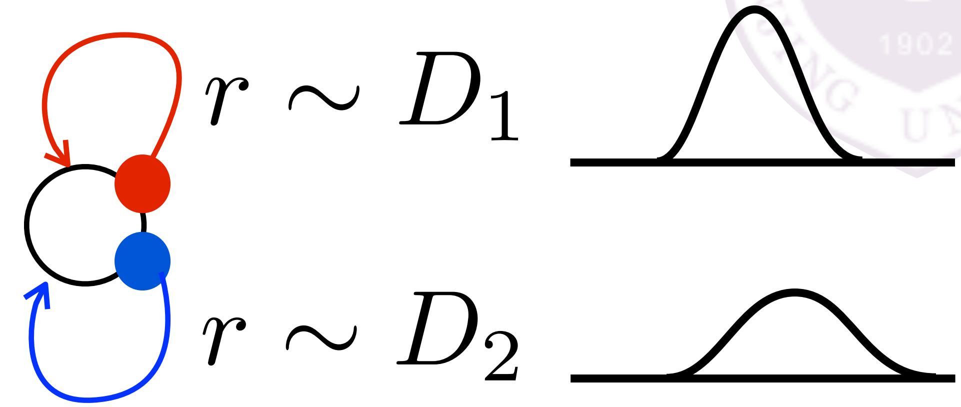
no exploration of the environment

needs exploration !

Exploration methods

one state MDP:

a.k.a. bandit model



maximize the long-term total reward

- exploration only policy: try every action in turn

  waste many trials

- exploitation only policy: try each action once, follow the best action forever

  risk of pick a bad action

balance between exploration and exploitation

Exploration methods

ϵ-greedy:

follow the best action with probability 1-ϵ

choose action randomly with probability ϵ

ϵ should decrease along time

softmax:

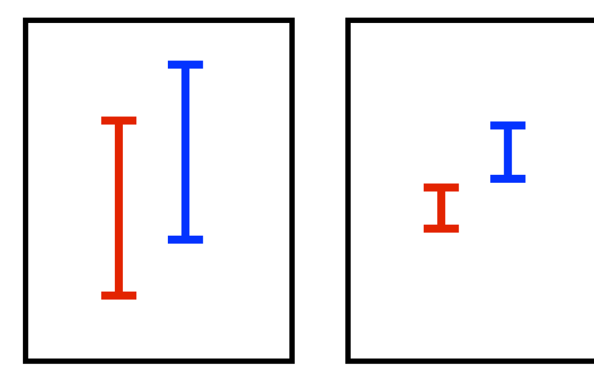
probability according to action quality

$$P(k) = e^{Q(k)/\theta} / \sum_{i=1}^{K} e^{Q(i)/\theta}$$

upper confidence bound (UCB):

choose by action quality + confidence

$$Q(k) + \sqrt{2 \ln n / n_k}$$



Action-level exploration

ϵ-greedy policy:

given a policy π

$$\pi_\epsilon(s) = \begin{cases} \pi(s), \text{with prob. } 1-\epsilon \\ \text{randomly chosen action, with prob. } \epsilon \end{cases}$$

ensure probability of visiting every state > 0

exploration can also be in other levels

Monte Carlo RL

$Q_0 = 0$

for i=0, 1, ..., m

  generate trajectory $< s_0, a_0, r_1, s_1, ..., s_T >$ by π_ϵ

  for t=0, 1, ..., T-1

    R = sum of rewards from t to T

    $Q(s_t, a_t) = (\mathrm{c}(s_t, a_t) Q(s_t, a_t) + \mathrm{R}) / (\mathrm{c}(s_t, a_t) + 1)$

    $\mathrm{c}(s_t, a_t){+}{+}$

  end for

  update policy $\pi(s) = \arg\max_a Q(s, a)$

end for

Monte Carlo RL - on/off-policy

this algorithm evaluates π_ϵ !   on-policy

what if we want to evaluate π ?   off-policy

importance sampling:

$$E[f] = \int_x p(x) f(x) \mathrm{d}x = \int_x q(x) \frac{p(x)}{q(x)} f(x) \mathrm{d}x$$

↓ sample from p ↓ sample from q

$$\frac{1}{m} \sum_{i=1}^{m} f(x) \qquad \frac{1}{m} \sum_{i=1}^{m} \frac{p(x)}{q(x)} f(x)$$

Monte Carlo RL -- off-policy

$Q_0 = 0$

for i=0, 1, ..., m

generate trajectory $<s_0, a_0, r_1, s_1, \ldots, s_T>$ by π_ϵ

for t=0, 1, ..., T-1

R = sum of rewards from t to $T \times \prod_{i=t+1}^{T-1} \frac{\pi(x_i, a_i)}{p_i}$

$Q(s_t, a_t) = (\mathrm{c}(s_t, a_t)\, Q(s_t, a_t) + \mathrm{R}) / (\mathrm{c}(s_t, a_t) + 1)$

$\mathrm{c}(s_t, a_t)$++

end for

update policy $\pi(s) = \arg\max_a Q(s, a)$

end for

$$p_i = \begin{cases} 1 - \epsilon + \epsilon/|A|, a_i = \pi(s_i), \\ \epsilon/|A|, a_i \neq \pi(s_i) \end{cases}$$

Monte Carlo RL

summary

Monte Carlo evaluation:
approximate expectation by sample average

action-level exploration

on-policy, off-policy: importance sampling

Monte Carlo RL:
evaluation + action-level exploration + policy improvement (on/off-policy)

Incremental mean

$Q(s_t, a_t) = (\mathrm{c}(s_t, a_t)\, Q(s_t, a_t) + \mathrm{R}) / (\mathrm{c}(s_t, a_t) + 1)$

$$\mu_t = \frac{1}{t}\sum_{i=1}^{t} x_i = \frac{1}{t}(x_t + \sum_{i=1}^{t-1} x_i) = \frac{1}{t}(x_t + (t-1)\mu_{t-1})$$

$$= \mu_{t-1} + \frac{1}{t}(x_t - \mu_{t-1})$$

In general, $\mu_t = \mu_{t-1} + \alpha(x_t - \mu_{t-1})$

Monte-Carlo update:

$$Q(s_t, a_t) \Leftarrow Q(s_t, a_t) + \alpha(\underbrace{R - Q(s_t, a_t)}_{\text{MC error}})$$

Temporal-Difference Learning - evaluation

update policy online learn as you go

TD Evaluation

Monte-Carlo update:

$$Q(s_t, a_t) \Leftarrow Q(s_t, a_t) + \alpha(\underbrace{R - Q(s_t, a_t)}_{\text{MC error}})$$

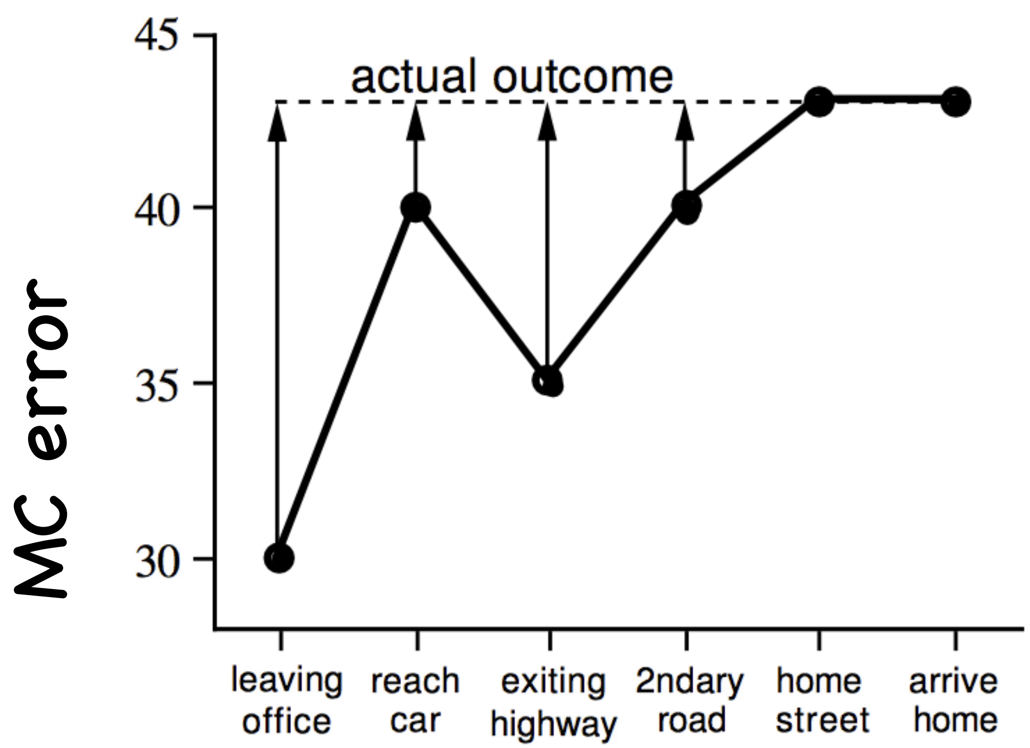
TD update:

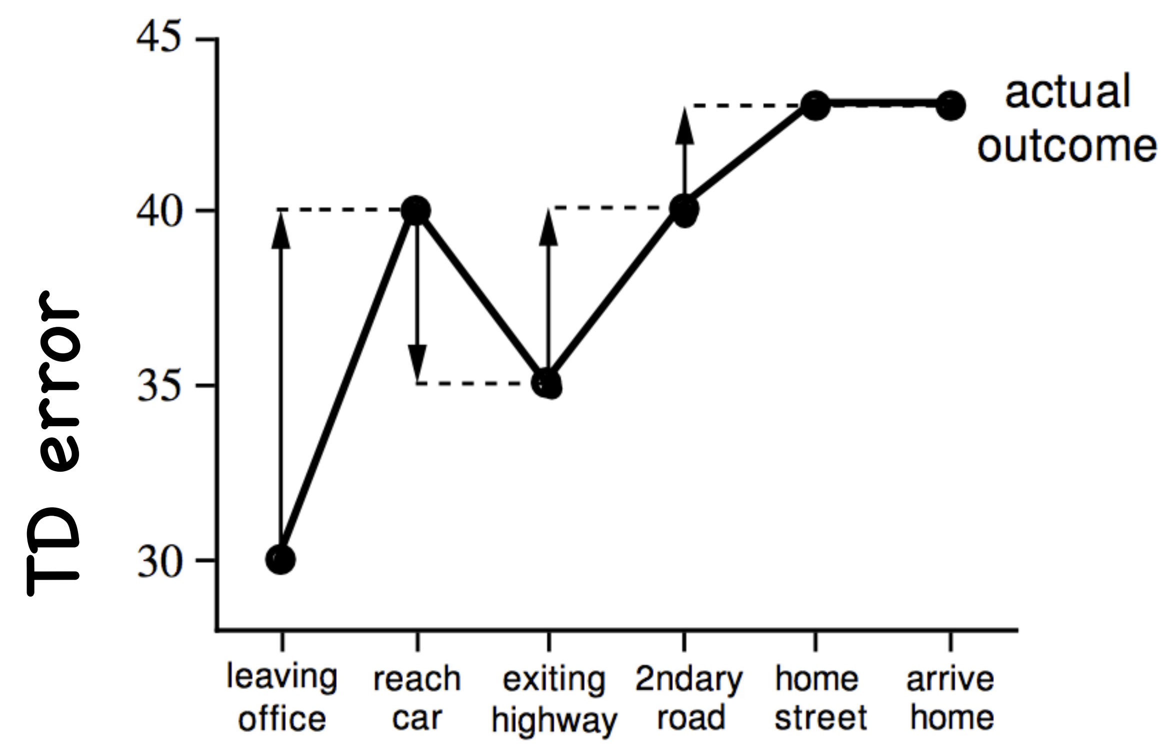
$$Q(s_t, a_t)$$

$$\Leftarrow Q(s_t, a_t) + \alpha(\underbrace{r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)}_{\text{TD error}})$$

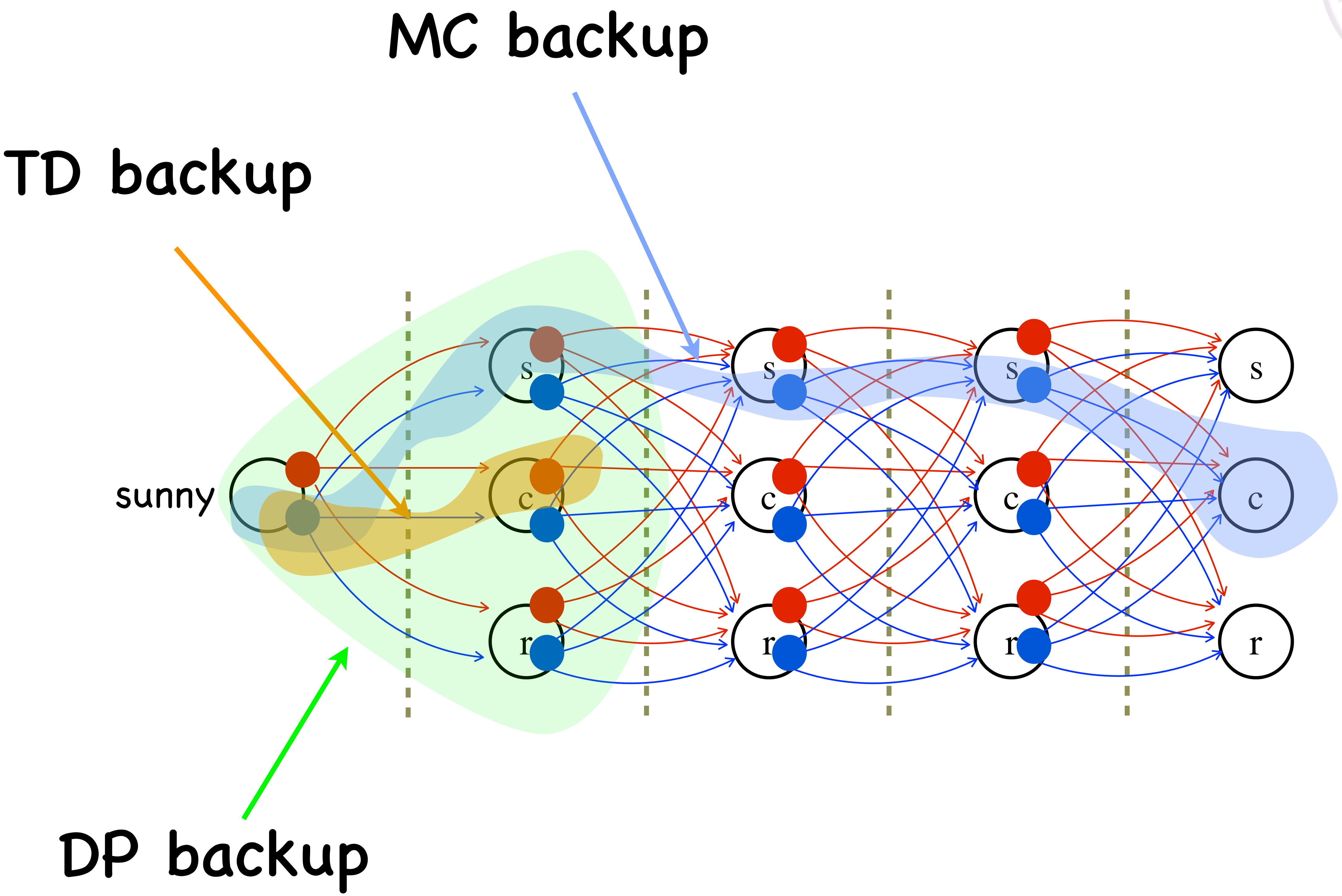
Temporal-Difference Learning - example

state	elapsed time	predicted remaining time	predicted total time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43





Temporal-Difference Learning - backups



SARSA

On-policy TD control

$Q_0 = 0$, initial state

for i=0, 1, ...

$a = \pi_\epsilon(s)$

s', r = do action a

$a' = \pi_\epsilon(s')$

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

$\pi(s) = \arg\max_a Q(s,a)$

$s = s'$

end for

Q-learning

Off-policy TD control

$Q_0 = 0$, initial state

for i=0, 1, ...

$a = \pi_\epsilon(s)$

s', r = do action a

$a' = \pi(s')$

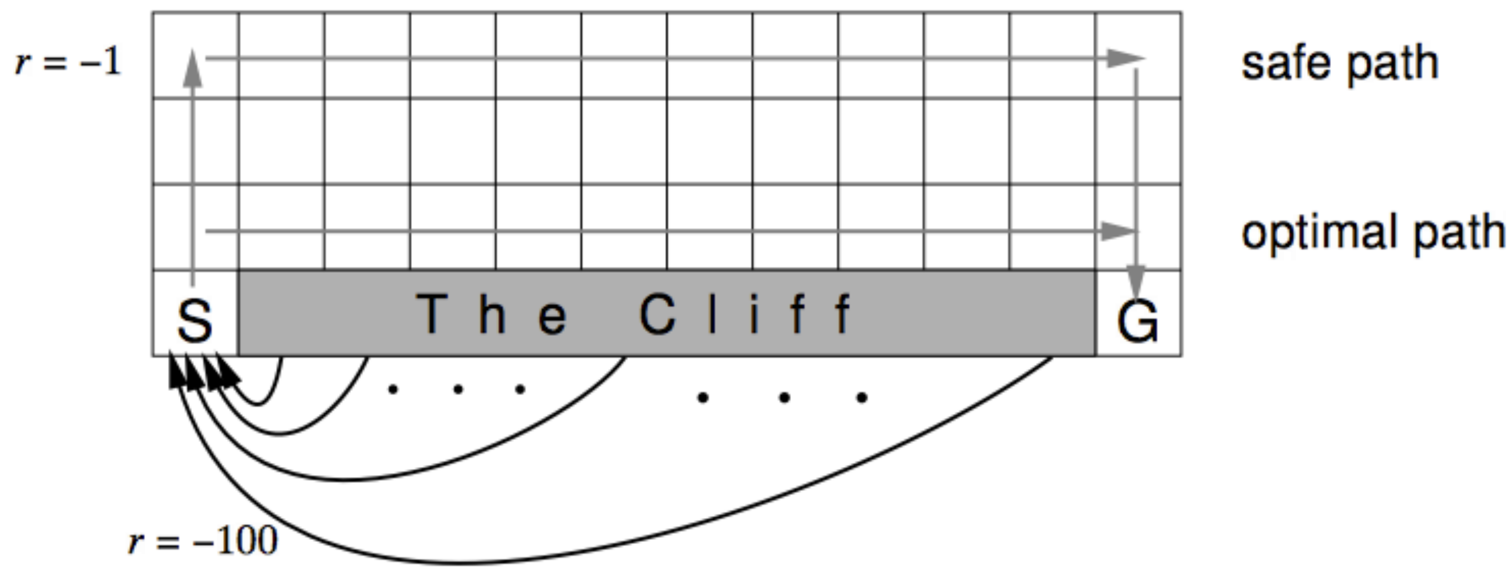
$Q(s,a) \mathrel{+}= \alpha(r + \gamma Q(s', a') - Q(s,a))$

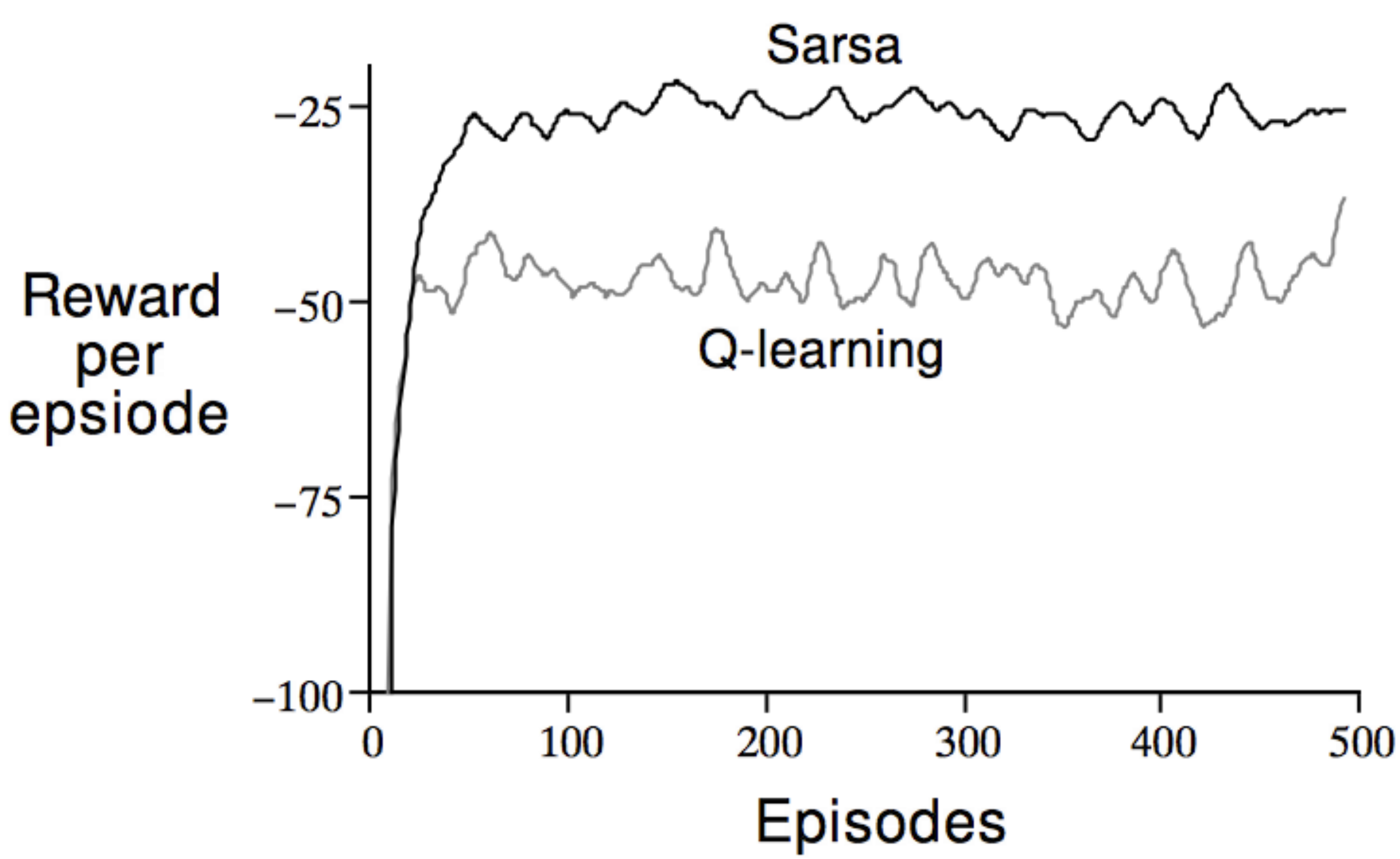
$\pi(s) = \arg\max_a Q(s,a)$

$s = s'$

end for

SARSA v.s. Q-learning



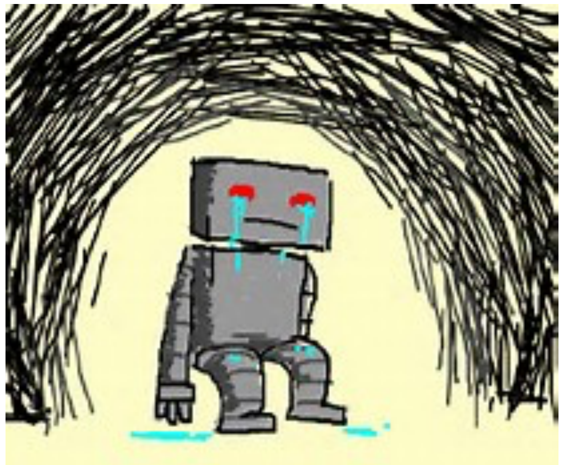


we can do RL now! … in (small) discrete state space

RL in continuous state space

MDP $<S, A, R, P>$

S (and A) is in $\mathbb{R}^n$



Value function approximation

modern RL

tabular representation

$\pi =$

s	0	0.3
	1	0.7
c	0	0.6
	1	0.4
r	0	0.1
	1	0.9

very powerful representation

can be all possible policies !

linear function approx.

$$\hat{V}(s) = w^\top \phi(s)$$

$$\hat{Q}(s,a) = w^\top \phi(s,a)$$

$$\hat{Q}(s,a_i) = w_i^\top \phi(s)$$

ϕ is a feature mapping

w is the parameter vector

may not represent all policies !

Value function approximation

to approximate Q and V value function

least square approximation

$$J(w) = E_{s\sim\pi}[\left(Q^{\pi}(s,a) - \hat{Q}(s,a)\right)^2]$$

online environment: stochastic gradient on single sample

$$\Delta w_t = \theta(Q^{\pi}(s_t,a_t) - \hat{Q}(s_t,a_t))\nabla_w \hat{Q}(s_t,a_t)$$

replace

Recall the errors:

MC update: $Q(s_t,a_t) += \alpha(\underline{R} - \underline{Q(s_t,a_t)})$

TD update: $Q(s_t,a_t) += \alpha(\underline{r_{t+1} + \gamma Q(s_{t+1},a_{t+1})} - \underline{Q(s_t,a_t)})$

target   model

Value function approximation

MC update:

$$\Delta w_t = \theta(R - \hat{Q}(s_t, a_t))\nabla_w \hat{Q}(s_t, a_t)$$

TD update:

$$\Delta w_t = \theta(r_{t+1} + \gamma\hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t))\nabla_w \hat{Q}(s_t, a_t)$$

eligibility traces

$$E_t = \gamma\lambda E_{t-1} + \nabla_w \hat{Q}(s_t, a_t)$$

Q-learning with function approximation

$w = 0$, initial state

for i=0, 1, ...

$a = \pi_\epsilon(s)$

s', r = do action a

a' $= \pi(s')$

$w+ = \theta(r + \gamma\hat{Q}(s,a) - \hat{Q}(s,a))\nabla_w\hat{Q}(s_t,a_t)$

$\pi(s) = \arg\max_a \hat{Q}(s,a)$

$s = s$'

end for

Approximation model

Linear approximation $\hat{Q}(s,a) = w^\top \phi(s,a)$

$$\nabla_w \hat{Q}(s,a) = \phi(s,a)$$

coarse coding: raw features

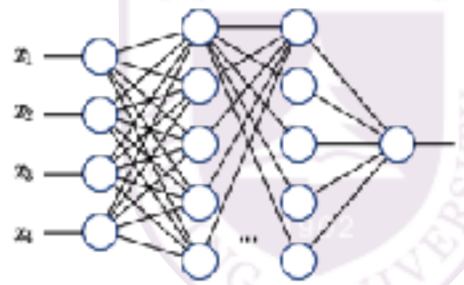
discretization: tide with indicator features

kernelization:

$$\hat{Q}(s,a) = \sum_{i=1}^{m} w_i K((s,a),(s_i,a_i))$$

(s_i, a_i) can be randomly sampled

Approximation model



Nonlinear model approximation $\hat{Q}(s,a) = f(s,a)$

neural network: differentiable model

recall the TD update:

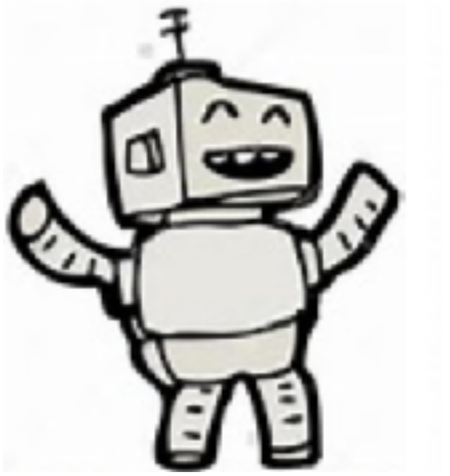
$$\Delta w_t = \theta(r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t)) \underline{\nabla_w \hat{Q}(s_t, a_t)}$$

follow the BP rule to pass the gradient

RL in continuous state space

Deep Reinforcement Learning

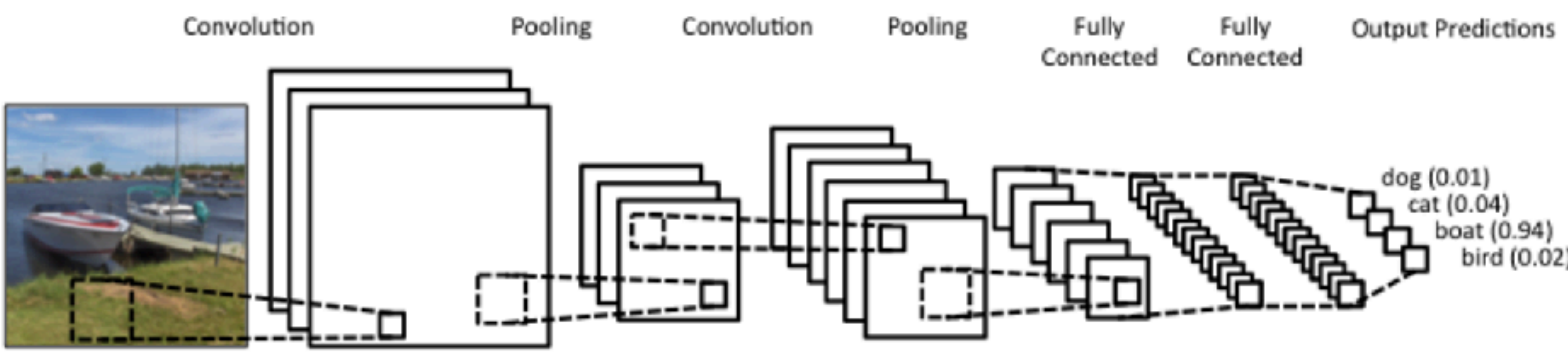
function approximation by
deep neural networks



Convolutional neural networks

a powerful neural network architecture for image analysis

- differentiable
- require a lot of samples to train



Deep Q-Network

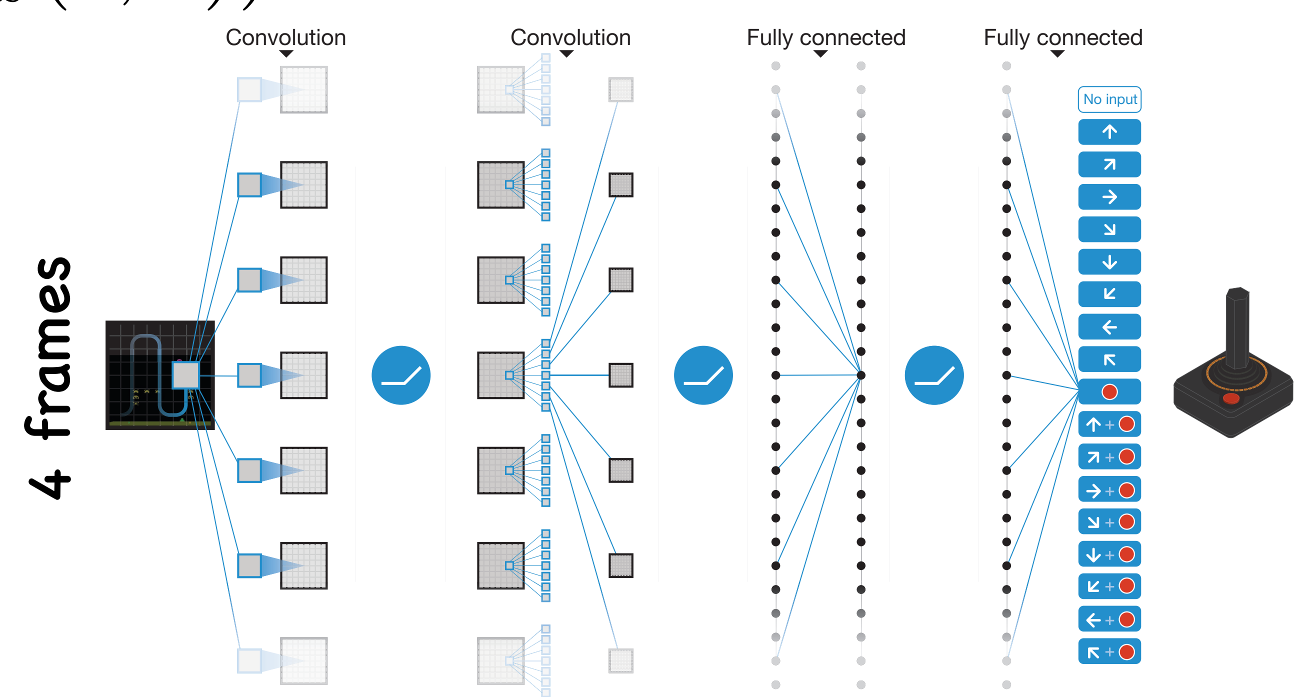
DQN

- using ϵ-greedy policy
- store 1million recent history (s,a,r,s’) in replay memory D
- sample a mini-batch (32) from D
- calculate Q-learning target $\tilde{Q}$
- update CNN by minimizing the Bellman error (delayed update)

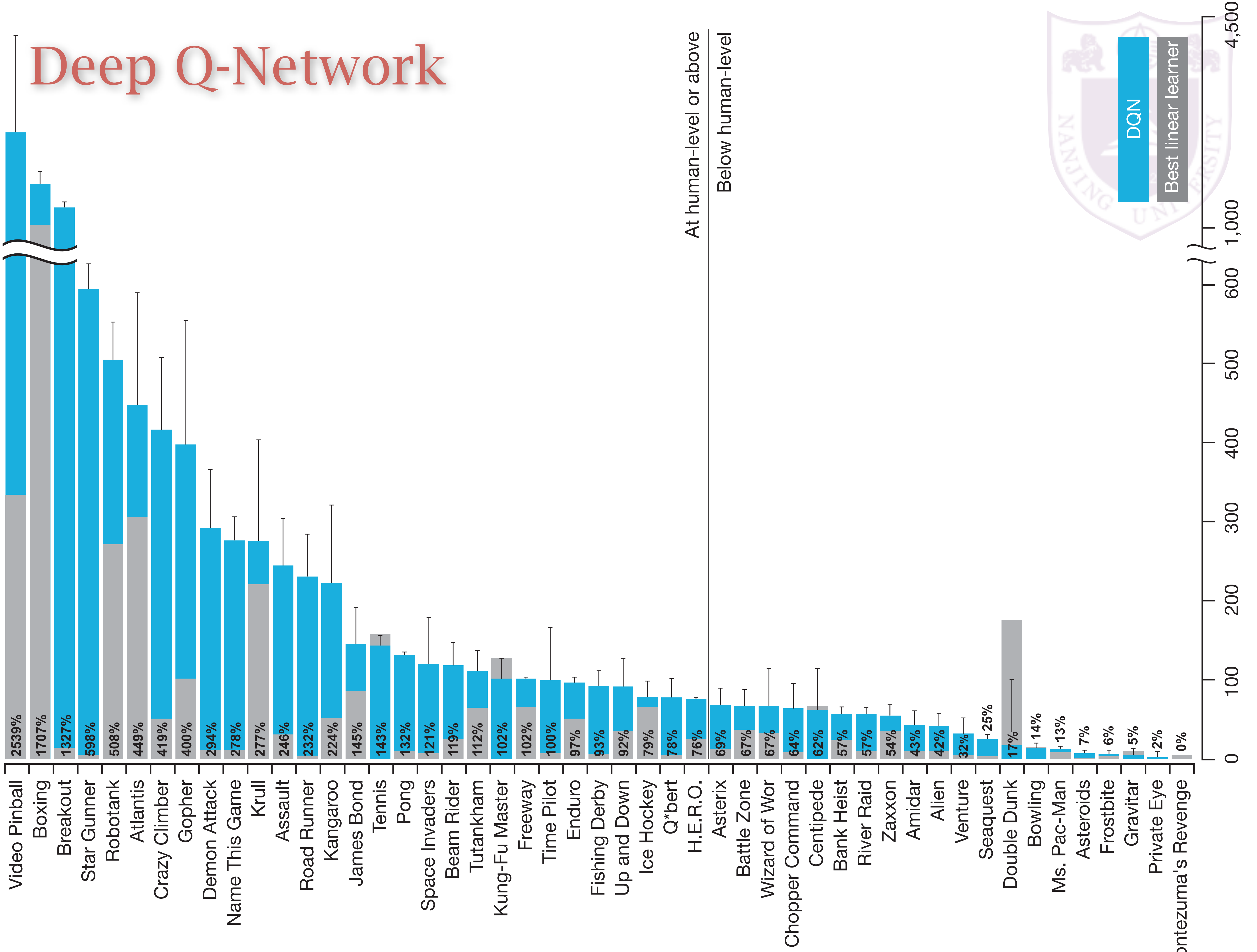
$$\sum (r + \gamma \max_{a'} \tilde{Q}(s', a') - Q_w(s, a))^2$$

DQN on Atari

learn to play from pixels



Deep Q-Network



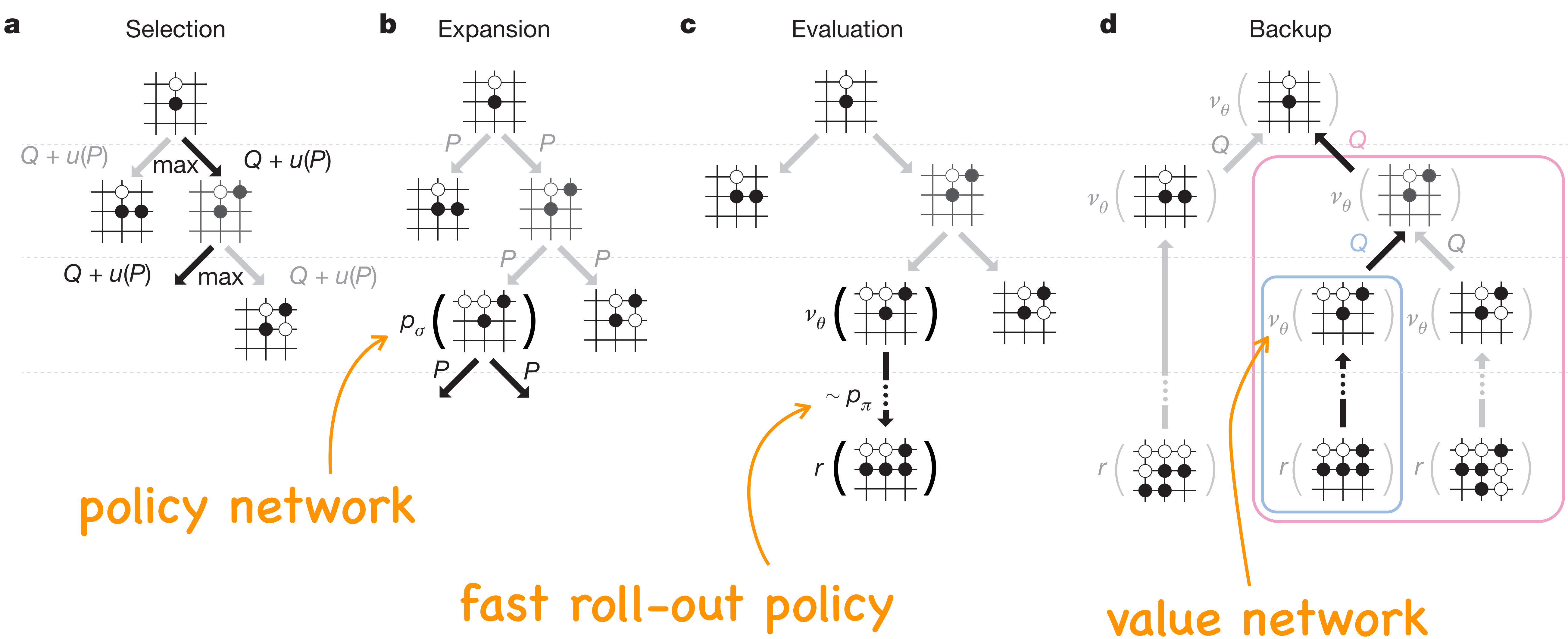
Deep Q-Network

effectiveness

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

AlphaGo

A combination of tree search, deep neural networks and reinforcement learning



fast roll-out policy:

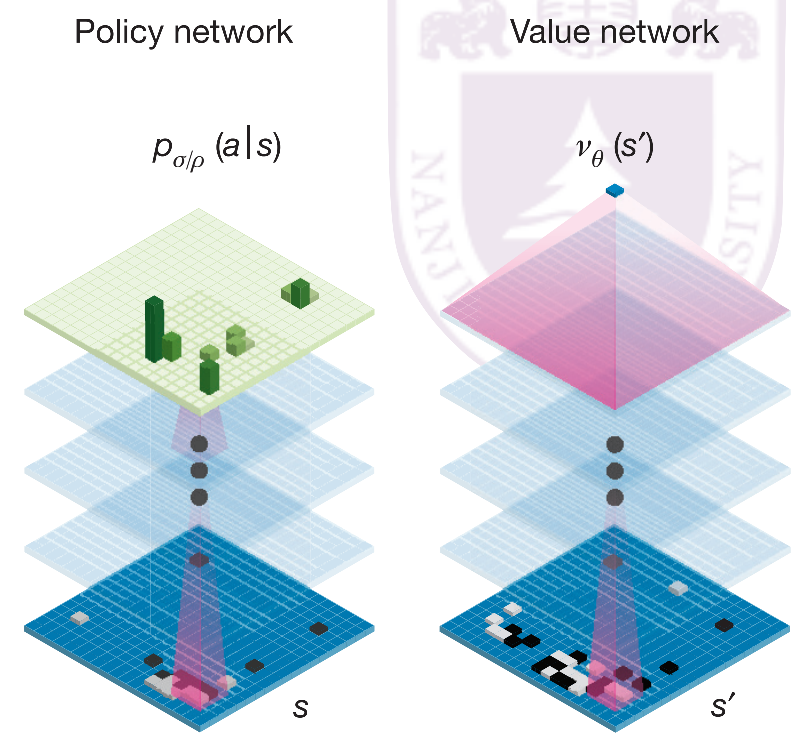
supervised learning from human v.s. human data

Feature	# of patterns	Description
Response	1	Whether move matches one or more response pattern features
Save atari	1	Move saves stone(s) from capture
Neighbour	8	Move is 8-connected to previous move
Nakade	8192	Move matches a *nakade* pattern at captured stone
Response pattern	32207	Move matches 12-point diamond pattern near previous move
Non-response pattern	69338	Move matches 3×3 pattern around move
Self-atari	1	Move allows stones to be captured
Last move distance	34	Manhattan distance to previous two moves
Non-response pattern	32207	Move matches 12-point diamond pattern centred around move

AlphaGo

policy network: a CNN output π(s,a)

value network: a CNN output V(s)



Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

policy network: initialization

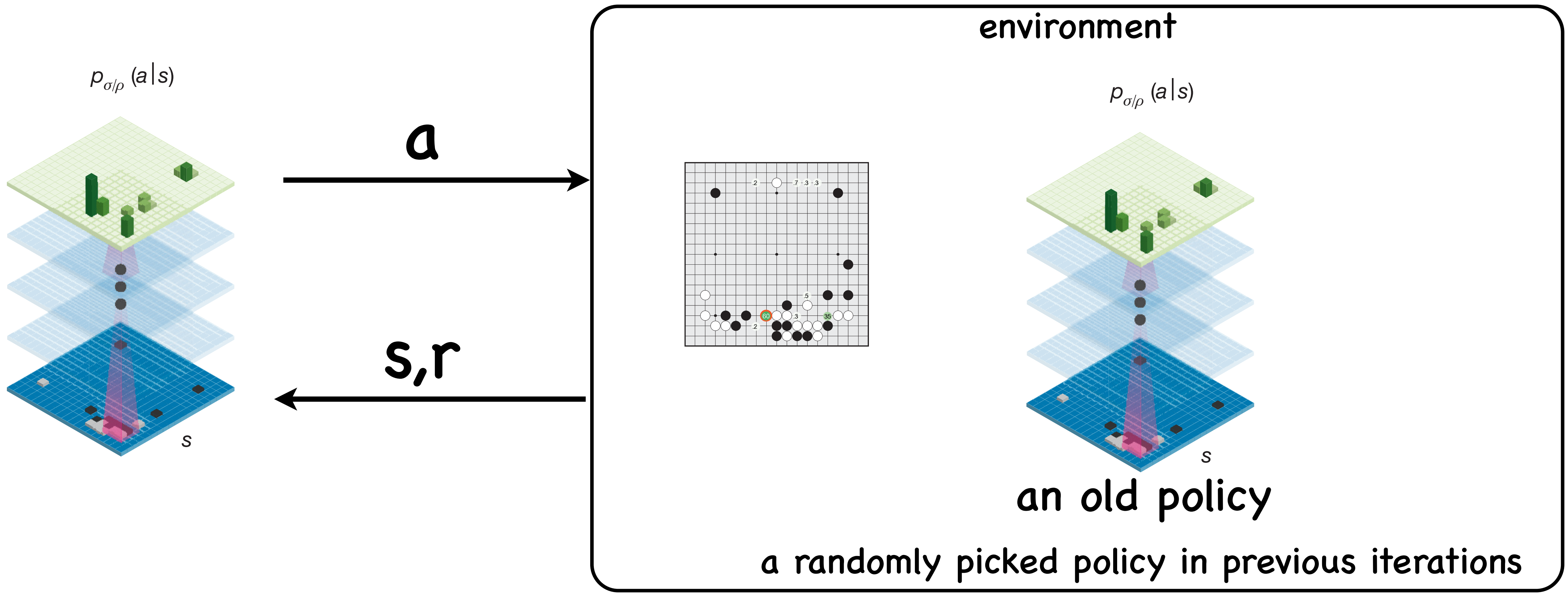
supervised learning from human v.s. human data

Architecture			Evaluation				
Filters	Symmetries	Features	Test accuracy %	Train accuracy %	Raw net wins %	*AlphaGo* wins %	Forward time (ms)
128	1	48	54.6	57.0	36	53	2.8
192	1	48	55.4	58.0	50	50	4.8
256	1	48	55.9	59.1	67	55	7.1
256	2	48	56.5	59.8	67	38	13.9
256	4	48	56.9	60.2	69	14	27.6
256	8	48	57.0	60.4	69	5	55.3
192	1	4	47.6	51.4	25	15	4.8
192	1	12	54.7	57.1	30	34	4.8
192	1	20	54.7	57.2	38	40	4.8
192	8	4	49.2	53.2	24	2	36.8
192	8	12	55.7	58.3	32	3	36.8
192	8	20	55.8	58.4	42	3	36.8

AlphaGo

policy network: further improvement

reinforcement learning

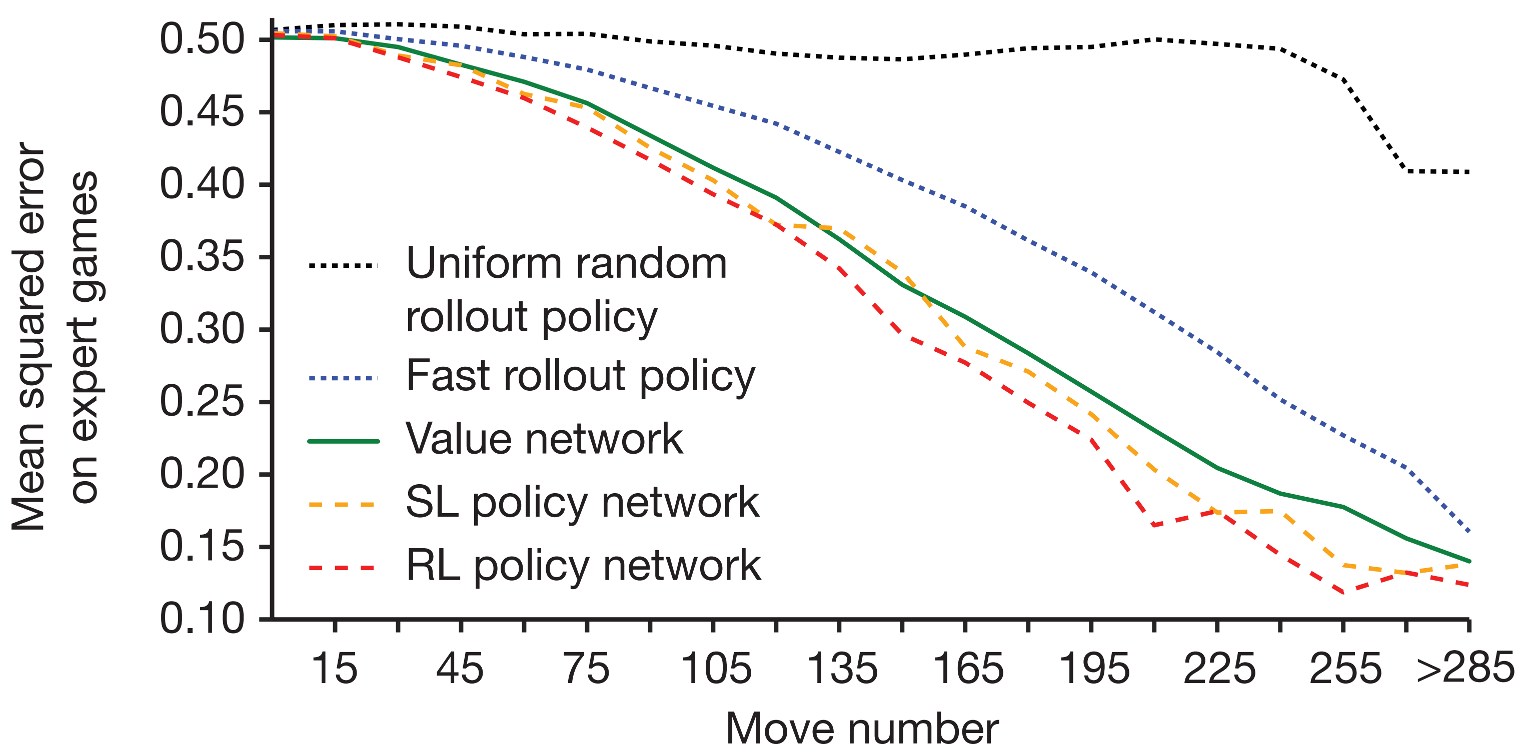


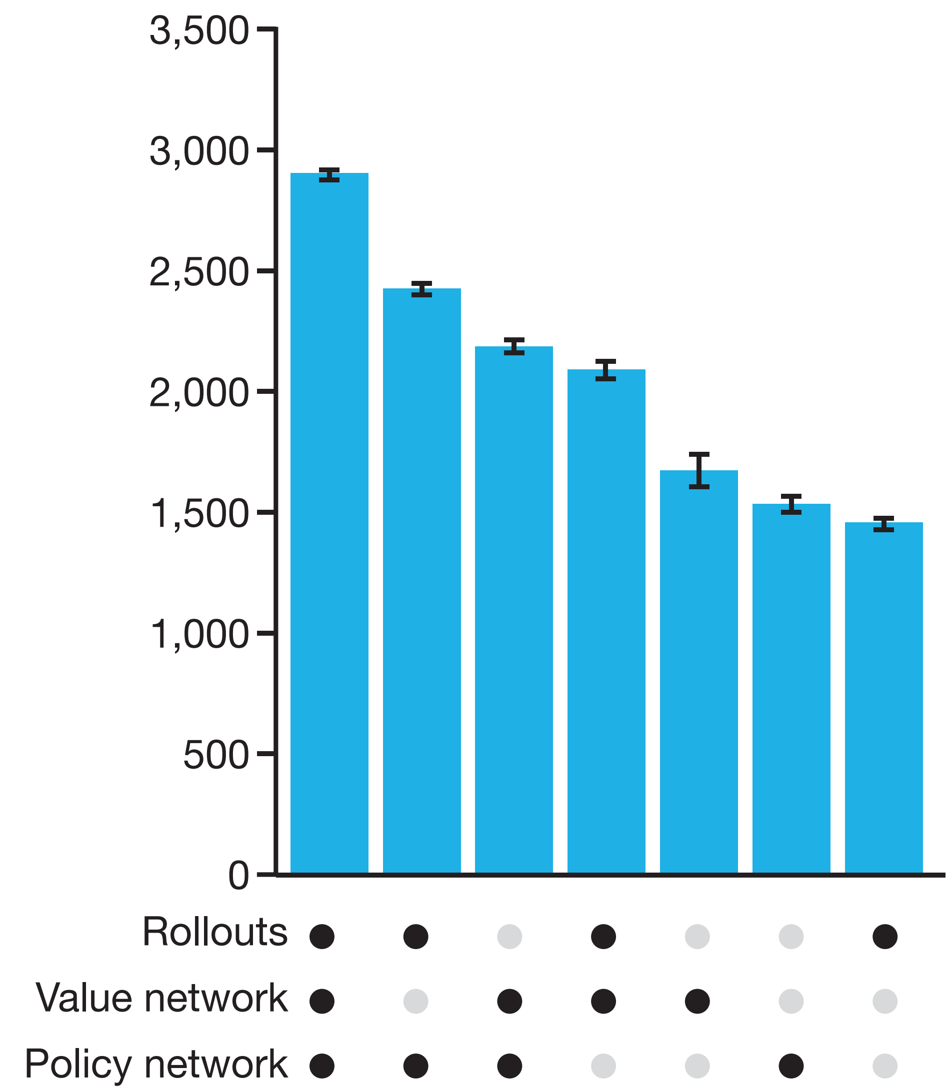
a.k.a. self-play

reward:
+1 -- win at terminate state
-1 -- loss at terminate state

AlphaGo

value network: supervised learning from RL data






《星际争霸II》
SIMPLE TEST MAP 64X64
本地玩家
加载完成
电脑难度
加载完成