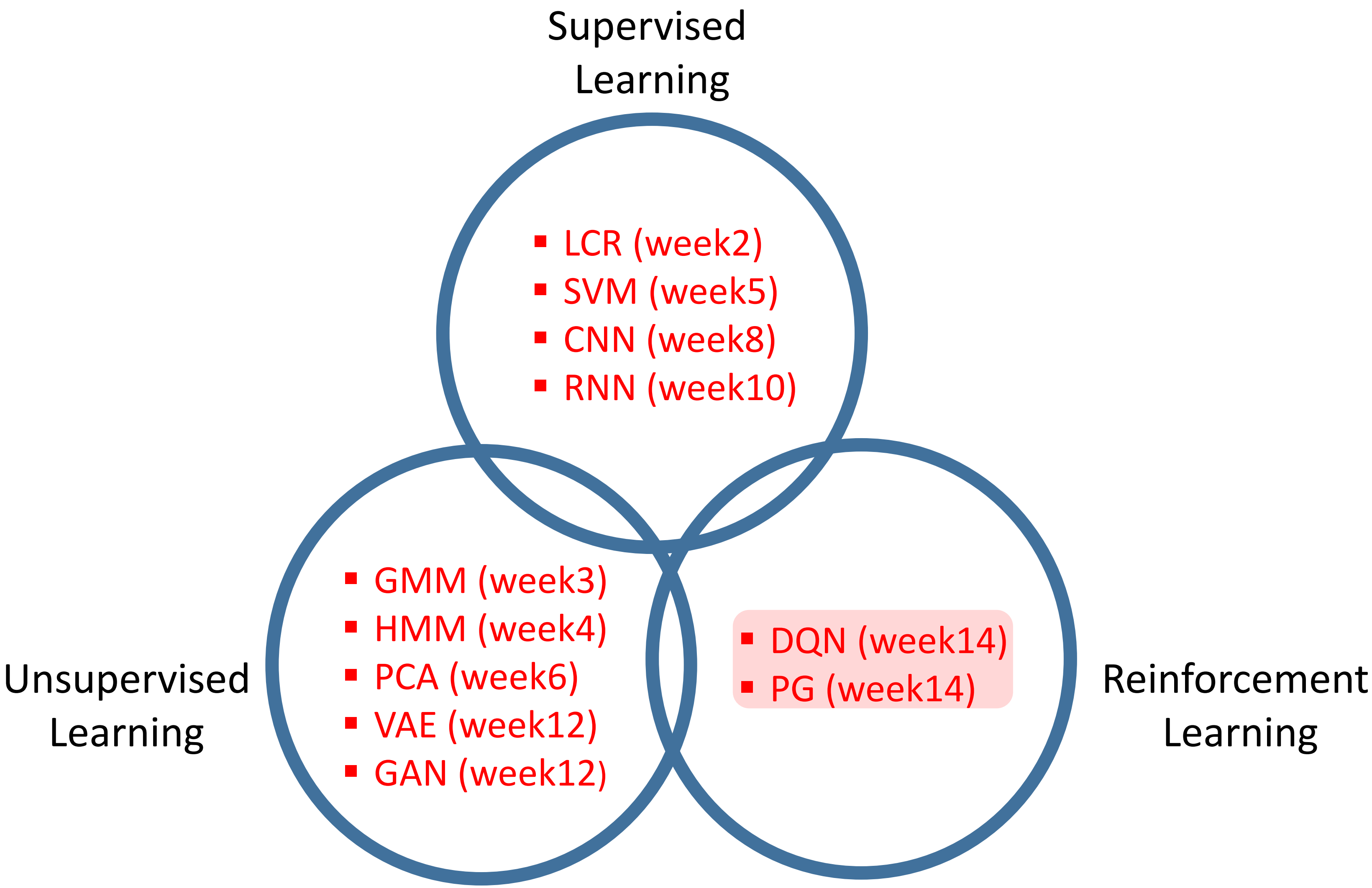
Practical Machine Learning

Lecture 14
Reinforcement Learning (RL): Deep Q Networks (DQN) and Policy Gradient (PG: AC/A3C)

Dr. Suyong Eum

OSAKA UNIVERSITY

Where we are



Supervised Learning

- LCR (week2)
- SVM (week5)
- CNN (week8)
- RNN (week10)

Unsupervised Learning

- GMM (week3)
- HMM (week4)
- PCA (week6)
- VAE (week12)
- GAN (week12)

Reinforcement Learning

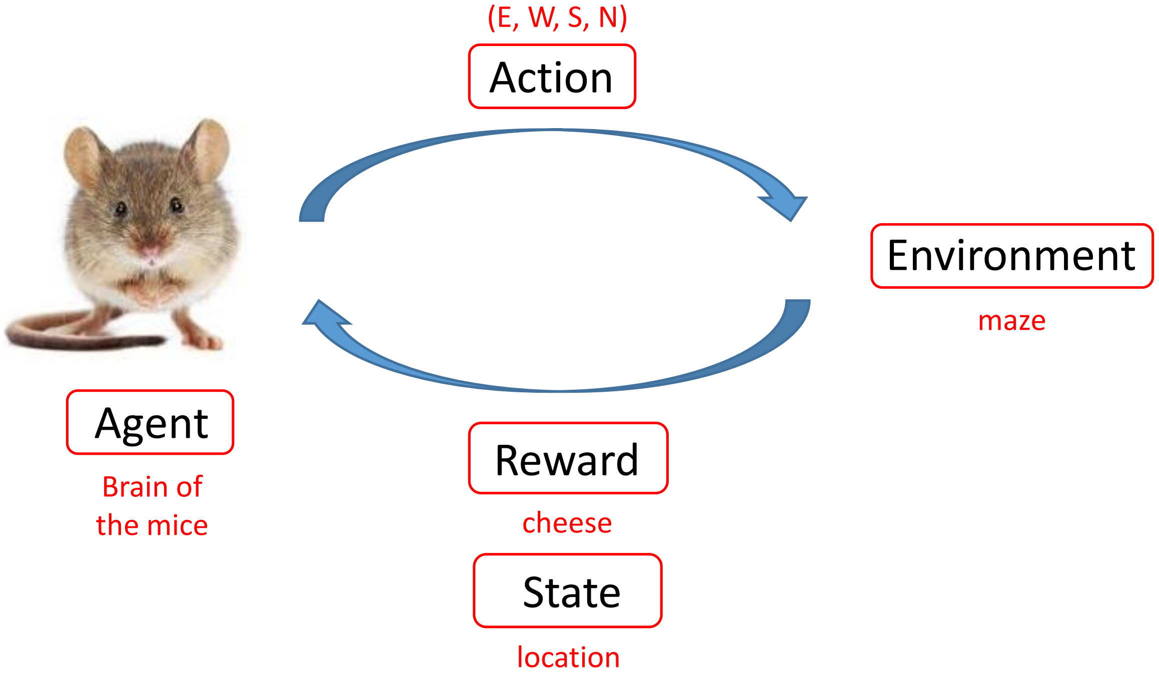
- DQN (week14)
- PG (week14)

You are going to learn

- ❑ What Reinforcement Learning (RL) is
- ❑ Deep Q Network (DQN)
- ❑ Policy Gradient (PG)
  - Actor Critic (AC)
  - Asynchronous Advantage Actor Critic (A3C)

Big picture: Reinforcement Learning (RL)

- Learning how to take actions in an environment so as to maximize future cumulative reward.



(E, W, S, N)

Action

Environment

maze

Agent

Brain of the mice

Reward

cheese

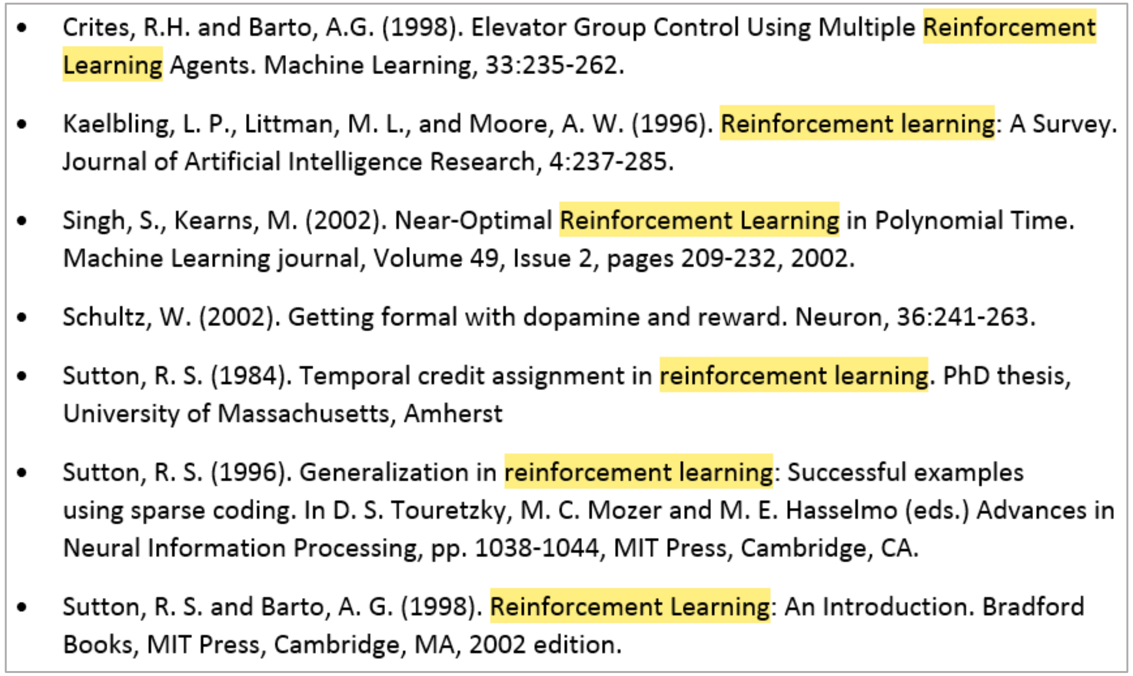
State

location

Reinforcement Learning (RL) has been around for a long time

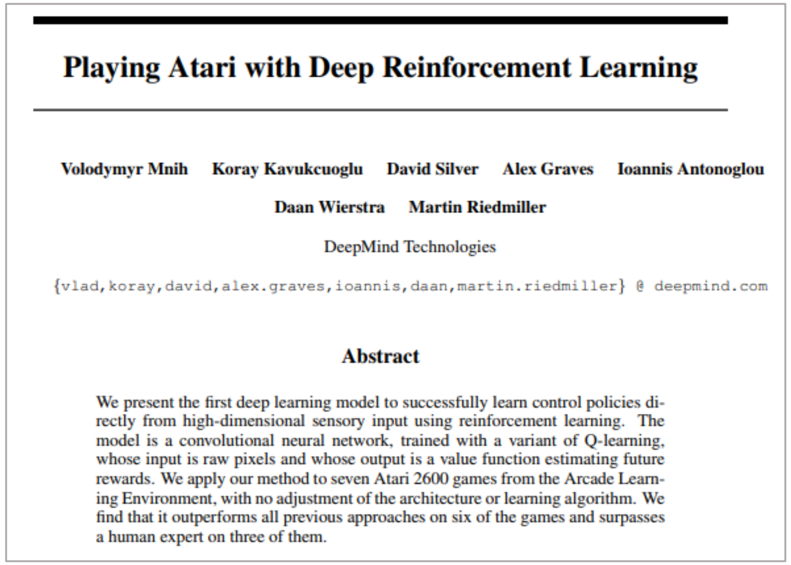
❑ Reinforcement Learning (RL) has been used for many applications where an agent interacts with an environment while trying to learn optimal sequence of decisions – optimal control problems:

- Manufacturing, e.g., robot arms to assemble cars.
- Financial strategy, e.g., buy or sell to maximize the value of the portfolio
- Inventory management or resource management



- Crites, R.H. and Barto, A.G. (1998). Elevator Group Control Using Multiple Reinforcement Learning Agents. Machine Learning, 33:235-262.
- Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A Survey. Journal of Artificial Intelligence Research, 4:237-285.
- Singh, S., Kearns, M. (2002). Near-Optimal Reinforcement Learning in Polynomial Time. Machine Learning journal, Volume 49, Issue 2, pages 209-232, 2002.
- Schultz, W. (2002). Getting formal with dopamine and reward. Neuron, 36:241-263.
- Sutton, R. S. (1984). Temporal credit assignment in reinforcement learning. PhD thesis, University of Massachusetts, Amherst
- Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coding. In D. S. Touretzky, M. C. Mozer and M. E. Hasselmo (eds.) Advances in Neural Information Processing, pp. 1038-1044, MIT Press, Cambridge, CA.
- Sutton, R. S. and Barto, A. G. (1998). Reinforcement Learning: An Introduction. Bradford Books, MIT Press, Cambridge, MA, 2002 edition.

Shed new light on RL with neural networks



Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou Daan Wierstra Martin Riedmiller

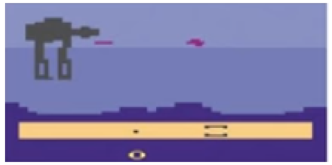
DeepMind Technologies

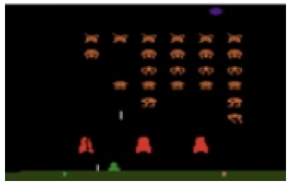
{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

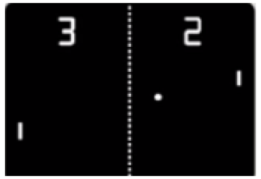
Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

2013







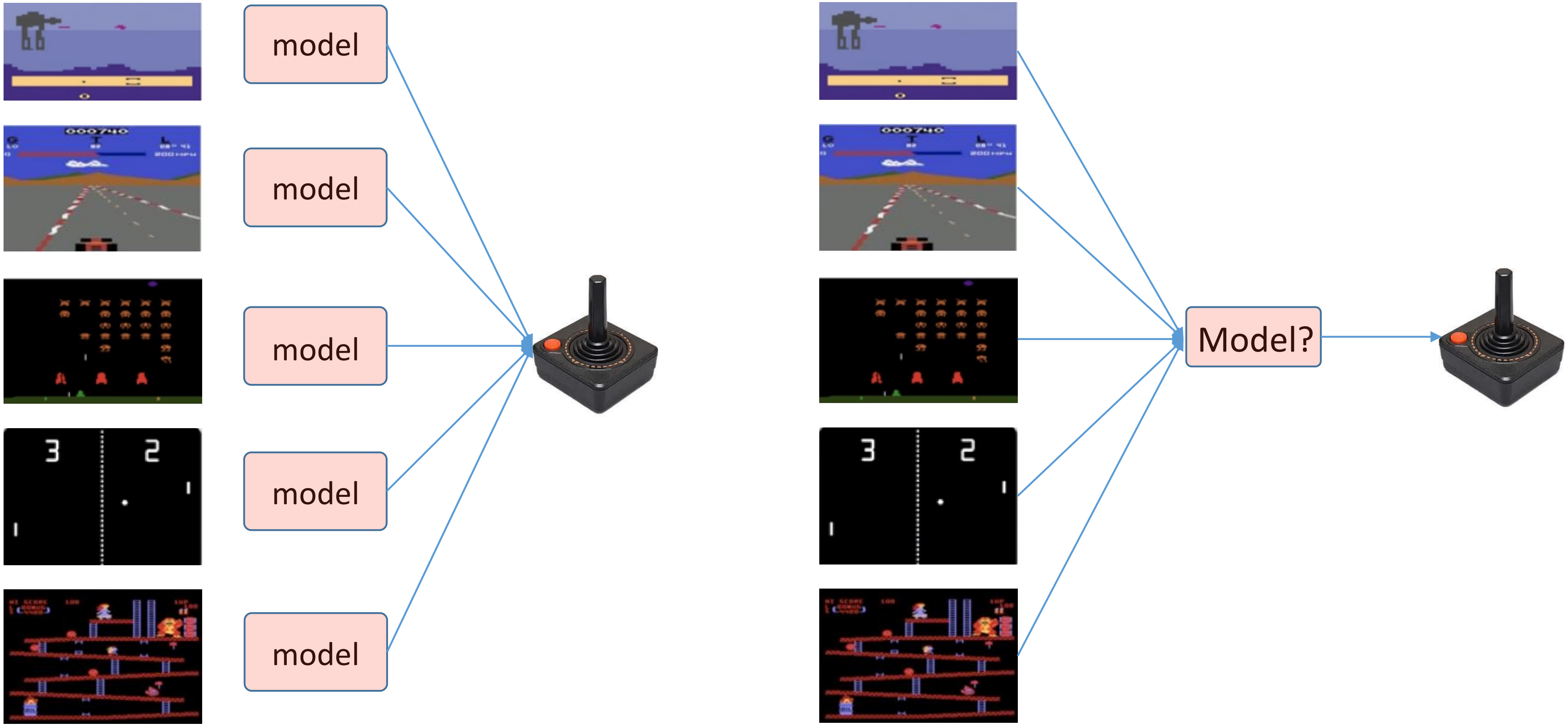
nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
LEARNING CURVE
Self-taught AI software attains human-level performance in video games
SHARE DATA IN OUTBREAKS
A GIANT IN THE EARLY UNIVERSE
TELEPORTATION FOR TWO

2015

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO
SONGBIRDS A LA CARTE
SAFEGUARD TRANSPARENCY
WHEN GENES GOT 'SELFISH'

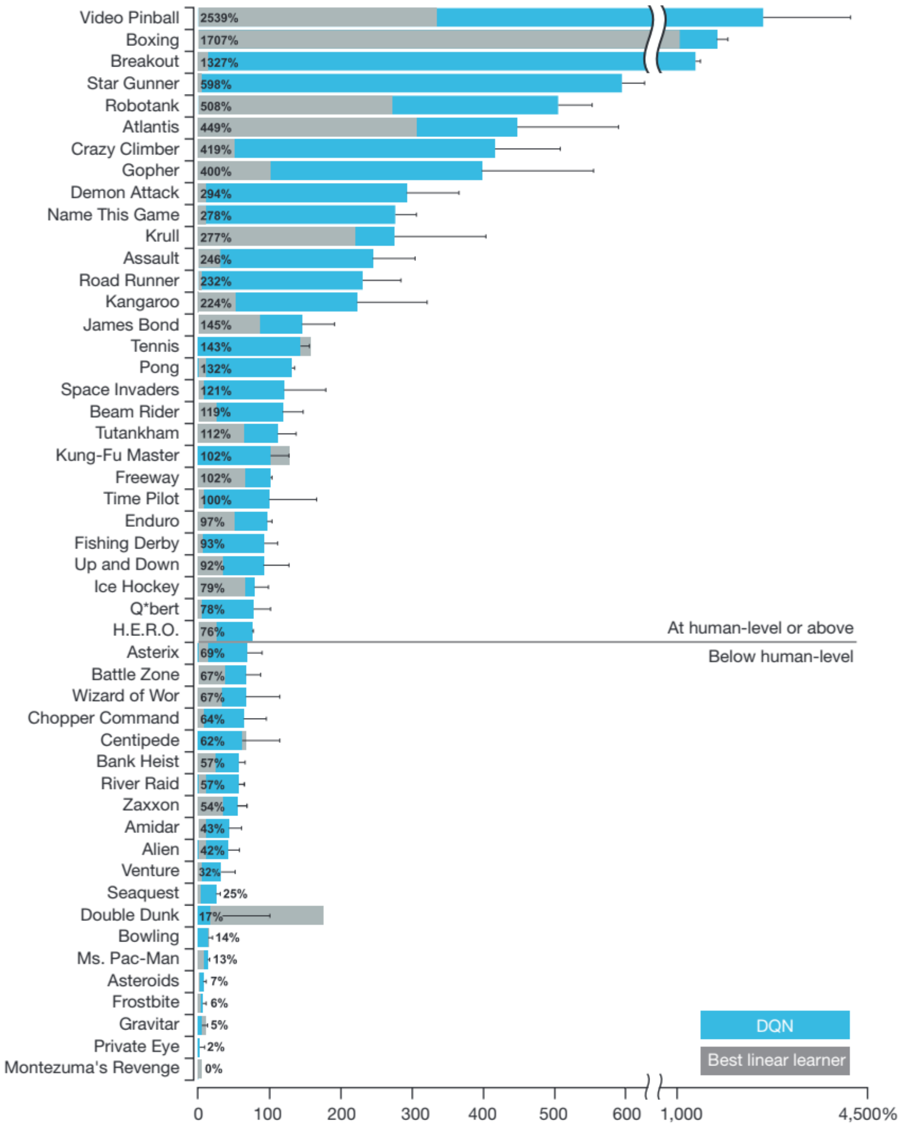
2016

Shed new light on RL with neural networks – cont.



How to model the environment?

How good is the reinforcement learning algorithm?



DeepMind AI Reduces Google Data Centre Cooling Bill by 40%

20 July 2016

- Human-level control through deep reinforcement learning, Feb. 26, 2015, Nature
- https://deepmind.com/applied/deepmind-for-google/

Terminology

State (S)

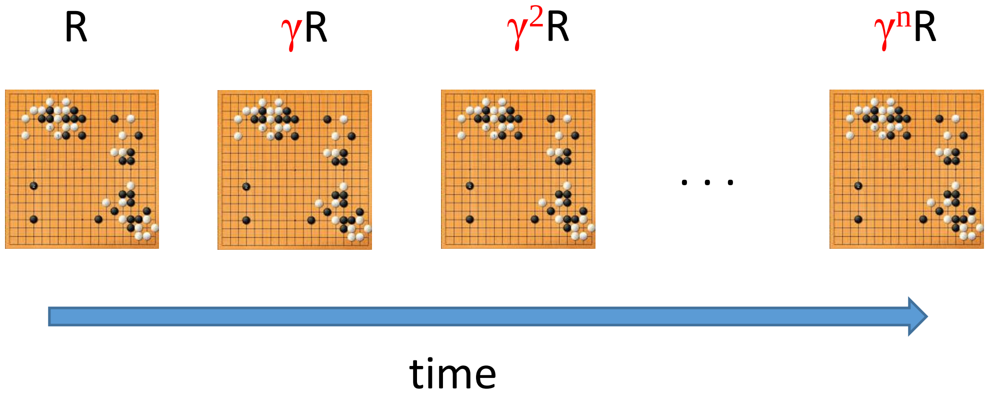
1) State (S)
2) Reward (R)
   - Discount factor (γ)
3) Policy (π)
   - Action (A)
4) Environment
   - Transit Probability (P)

- A representation of environment that an agent recognizes.
- A agent takes an action given state
- E.g., pixel information as shown below

Reward (R) and Discount factor (γ)

1) State (S)
2) Reward (R)
   - Discount factor (γ)
3) Policy (π)
   - Action (A)
4) Environment
   - Transit Probability (P)

- When an agent takes an action, it considers two types of rewards
  1. Immediate reward
  2. Future accumulated reward
- Reward in a different time step may need to be treated differently: $\gamma[0,1]$

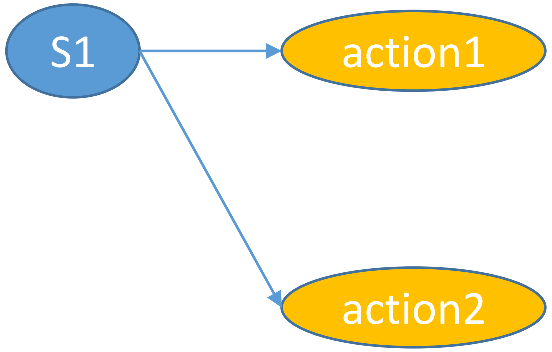


Policy (π) and Action (A)

1) State (S)
2) Reward (R)
   - Discount factor (γ)
3) Policy (π)
   - Action (A)
4) Environment
   - Transit Probability (P)

- Agent has a set of actions given state
- Policies π is a distribution over actions given states: probability of action *a* given state *s*
  - *Deterministic policy: a = π(s)*
  - *Stochastic policy: π(a|s)*

$$\pi(\mathrm{a}\,|\,\mathrm{s}) = \mathrm{P}[\mathrm{A}_t = \mathrm{a}\,|\,\mathrm{S}_t = \mathrm{s}]$$

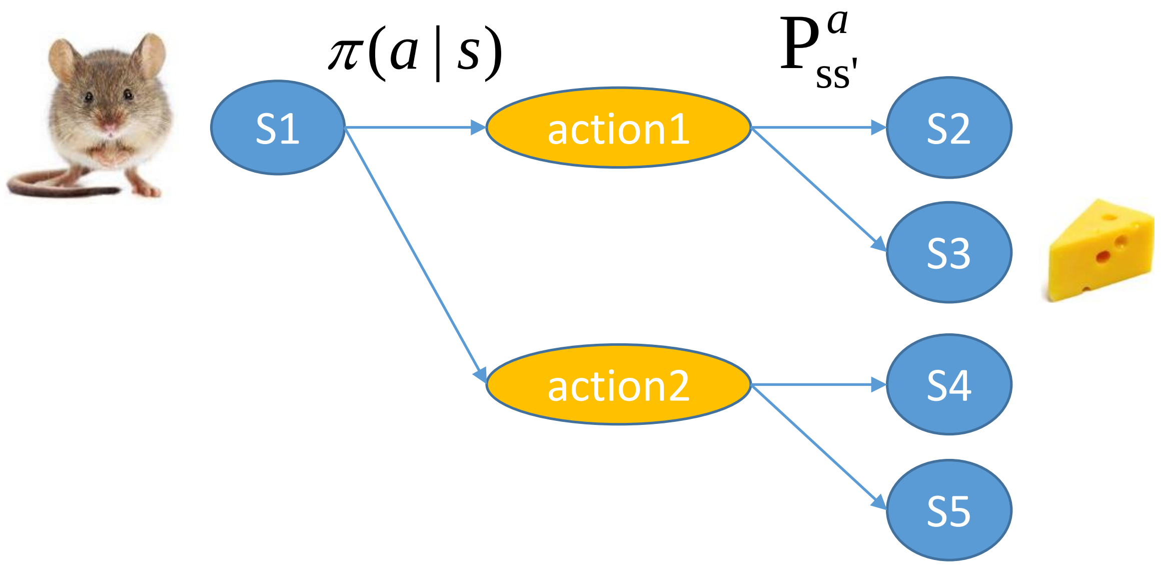


Environment

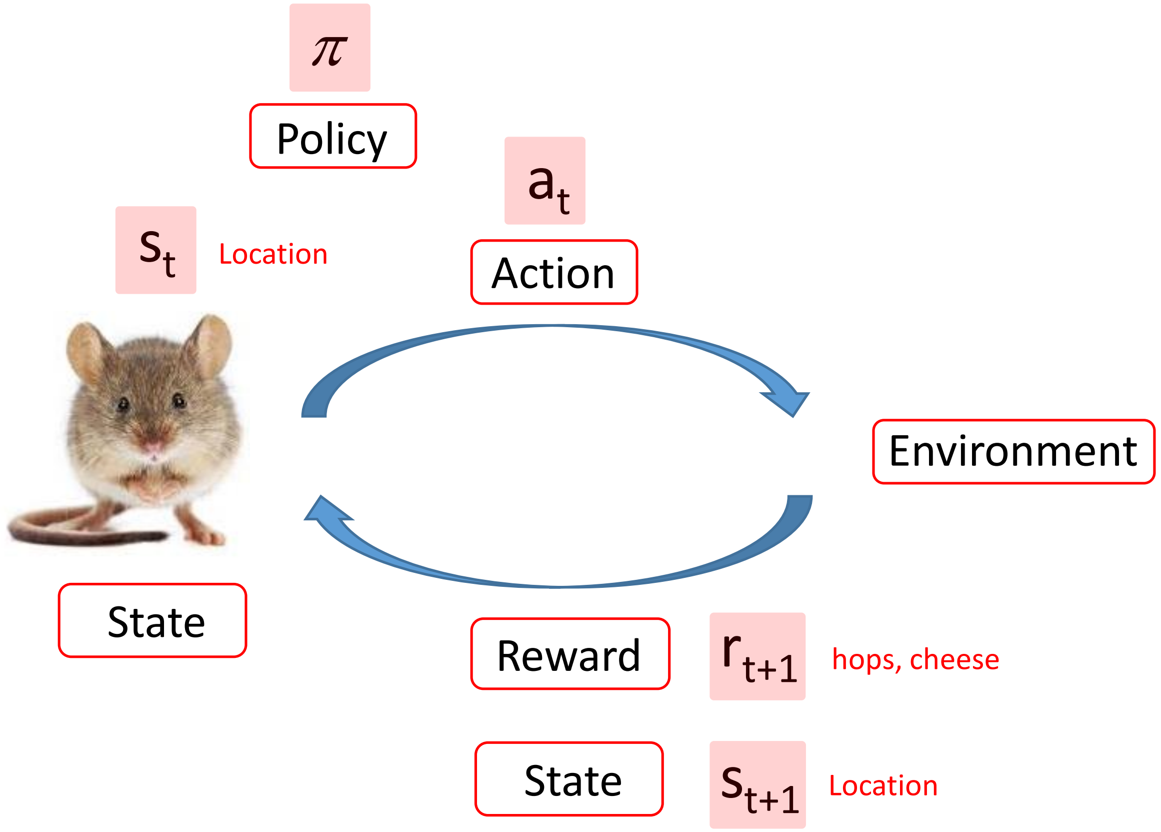
1) State (S)
2) Reward (R)
   - Discount factor (γ)
3) Policy (π)
   - Action (A)
4) Environment
   - Transit Probability (P)

- Due to uncertainty of environment, an action taken by an agent does not guarantee to a certain state.
- The uncertainty is represented as transition probability.

$$P^{a}_{ss'} = P^{a}[S_{t+1} = s' | S_{t} = s, A_{t} = a]$$



Reinforcement Learning (RL) process



π

Policy

a_t

Action

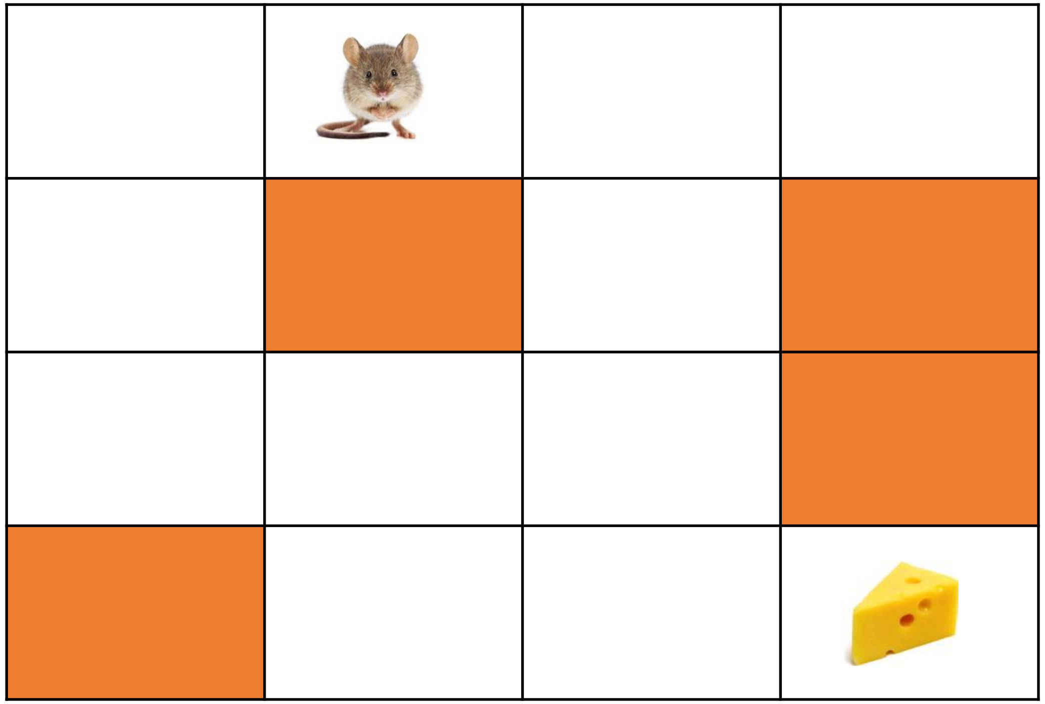
s_t Location

State

Environment

Reward r_{t+1} hops, cheese

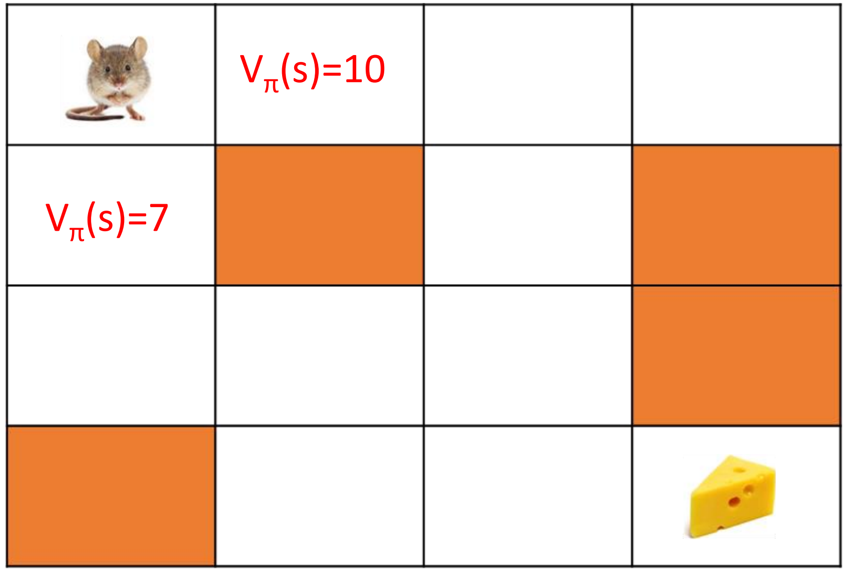
State s_{t+1} Location



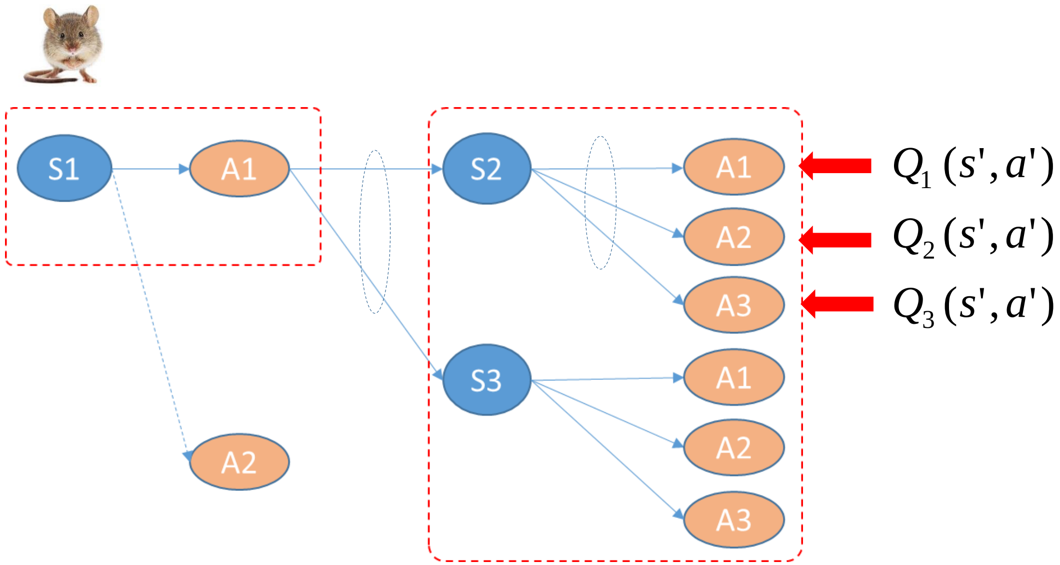
$$r_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k}$$

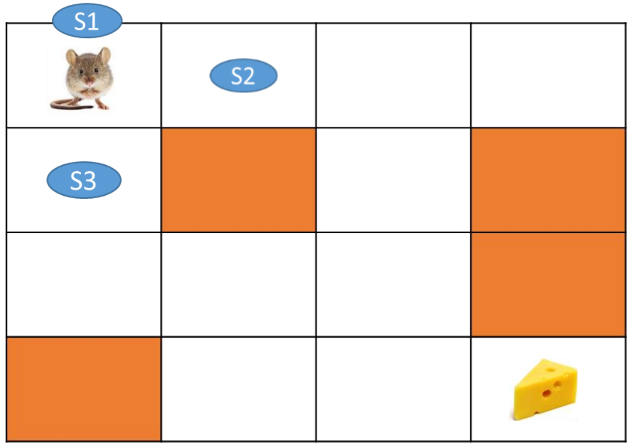
How to find the optimum policy?: two functions as an signalpost

1) State value function: $V_{\pi}(s)$
   - Expected reward when starting in state *s* and following policy π thereafter.
   - The mouse needs to know the value of next state before making an action.
2) Action value function: $Q_{\pi}(s,a)$
   - Expected reward when taking action *a* in state *s* and following policy π thereafter.
   - The mouse just needs to take an action based on Q value.



How to find the Q(s,a) function?: Q learning





$$Q_\pi(s,a) = r_s^a + \gamma \max_{a'} Q_\pi(s',a')$$

Immediate reward

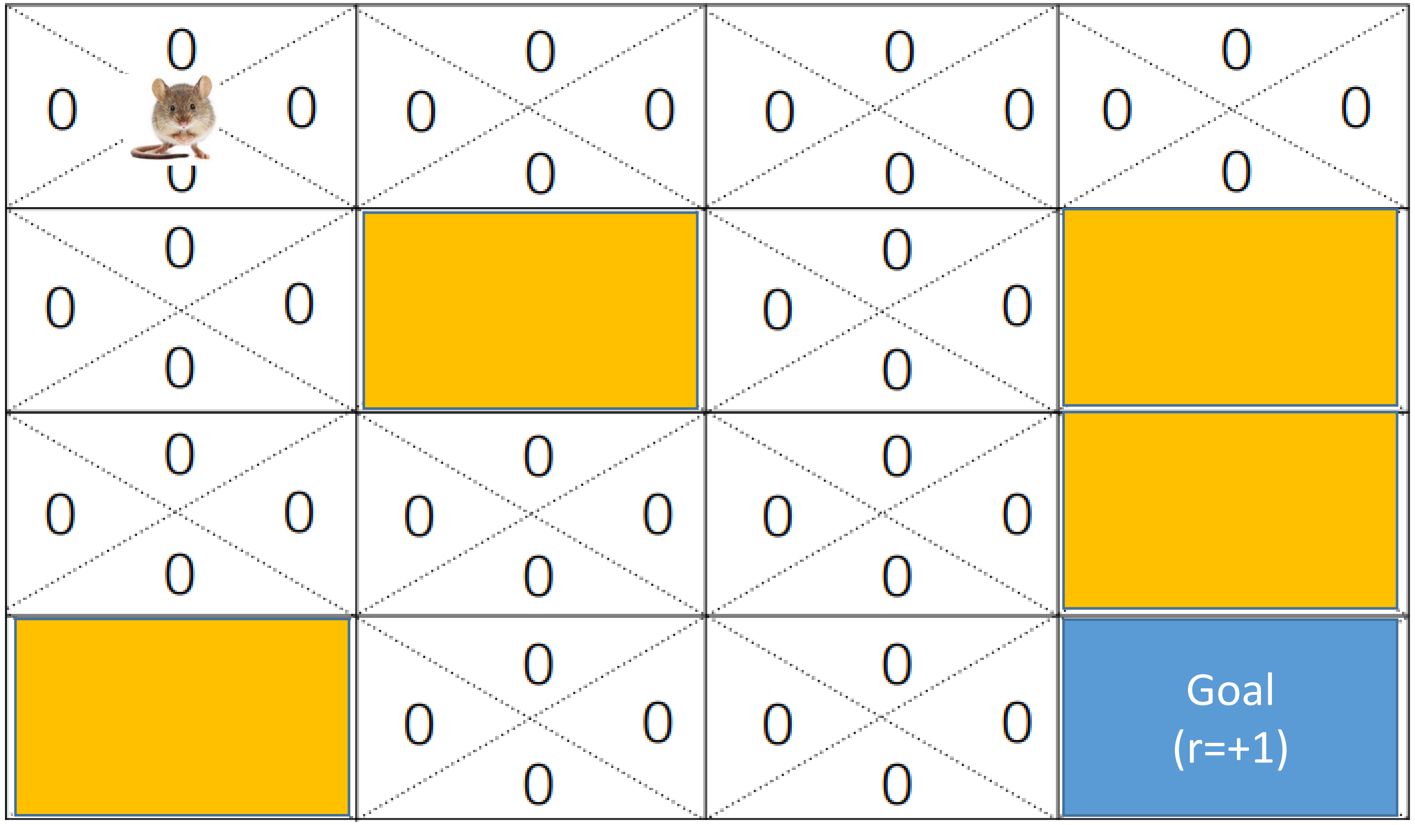
Discounted reward of successor state

Q learning: example

Q learning: Q(s,a) based on Q table

❑ One box represents a state and actions which can be taken by an agent (N, E, S, W)

❑ Initial Q values at individual states are set to zero

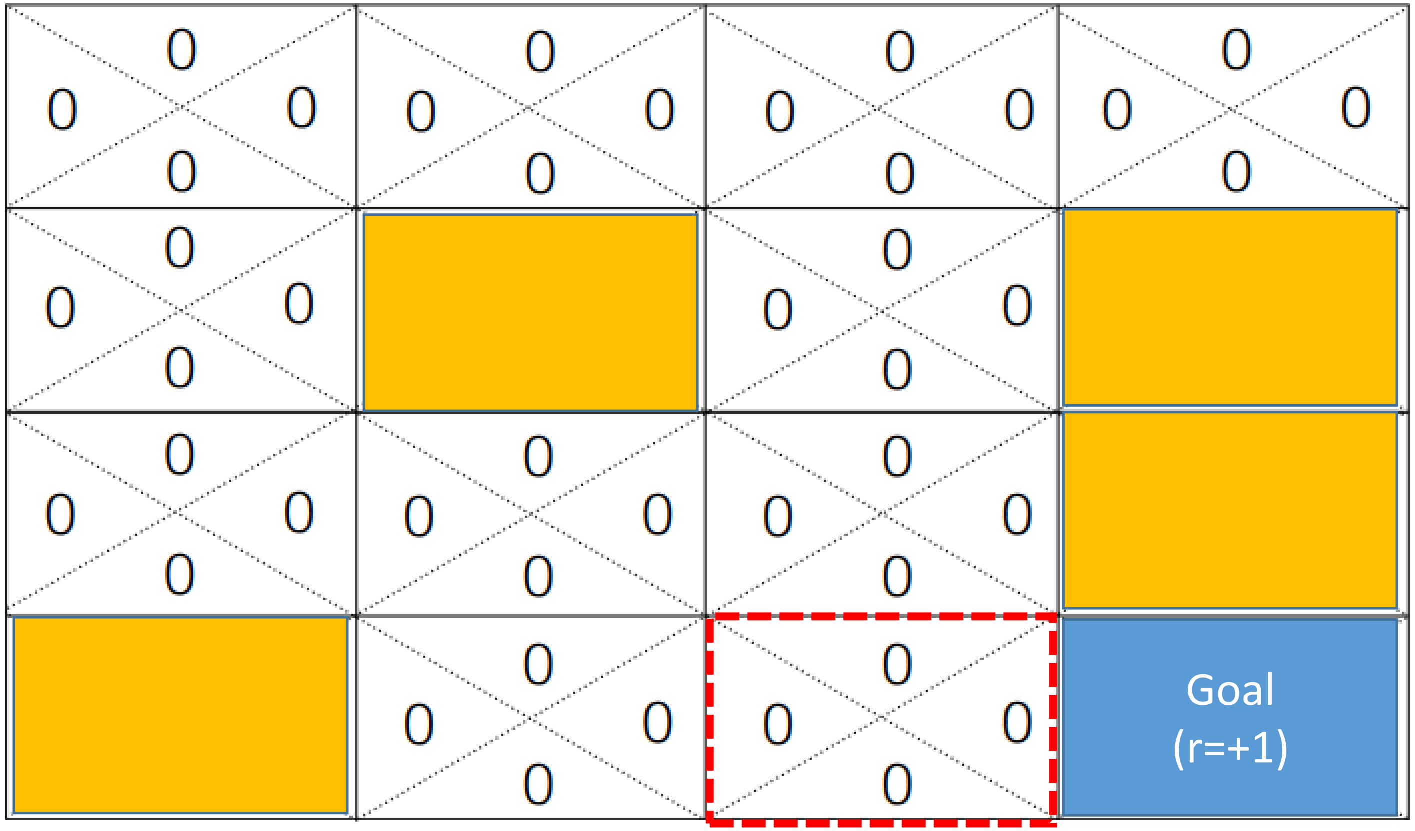


https://hunkim.github.io/ml/RL/rl03.pdf

Q learning: Q(s,a) based on Q table

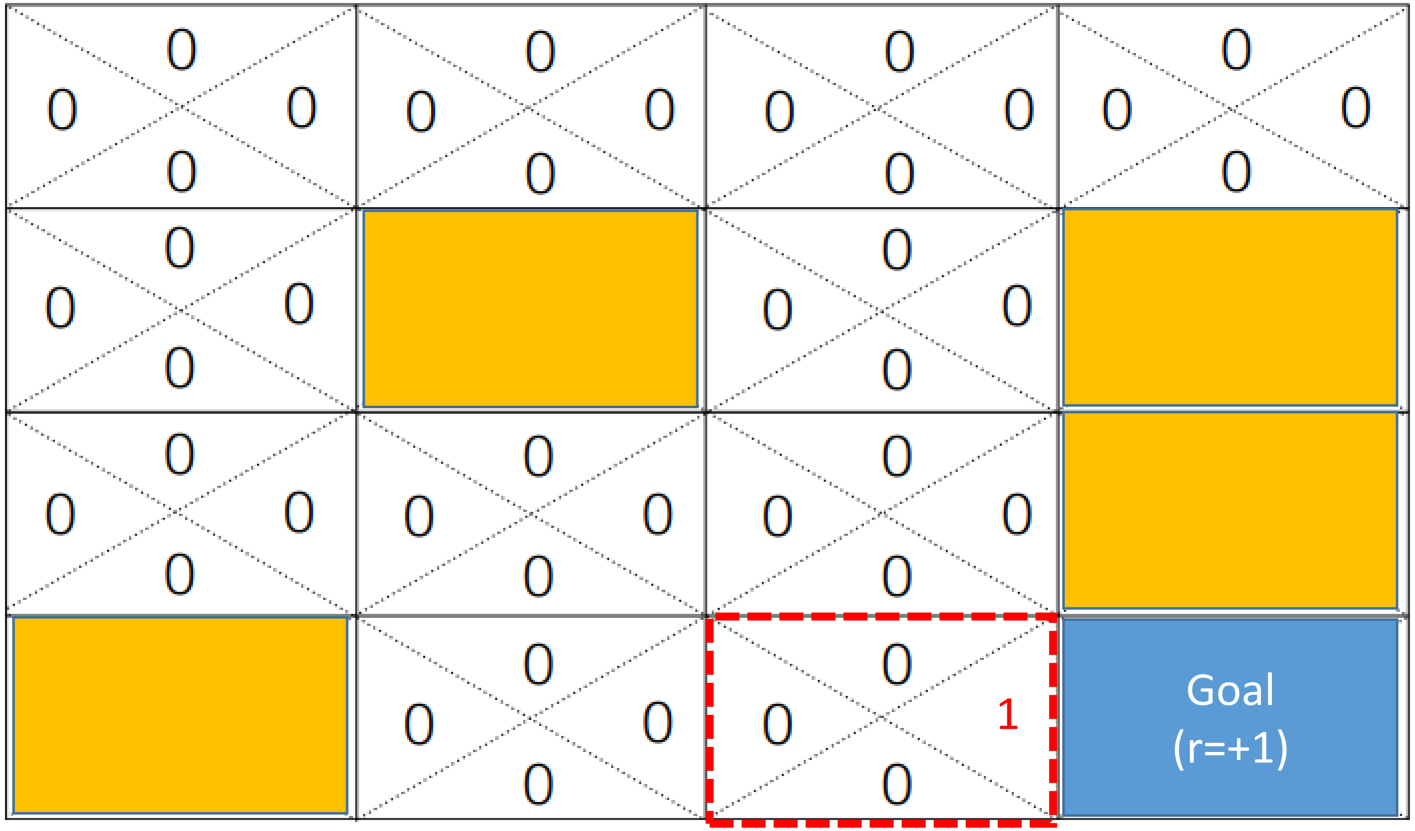
❑ Agent takes an action at each state based on value of Q(s,a).

❑ Assuming that an agent is at the next to the goal and takes an action to E (East).



Q learning: Q(s,a) based on Q table

- ❑ Q(s, E) is set to one: 1: (immediate reward) + 0: Q(s', a')

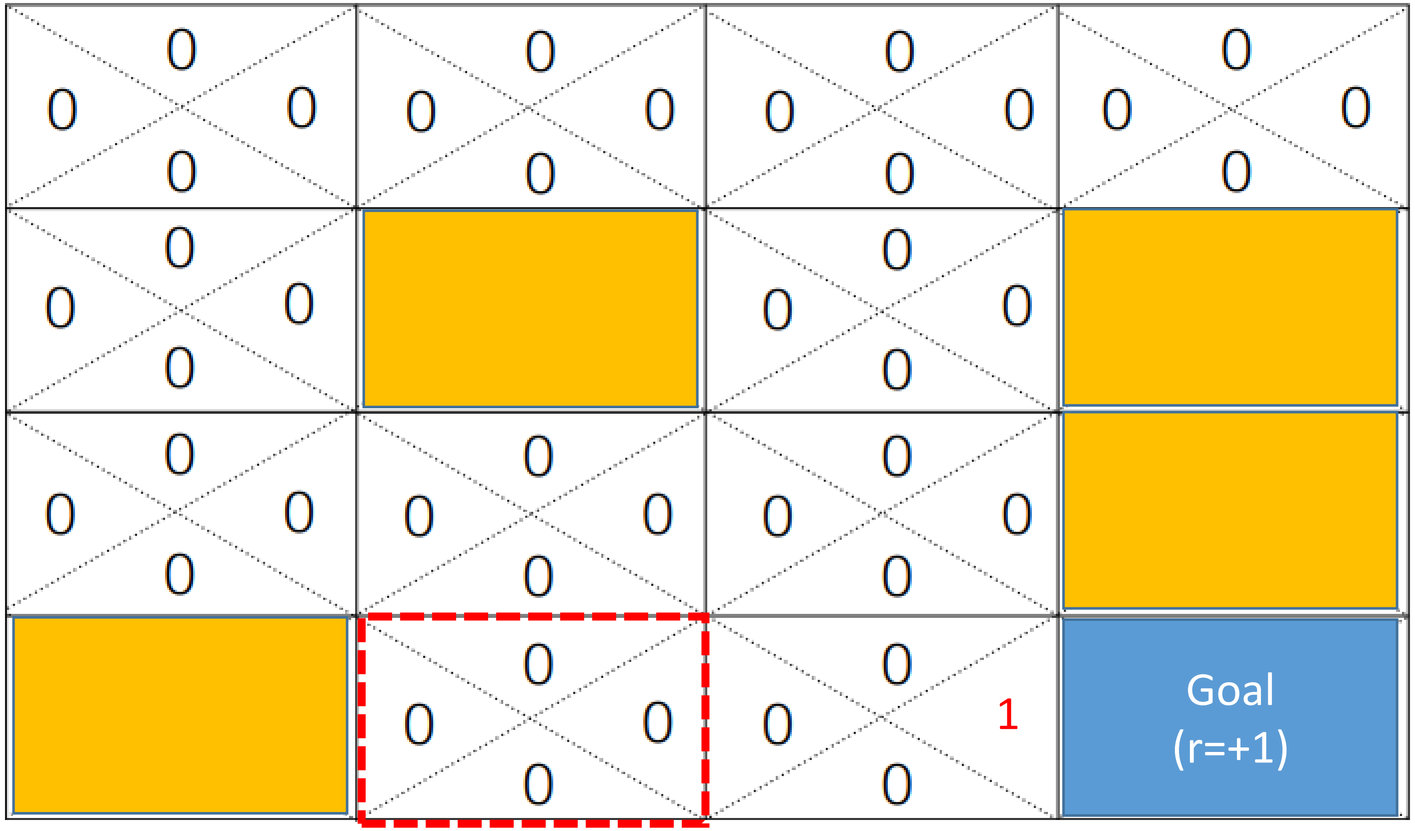


Q(s,a) = 1 : (immediate reward) + 0 : Q(s',a')

https://hunkim.github.io/ml/RL/rl03.pdf

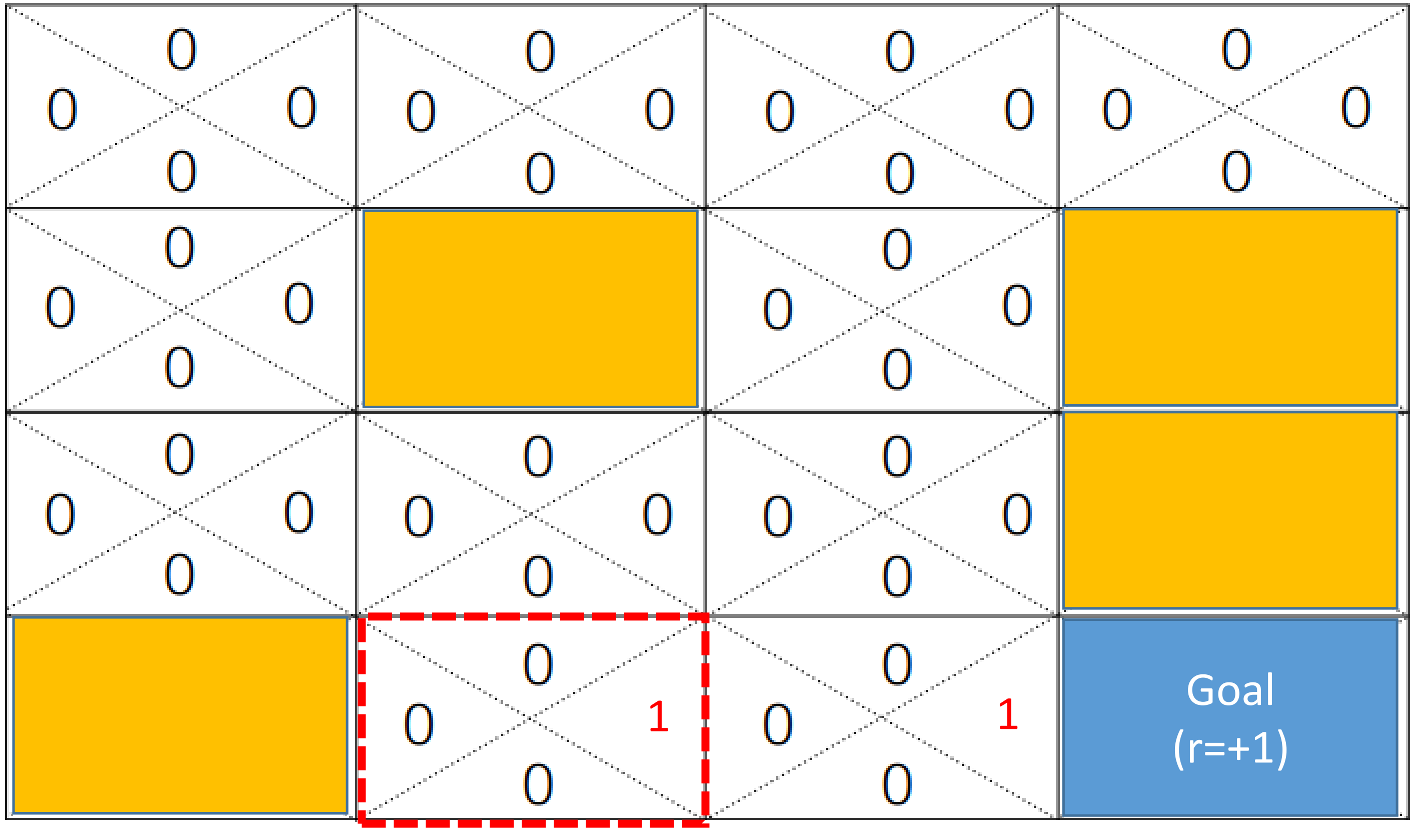
Q learning: Q(s,a) based on Q table

- ❑ Again assume that an agent ends up to the state below and takes an action to E
- ❑ Then, Q(s, E) is set to one: 0 (immediate reward) + 1 Q(s', a')



Q learning: Q(s,a) based on Q table

- ❑ Again assume that an agent ends up to the state below and takes an action to E
- ❑ Then, Q(s, E) is set to one: 0 (immediate reward) + 1 Q(s’, a’)

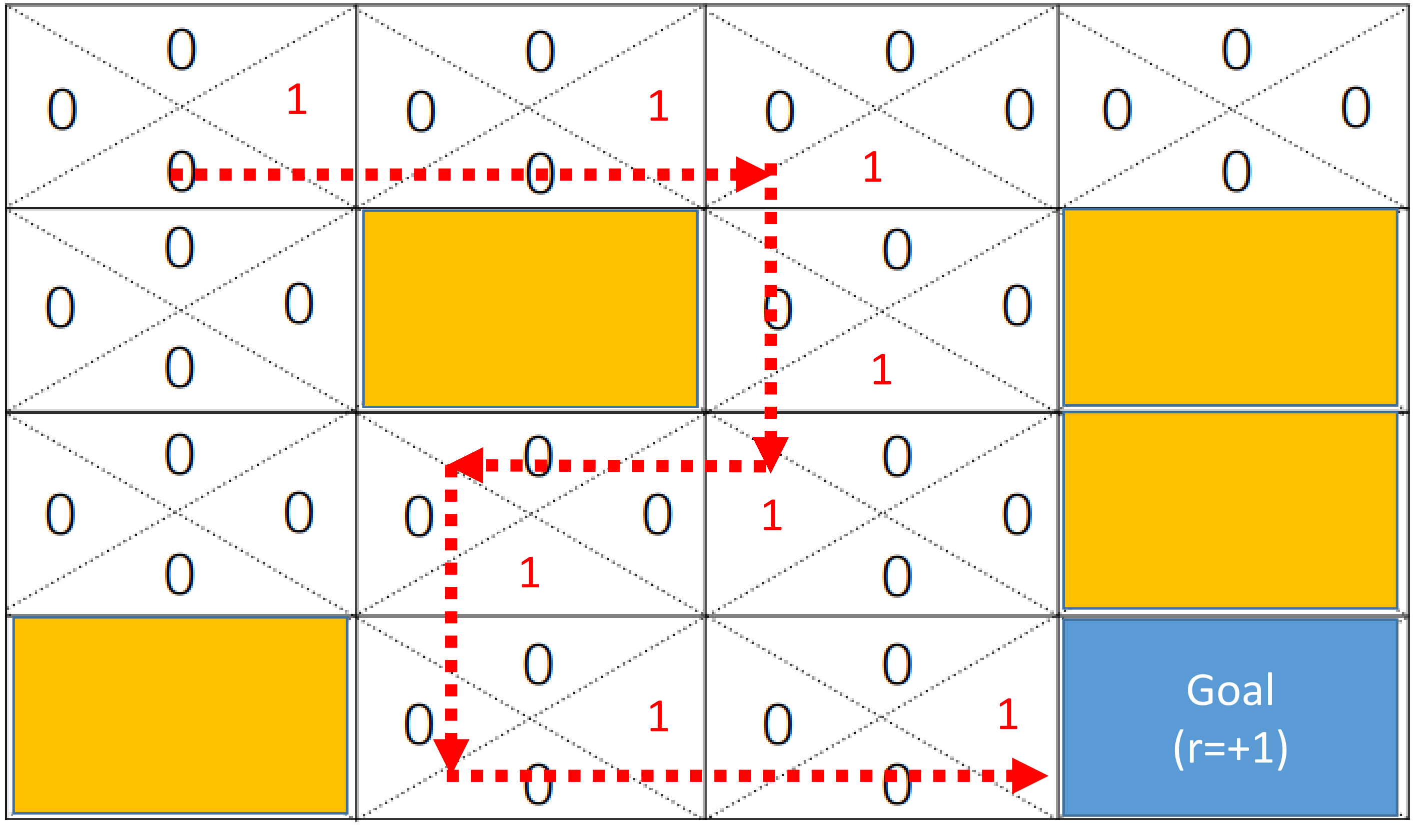


Q(s,a) = 0 : (immediate reward) + 1 : Q(s’,a’)

https://hunkim.github.io/ml/RL/rl03.pdf

Q learning: Q(s,a) based on Q table

- ❑ In the same way, Q table can be built as follows:



https://hunkim.github.io/ml/RL/rl03.pdf

Q learning: Exploit vs Exploration

E-greedy policy

```
e = 0.1

if rand < e:      → 10% random decision
        action = random
else:             → 90% deterministic decision
        action = argmax(Q(s,a))
```

Decaying E-greedy policy

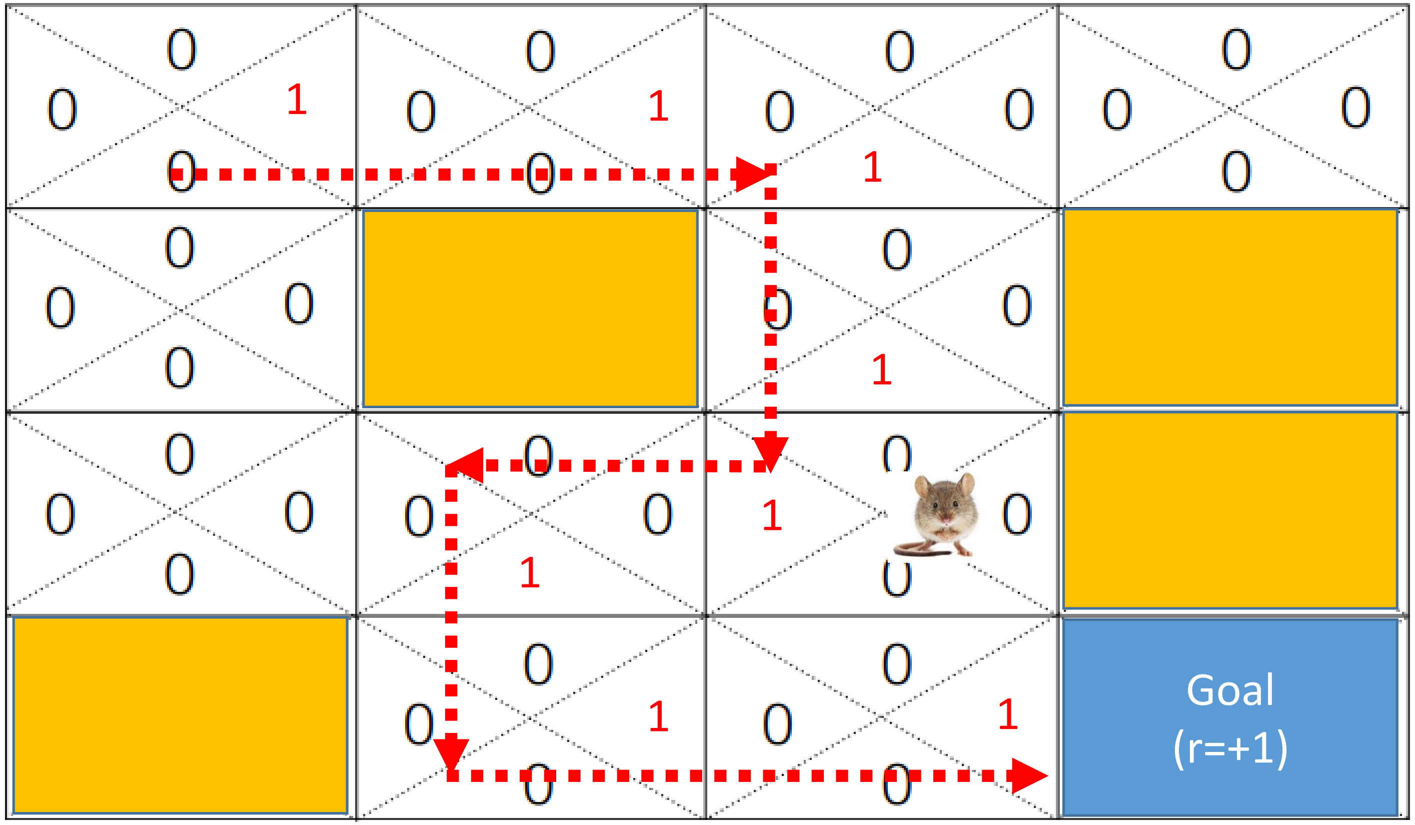
```
for i in range (1000)

e = 0.1 / (i+1)     → Random to deterministic
                      decision as iteration goes on
if rand < e:
        action = random
else:
        action = argmax(Q(s,a))
```

Machine Learning, T. Mitchell, McGraw Hill p375

Q learning: Exploit vs Exploration

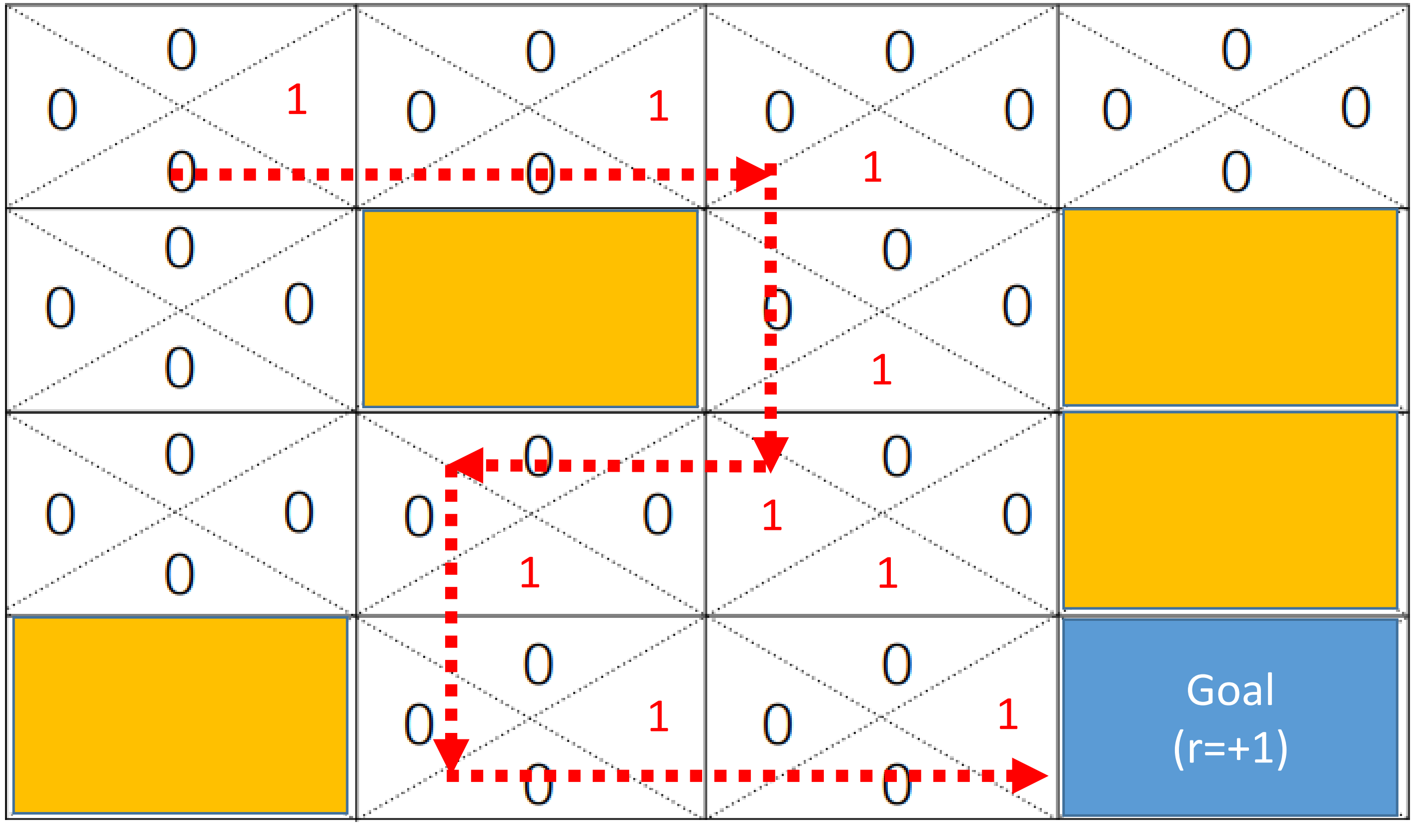
- ❑ The route to the goal is not an optimum.
- ❑ How to select a different route occasionally?



https://hunkim.github.io/ml/RL/rl04.pdf

Q learning: Exploit vs Exploration

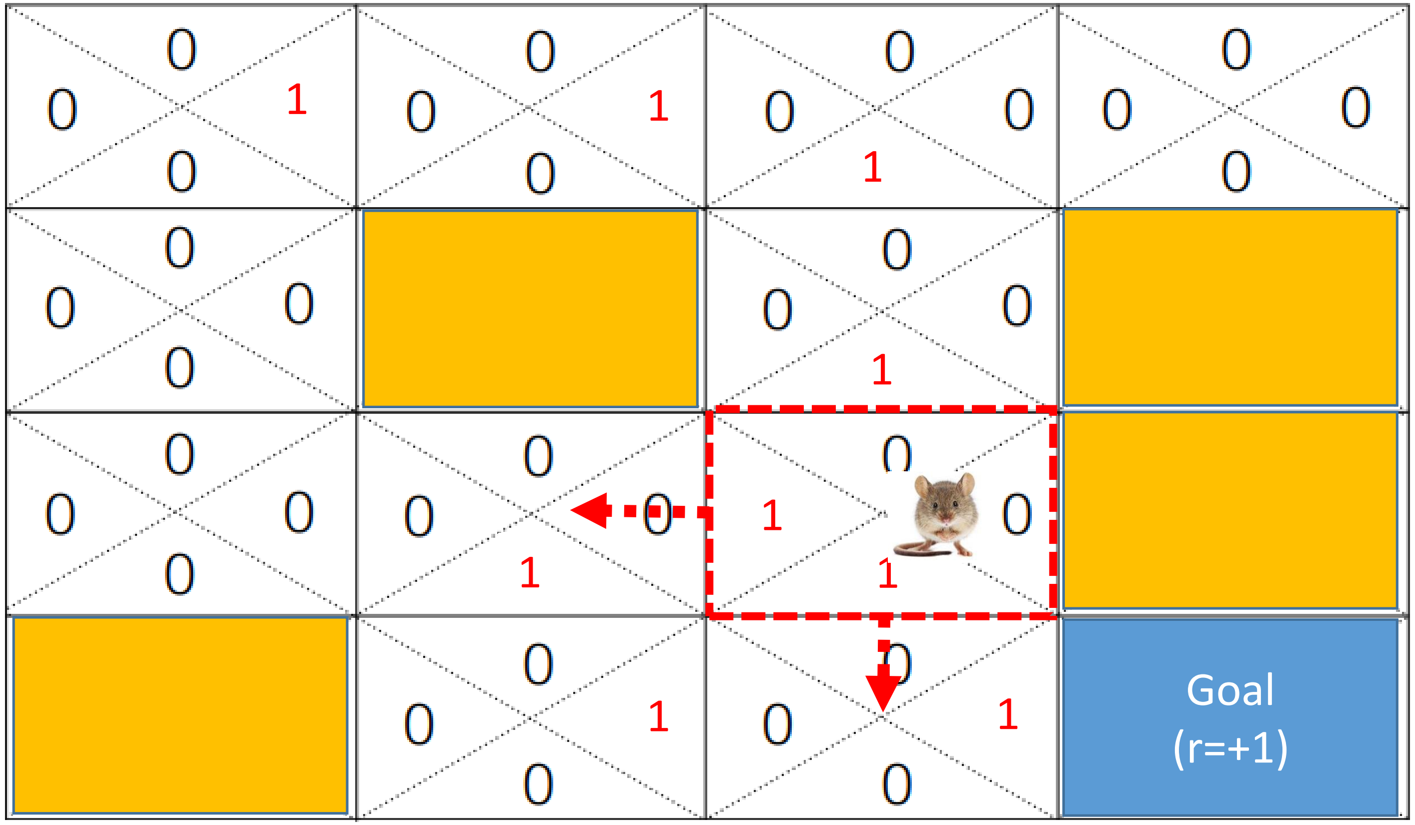
- ❑ The route to the goal is not an optimum.
- ❑ How to select a different route occasionally?



https://hunkim.github.io/ml/RL/rl04.pdf

Q-learning: Discounted reward γ

- ❑ Q value does not tell which path is better

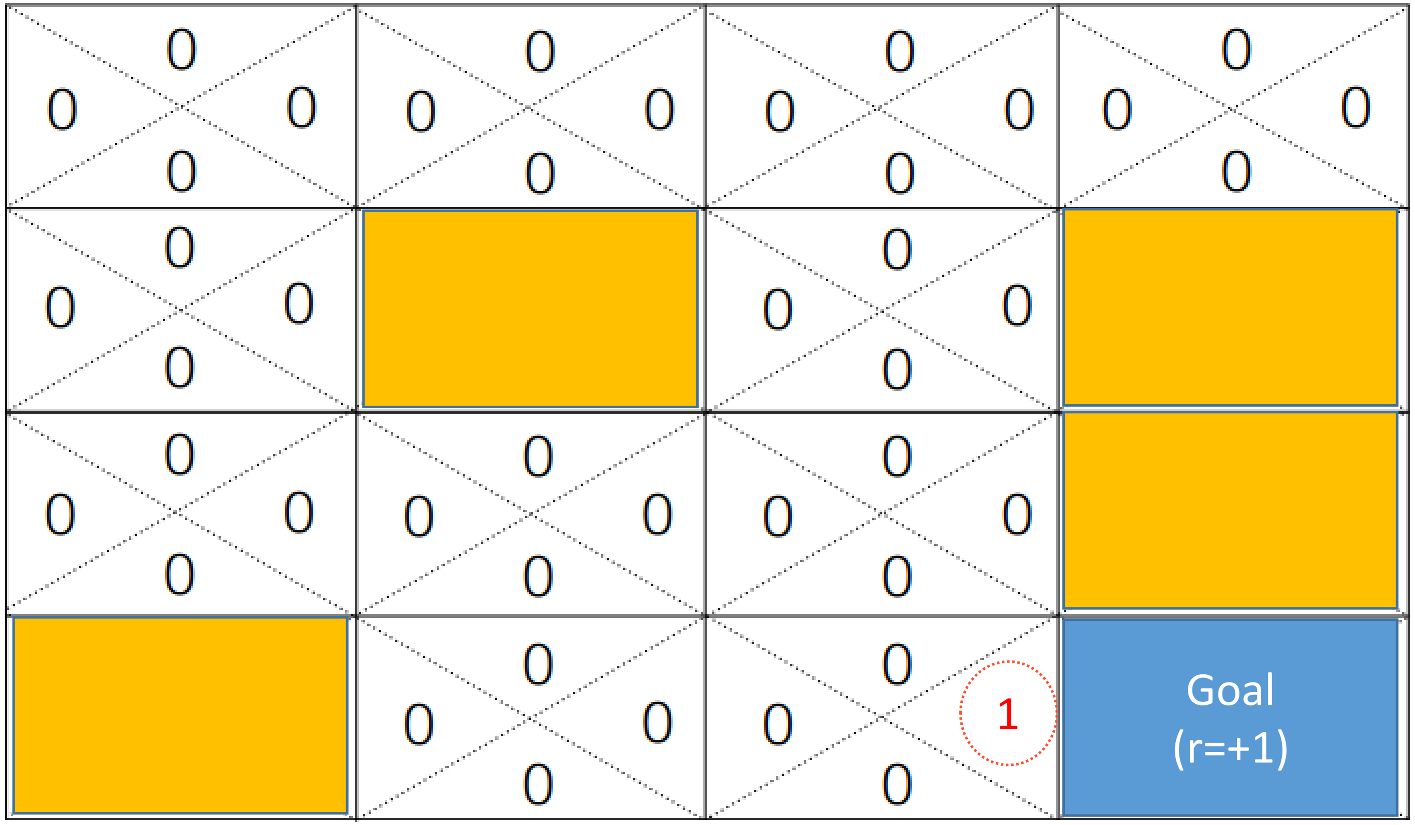


Q(s,a) = 0 : (immediate reward) + 1 : Q(s’,a’)

https://hunkim.github.io/ml/RL/rl04.pdf

Q-learning: Discounted reward γ

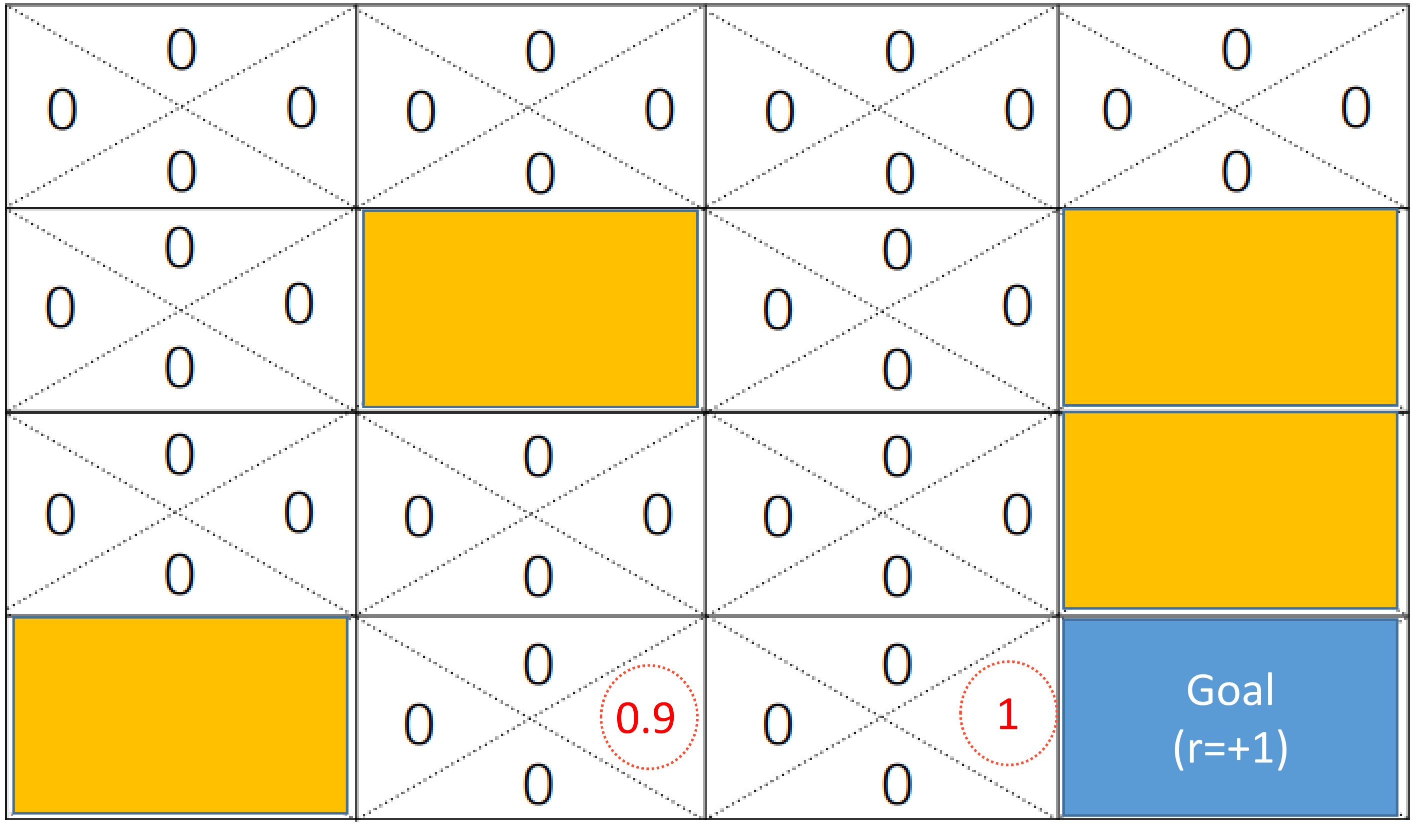
- ❑ Q value does not tell which path is better
- ❑ Let's introduce discounted reward $\gamma=0.9$ to the equation



$Q(s,a) = r + \gamma\, Q(s',a') = 1 + 0.9 \times 0 = 1$

Q-learning: Discounted reward γ

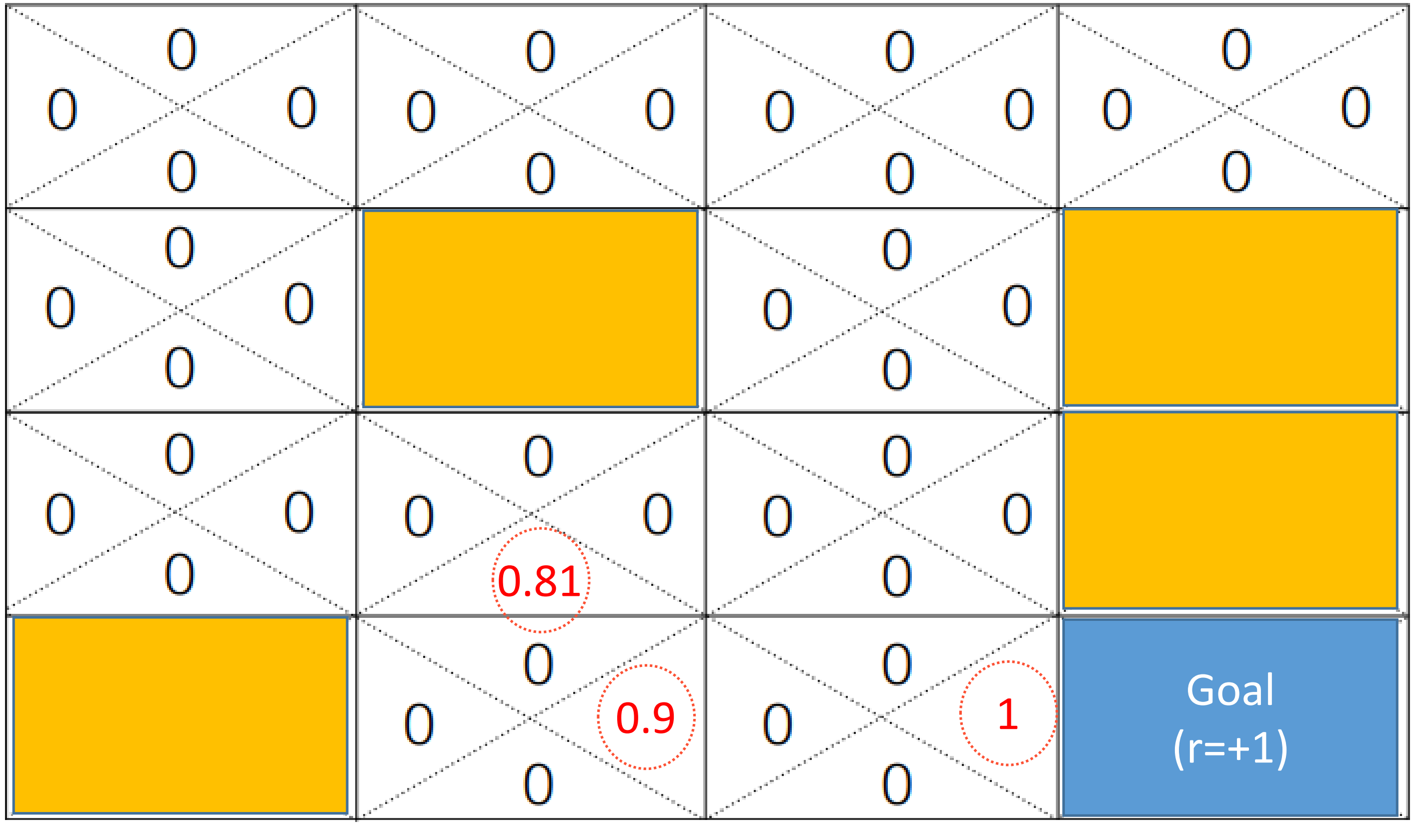
- ❑ Q value does not tell which path is better
- ❑ Let’s introduce discounted reward γ=0.9 to the equation



$Q(s,a) = r + \gamma\ Q(s',a') = 0 + 0.9 \times 1 = 0.9$

Q-learning: Discounted reward γ

- ❑ Q value does not tell which path is better
- ❑ Let's introduce discounted reward γ=0.9 to the equation

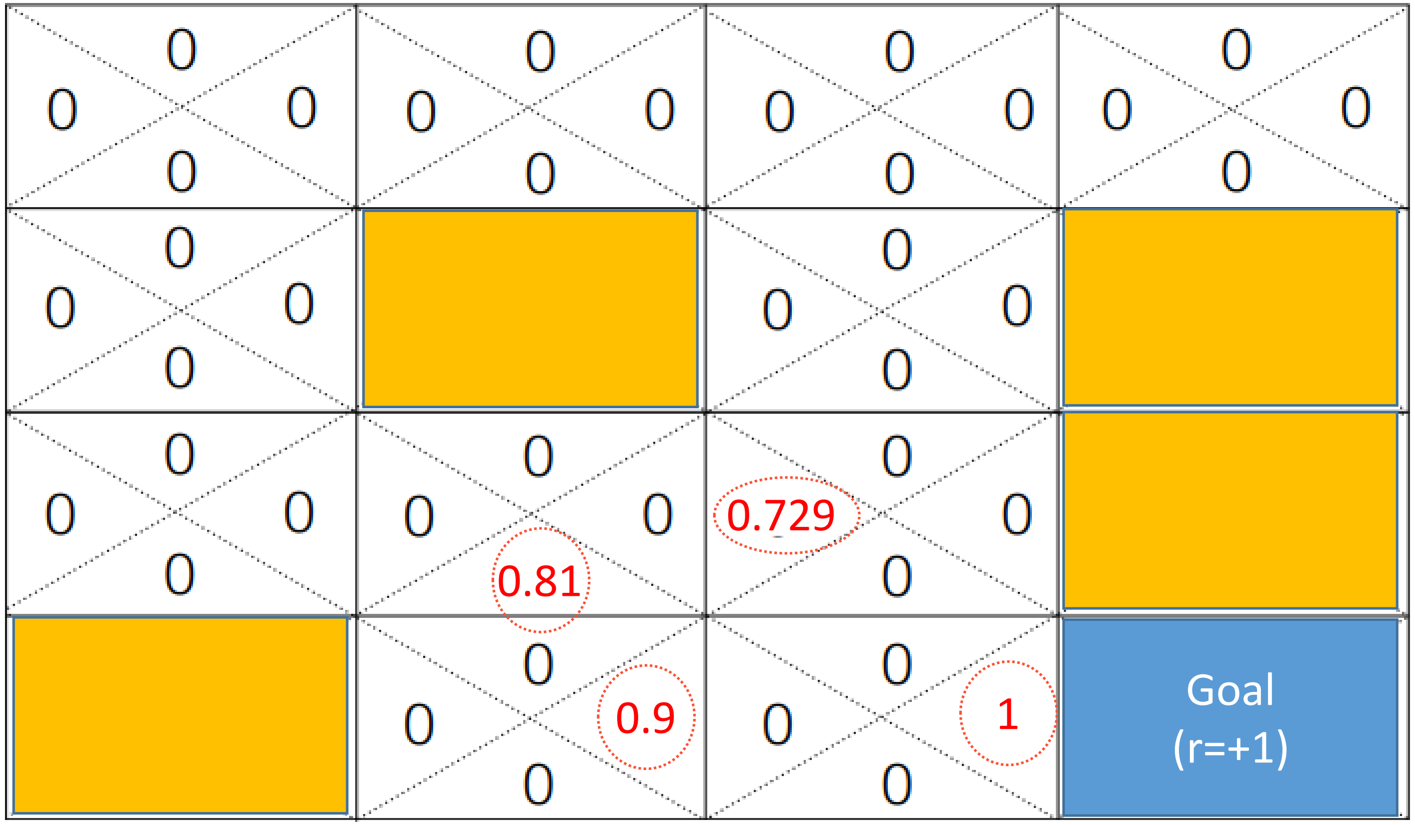


Q(s,a) = r + γ Q(s',a')= 0 + 0.9 x 0.9 = 0.81

https://hunkim.github.io/ml/RL/rl04.pdf

Q-learning: Discounted reward γ

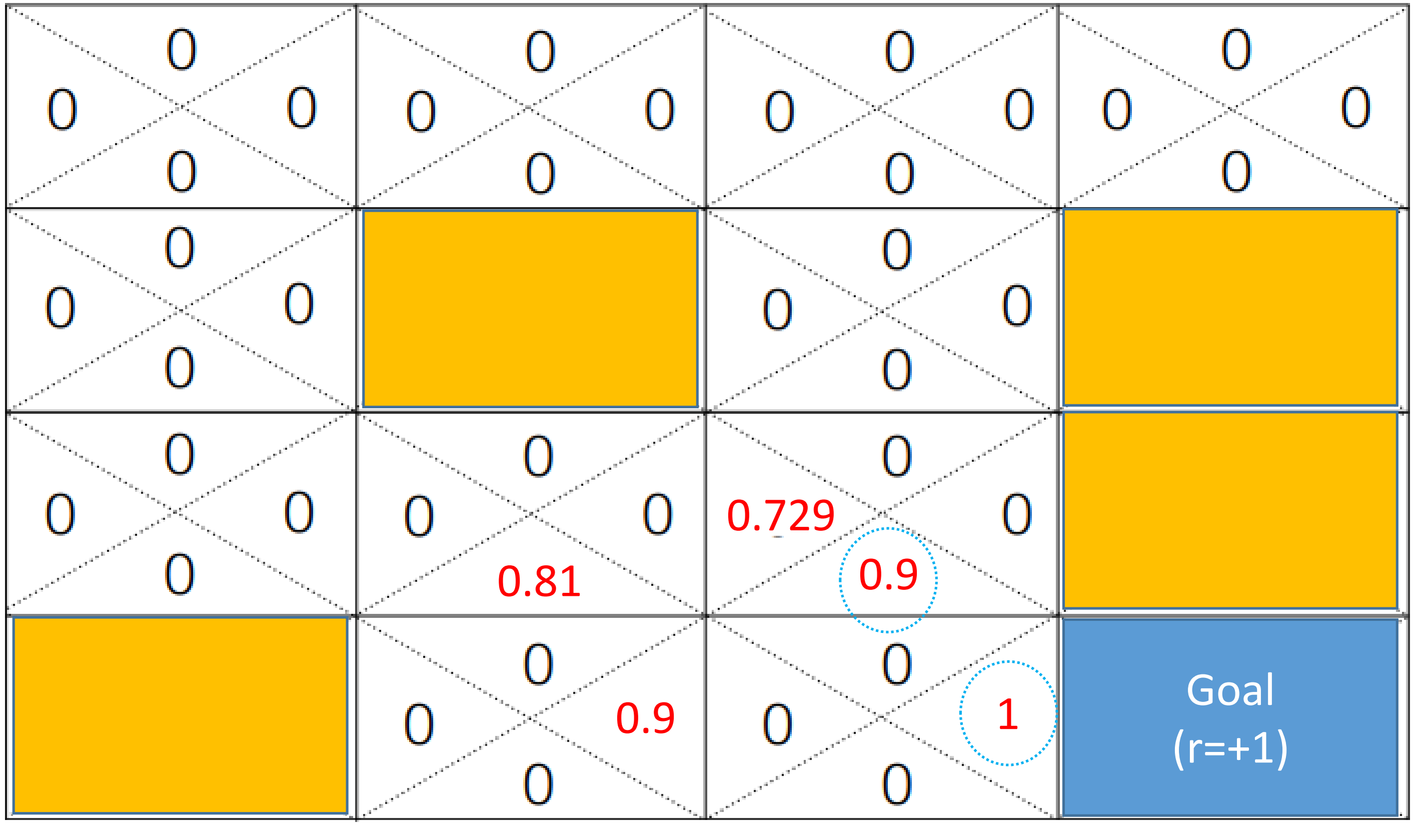
- ❑ Q value does not tell which path is better
- ❑ Let's introduce discounted reward γ=0.9 to the equation



$Q(s,a) = r + \gamma\ Q(s',a') = 0 + 0.9 \times 0.81 = 0.729$

Q-learning: Discounted reward γ

- ❑ Q value does not tell which path is better
- ❑ Let’s introduce discounted reward $\gamma=0.9$ to the equation

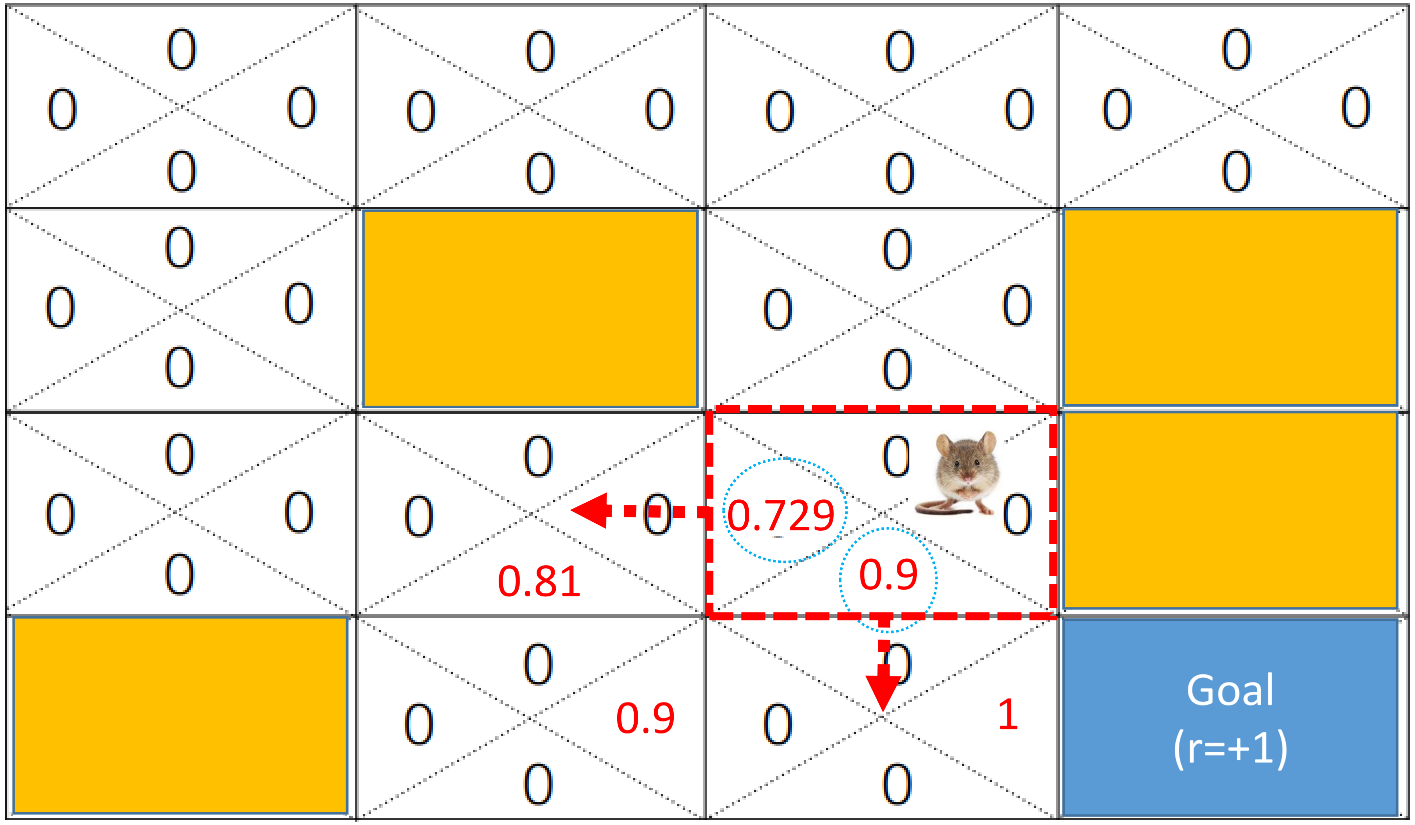


$Q(s,a) = r + \gamma\ Q(s',a') = 0 + 0.9 \times 1 = 0.9$

https://hunkim.github.io/ml/RL/rl04.pdf

Q-learning: Discounted reward γ

- ❑ Q value does not tell which path is better
- ❑ Let’s introduce discounted reward $\gamma=0.9$ to the equation



$$Q(s,a) = r + \gamma\ Q(s',a') = 0 + 0.9 \times 1 = 0.9$$

Q learning algorithm

```
For each (s, a) pair, initialize table entry $\hat{Q}(s, a)$ to zero.
Observe the current state s
Do forever:
    ▪ Select an action a and execute it
    ▪ Receive immediate reward r
    ▪ Observe the new state s'
    ▪ Update the table entry for $\hat{Q}(s, a)$ as follows:
          $\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$
    ▪ s ← s'
```

Exploit & Exploration

Discounted reward

Machine Learning, T. Mitchell, McGraw Hill p375

Deep Q-Networks (DQN)

Q learning problem

- ❑ Q (s,a) needs to be computed for every state(s)-action(a) pair.
  - If the problem size is small, we can handle it using Q table
- ❑ When state space is huge: computationally infeasible for entire state space
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Automatic driving: continuous state space
- ❑ Too many states to store in memory and also too slow to learn state space

Backgammon

Go

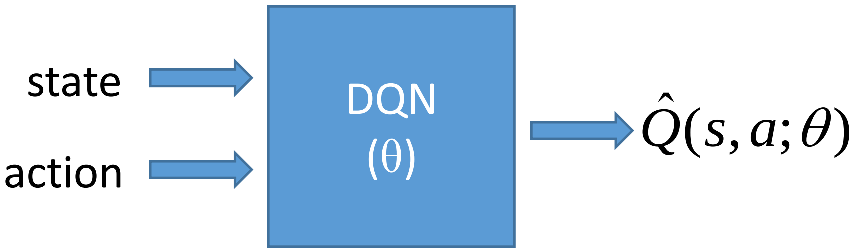
Deep Q-Network (DQN)

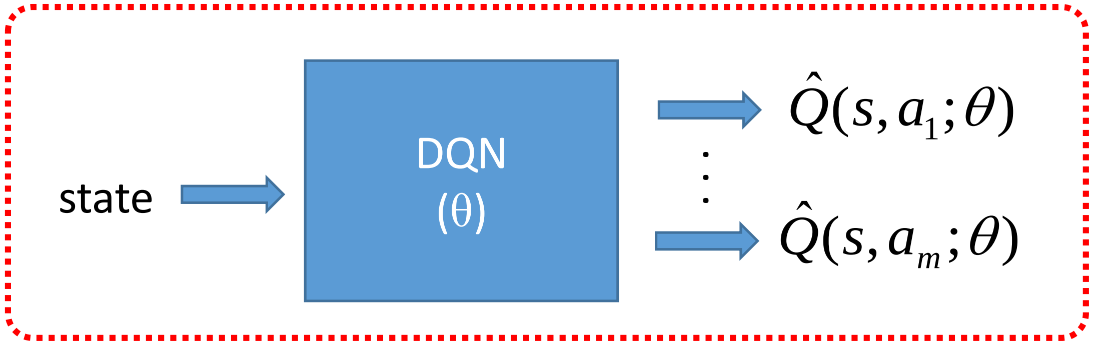
□ In 2013, a team of DeepMind (AlphaGo) proposed convolutional neural networks as an approximation of the Q(s, a) function.

□ Then, it was named as Deep Q-networks (DQN)

$$\hat{Q}(s,a;\theta) \approx Q_\pi(s,a)$$

Hyper parameters + neural network parameters





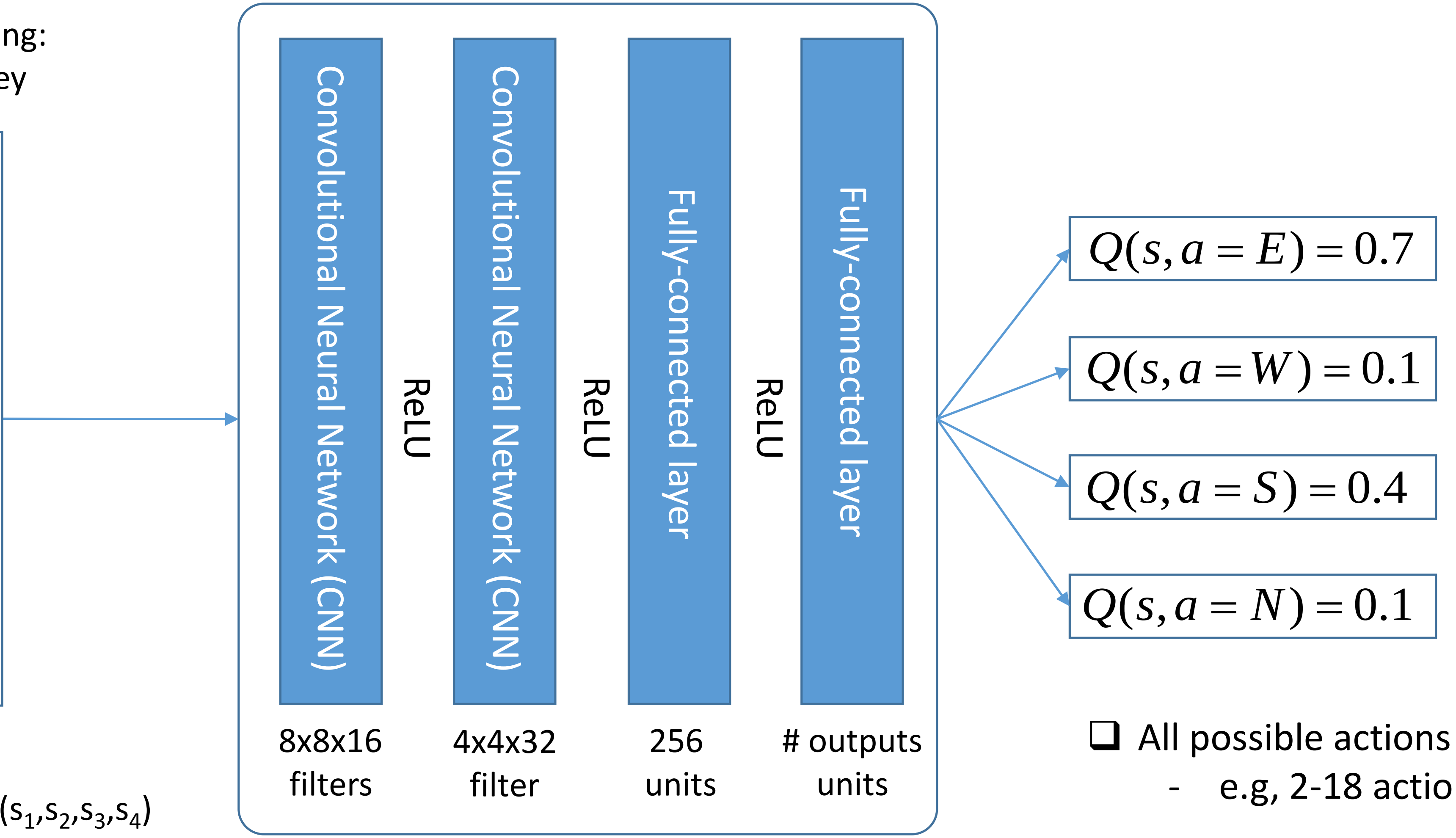
Deep Q-Network architecture (2013)

$\phi(s_t)$

Preprocessing:
RGB to grey

84x84x4

4 previous frames (s_1,s_2,s_3,s_4)
(e.g., Single frame cannot tell the movement of a ball)



Convolutional Neural Network (CNN)

ReLU

Convolutional Neural Network (CNN)

ReLU

Fully-connected layer

ReLU

Fully-connected layer

8x8x16 filters

4x4x32 filter

256 units

# outputs units

$Q(s, a = E) = 0.7$

$Q(s, a = W) = 0.1$

$Q(s, a = S) = 0.4$

$Q(s, a = N) = 0.1$

❑ All possible actions
- e.g, 2-18 actions

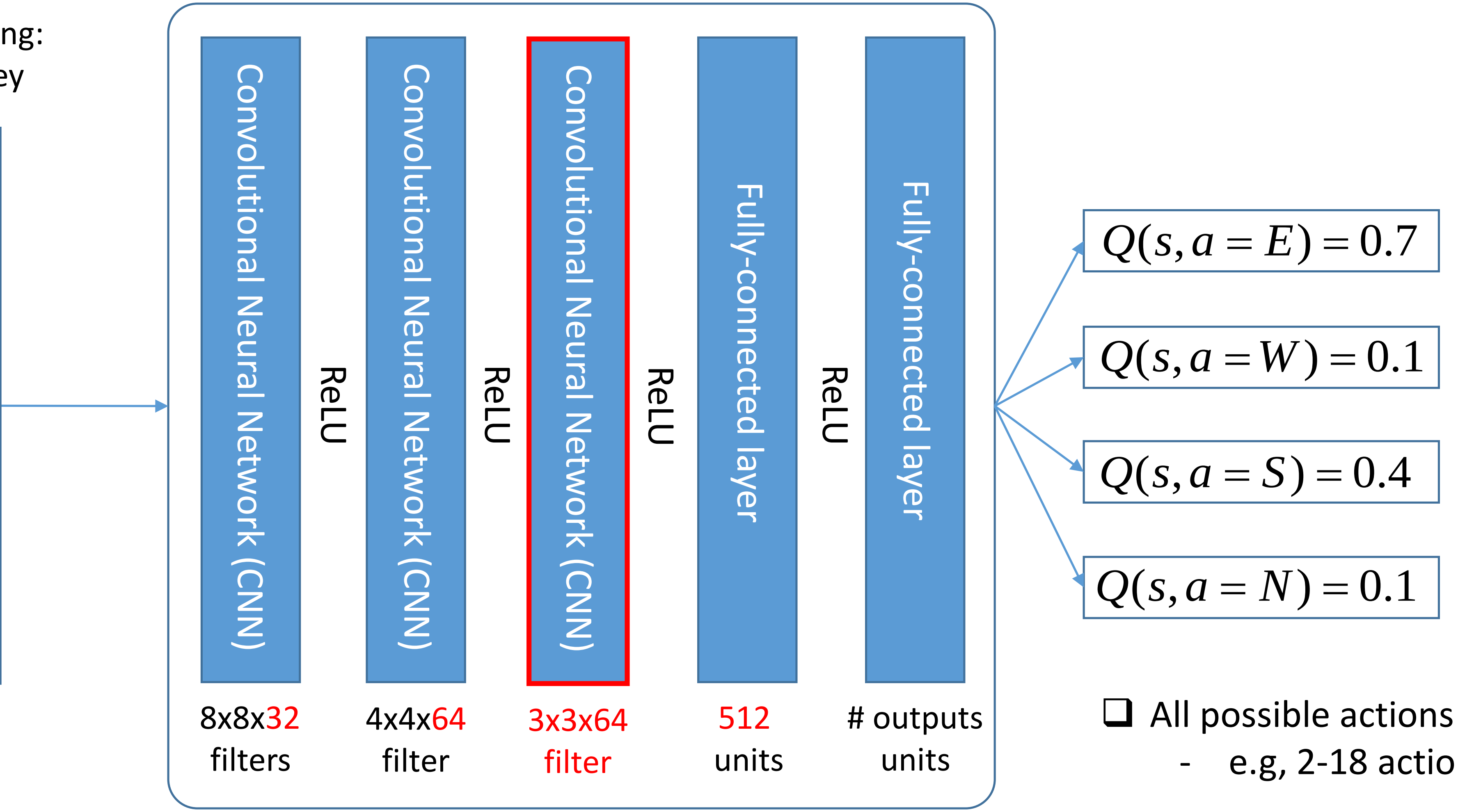
DQN

Deep Q-Network architecture (2015)

$\phi(s_t)$

Preprocessing:
RGB to grey

84x84x4



DQN

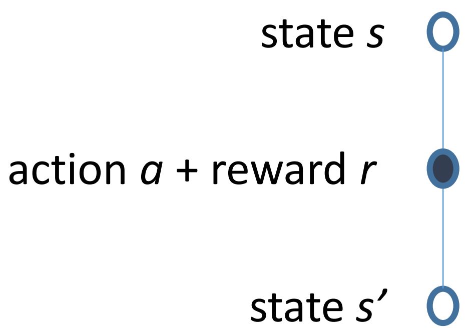
- ❑ All possible actions
  - e.g, 2-18 actions

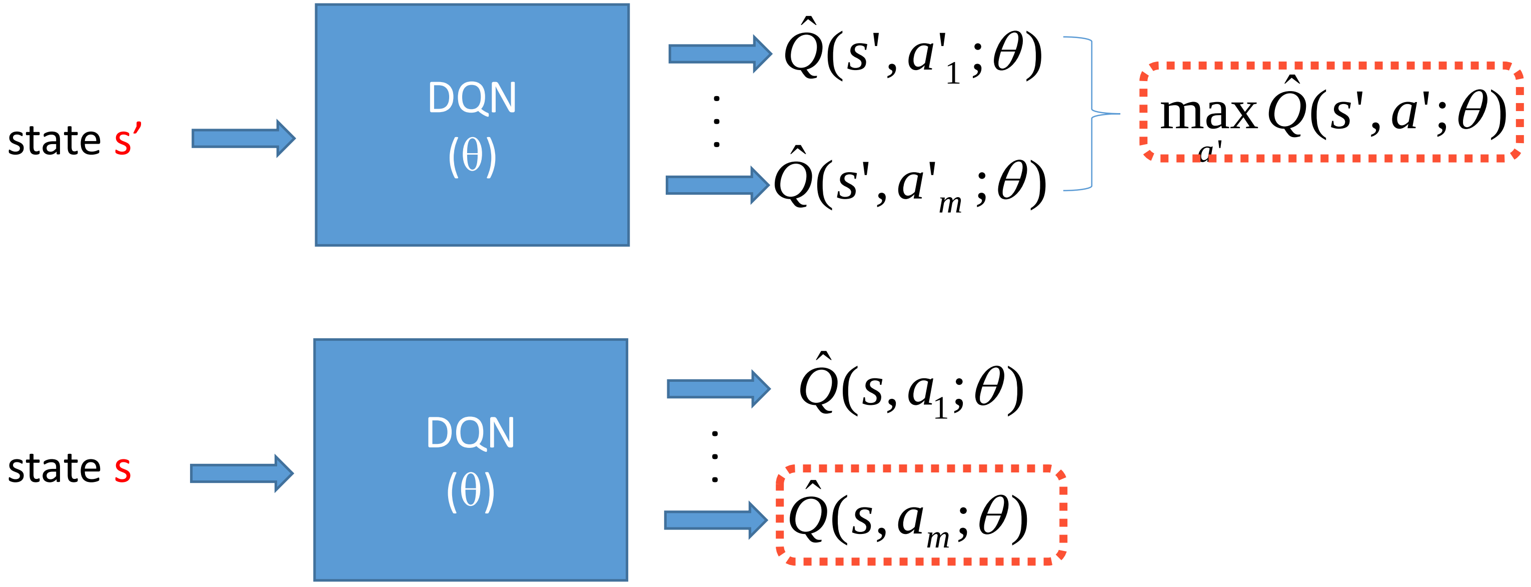
Loss function of the DQN

$$L_i(\theta_i) = \mathrm{E}_{\underbrace{(s,a,r,s')\sim U(D)}}\left[\left(\overbrace{r+\gamma \max_{a'} Q(s',a'|\boxed{\theta_i^-})}^{\text{Target future reward}} - \overbrace{Q(s,a,|\boxed{\theta_i})}^{\text{Expected future reward}}\right)^2\right]$$

- Sample data (s,a,r,s’) randomly drawn from data pool U(D)
- Experience Replay

- Two different neural networks
- Fixed Q-target





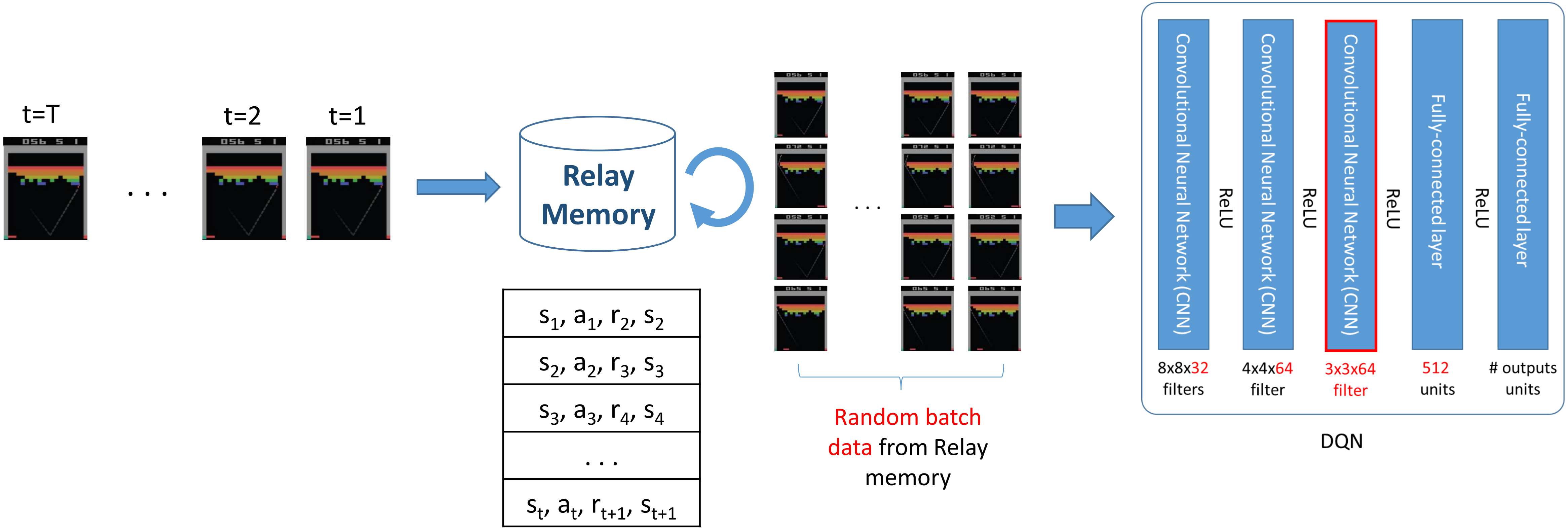
state s’
DQN
(θ)
$\hat{Q}(s',a'_1;\theta)$
$\hat{Q}(s',a'_m;\theta)$
$\max_{a'} \hat{Q}(s',a';\theta)$
state s
DQN
(θ)
$\hat{Q}(s,a_1;\theta)$
$\hat{Q}(s,a_m;\theta)$

Deep Q-Network architecture (2013 & 2015)

- ❑ Neural networks were used previously for RL
  - Temporal Difference Learning and TD-Gammon (1992)
  - Deep Auto-Encoder Neural Networks in RL (2010)
- ❑ However, they were not successful due to oscillates or divergence of neural nets
- ❑ How does DQN handle this problem?
  1) Experience replay
  2) Fixed Q-targets
  3) Go deep

1) Experience replay

- ❑ Consecutive data frames are highly correlated
- ❑ Experience replay aims to remove the correlation between data samples

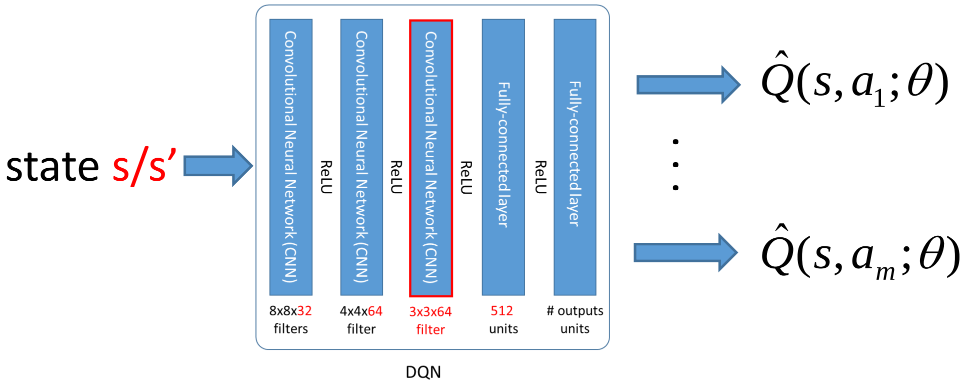


s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
. . .
$s_t, a_t, r_{t+1}, s_{t+1}$

2) Fixed Q-target

- ❑ Originally the target future reward and the expected future reward are sharing the same neural net.

$$L_i(\theta_i) = \mathrm{E}_{(s,a,r,s')\sim U(D)}\left[\left(r+\gamma \max_{a'} Q(s',a'|\overbrace{\theta_i) - Q(s,a,|\theta_i}^{\text{Same neural networks}})\right)^2\right]$$


state s/s’

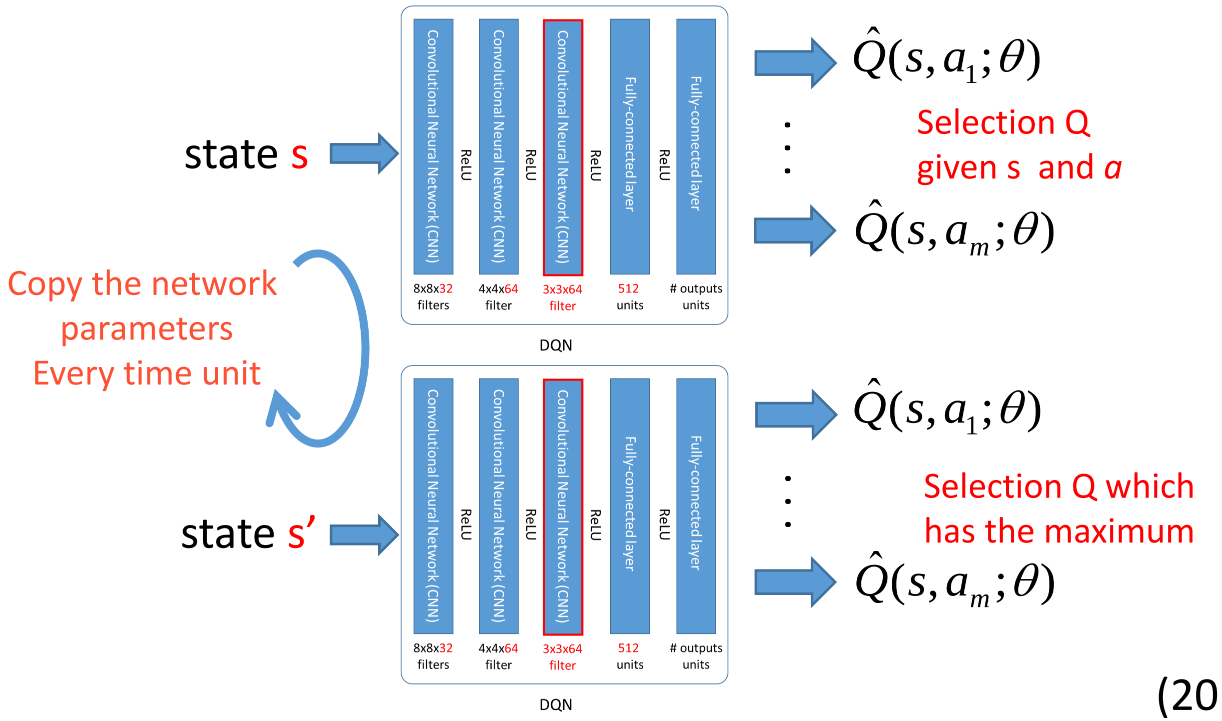
$\hat{Q}(s,a_1;\theta)$

$\vdots$

$\hat{Q}(s,a_m;\theta)$

(2013)

$$L_i(\theta_i) = \mathrm{E}_{(s,a,r,s')\sim U(D)}\left[\left(r+\gamma \max_{a'} Q(s',a'|\overbrace{\theta_i^-) - Q(s,a,|\theta_i}^{\text{Two different neural networks}})\right)^2\right]$$


state s

$\hat{Q}(s,a_1;\theta)$

$\vdots$

Selection Q given s and *a*

$\hat{Q}(s,a_m;\theta)$

Copy the network parameters Every time unit

state s’

$\hat{Q}(s,a_1;\theta)$

$\vdots$

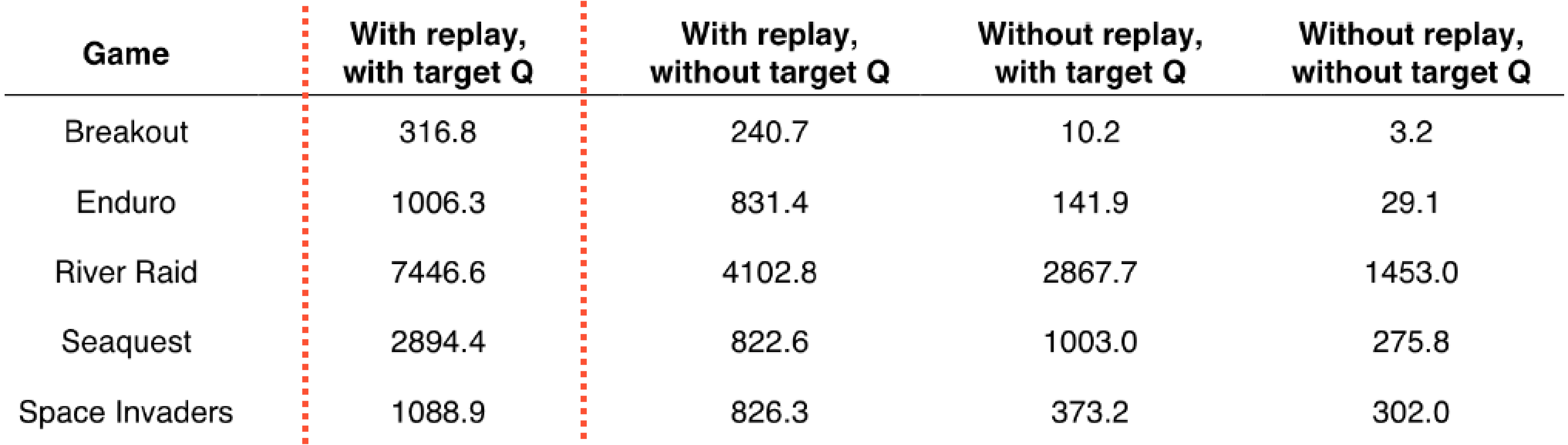
Selection Q which has the maximum

$\hat{Q}(s,a_m;\theta)$

(2015)

How effect they are?

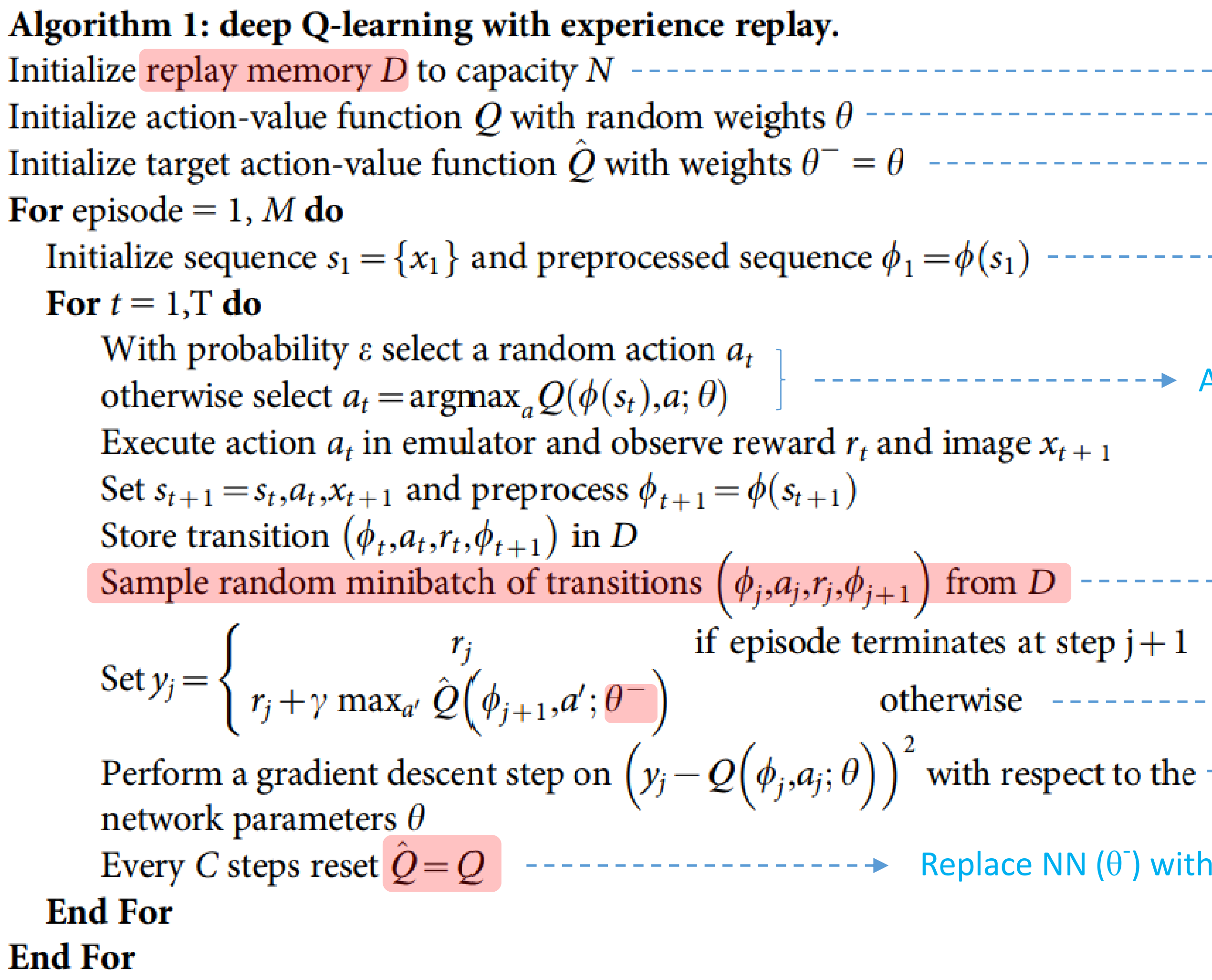
2015 DQN



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

Human-level control through deep reinforcement learning, Feb. 26, 2015, Nature

Algorithm (DQN 2015)



```
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, $M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t$ = 1,T do
      With probability $\varepsilon$ select a random action $a_t$
      otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
      Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
      Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
      Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
      Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
      Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
      Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
      network parameters $\theta$
      Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

Data pool size and initialization

Weight of 1st NN initialization

Weight of 2nd NN initialization

Preprocessing, e.g., RGB to gray

Action selection using E-greedy: off-policy

Experience replay

Target future reward is obtained from NN (θ^{-})

Update NN (θ) without changing NN (θ^{-})

Replace NN (θ^{-}) with NN (θ) every C steps

Human-level control through deep reinforcement learning, Feb. 26, 2015, Nature

Policy Gradient (PG)

Difference between DQN and Policy Gradient

❑ When we can approximate Q function for all states and action pairs, we can obtain the optimal π* by following way:

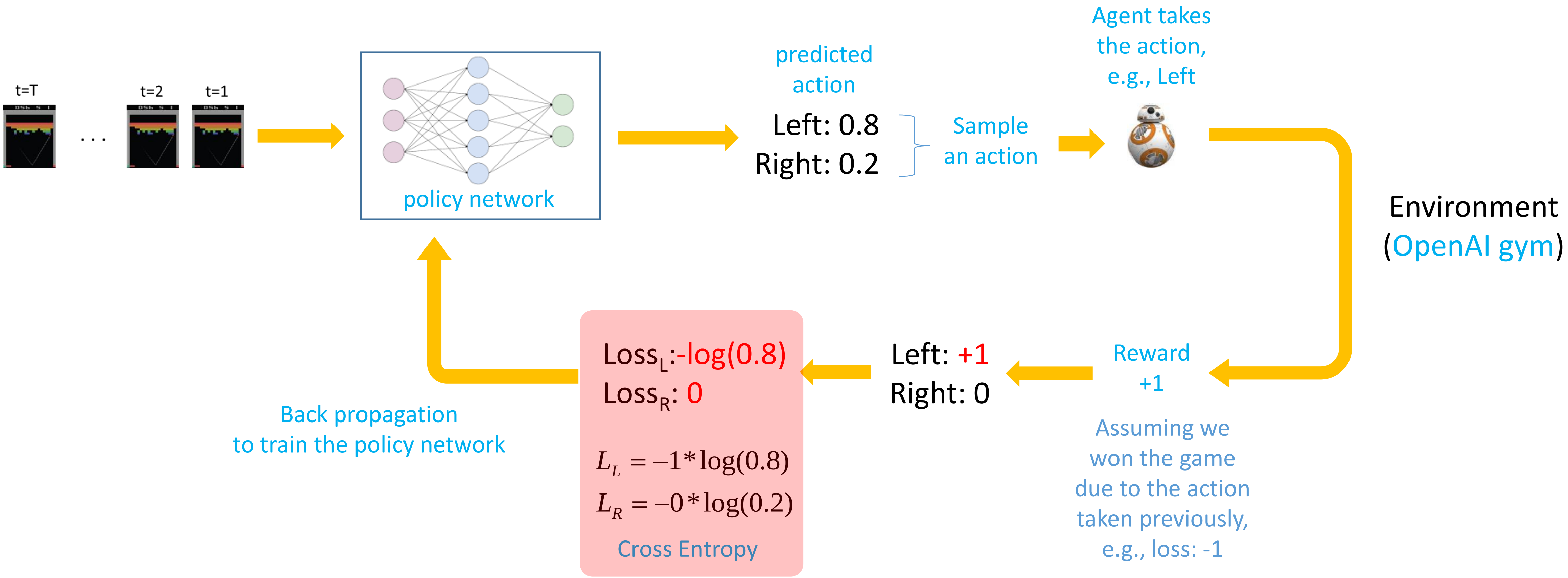
$$\pi^*(s) = \arg\max_a Q(s,a)$$: optimal policy

❑ Policy Gradient (PG) directly optimizes the policy function π without obtaining Q function.

- Similar to DQN, PG can also use a neural network (Policy Network): the output is the probability of each action at given state.

Loss function for Policy Network (PN): Cross Entropy

❑ First, let’s see the overall operation of policy gradient method



predicted action

Left: 0.8
Right: 0.2

Sample an action

Agent takes the action, e.g., Left

Environment (OpenAI gym)

Reward +1

Assuming we won the game due to the action taken previously, e.g., loss: -1

Left: +1
Right: 0

Loss$_L$:-log(0.8)
Loss$_R$: 0

$$L_L = -1 * \log(0.8)$$

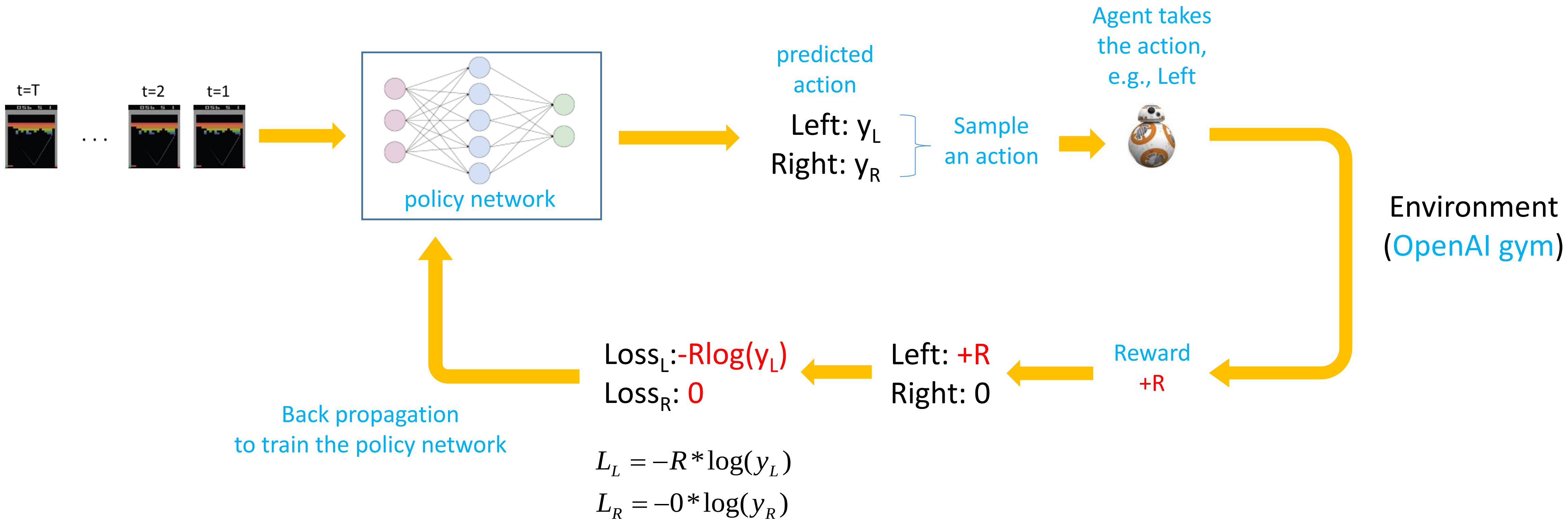
$$L_R = -0 * \log(0.2)$$

Cross Entropy

Back propagation to train the policy network

Loss function for Policy Network (PN): Derivation

- ❑ Let’s generalize the loss function by introducing Reward.



predicted action

Left: y_L

Right: y_R

Sample an action

Agent takes the action, e.g., Left

Environment (OpenAI gym)

Reward +R

Left: +R

Right: 0

Loss$_L$:-Rlog(y_L)

Loss$_R$: 0

Back propagation to train the policy network

$$L_L = -R * \log(y_L)$$

$$L_R = -0 * \log(y_R)$$

Cross Entropy

Loss function for Policy Network (PN): Derivation - cont

- PG aims to obtain an optimum policy which maximizes the future reward.
- In the previous slide, we want to train PN in a way that
  - When an agent follows the policy given by the outcome of the PN, it expects high future reward.

$$L_i(\theta_i) = \log(\pi_\theta(s,a)) \cdot R$$

- Expected future reward triggered by the sampled action

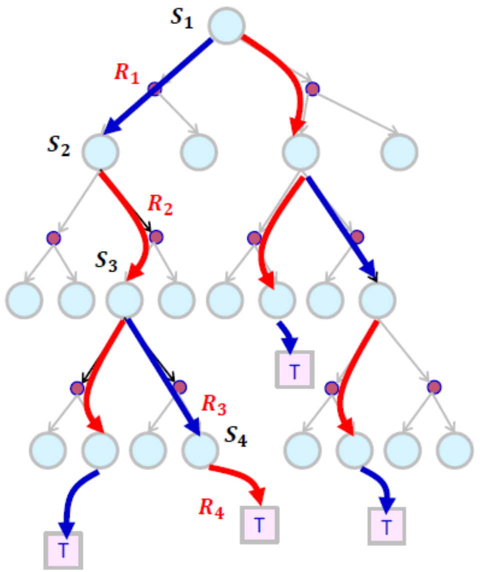
- An action is sampled from a policy
- NN models the policy distribution

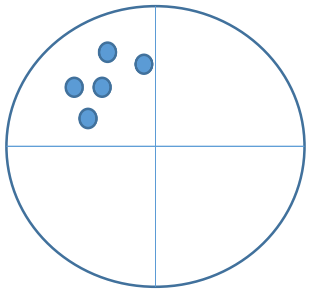
- "-" sign disappeared because we want to maximize the reward (good one has large reward)

- This equation says: the parameter θ of PN is updated by optimizing the policy π which maximizes future reward.

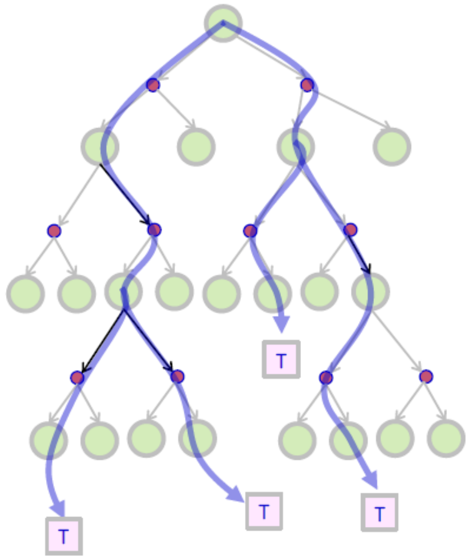
Q learning vs Policy gradient

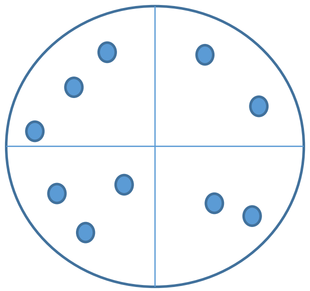
Q learning	Policy gradient
▪ Learning Q(s,a): modeling (Reward) values of actions - Value based approach: learning Q values	▪ Learning π(a): modeling probability of actions - Policy based approach: learning policy directly
▪ Deterministic policies: - e.g., cannot model rock-paper-scissors game	▪ Stochastic policies - e.g., can model rock-paper-scissors game
▪ Off-policy: an action is taken greedily - Greed search to calculate Q(s,a) and then determine a policy	▪ On-policy: an action is taken with a policy - Following a trajectory created by a policy and update it with given reward at the end.
▪ Learning update occurred step-by-step (bootstrapping) - Low variance but high bias	▪ Learning update occurred episode-by-episode - High variance but low bias





High bias





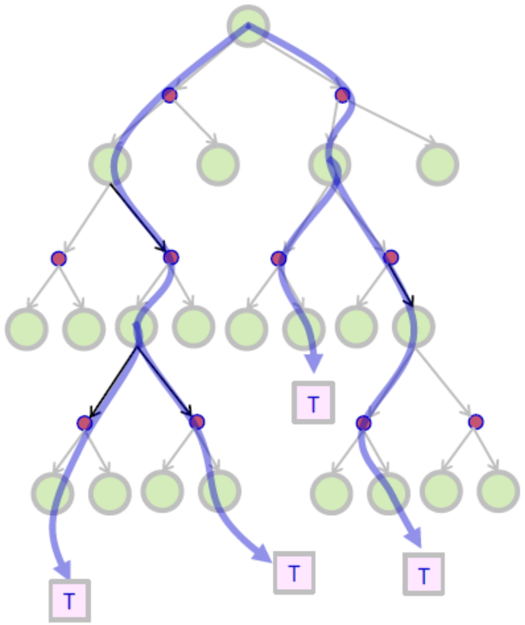
High variance

Action Critic (AC)

Action critic = Q learning + Policy gradient

- ❑ It aims to deal with following two problems in PG.
  1) PG uses episode-by-episode learning update, which disables on-line learning.
  2) PG tends to produce a policy with high variance.

Policy Gradient approach
Monte Carlos (MC) Learning
Episode-by-episode

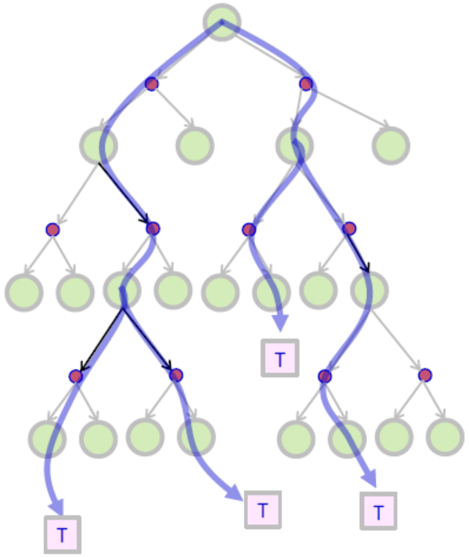


1) Learning update step-by-step: Policy gradient theorem

- ❑ For the first problem, let's change the reward function *R* to Q(s, a) function and so learning update can be done step-by-step, which enables on-line learning.
  - Proof in "*policy gradient methods for reinforcement learning with function approximation*"

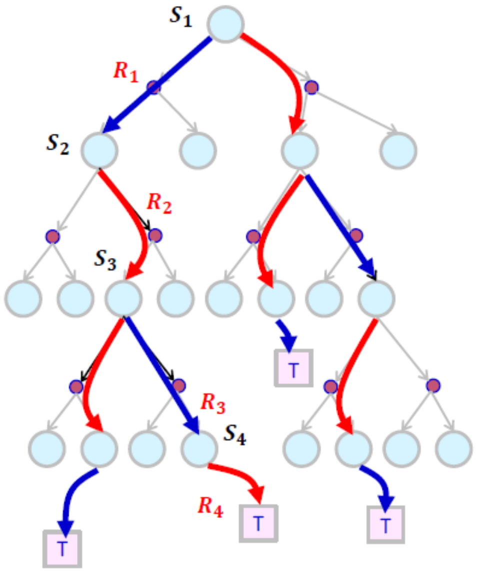
Monte Carlos (MC) learning

$$L_i(\theta_i) = \log(\pi_\theta(s,a)) \cdot R$$



Temporal-Difference (TD) learning

$$L_i(\theta_i) = \log(\pi_\theta(s,a)) \cdot Q(s,a)$$

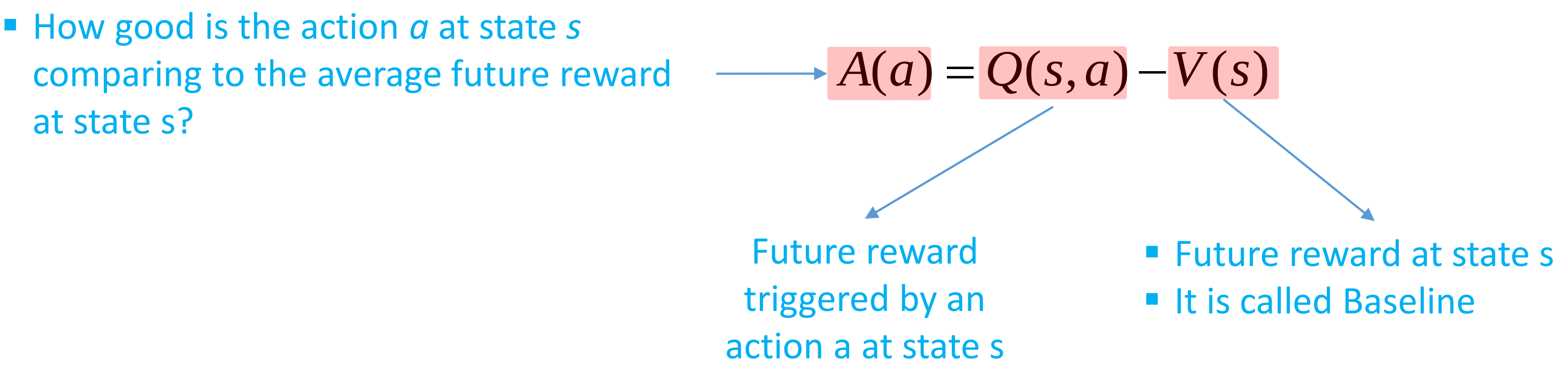


https://dl.acm.org/citation.cfm?id=3009806

2) Reducing variance

❑ For the second problem, let’s introduce “advantage” which replaces “Q function”

❑ Well, it is a kind of normalization process if you see its definition below.



- How good is the action *a* at state *s* comparing to the average future reward at state s?

$$A(a) = Q(s,a) - V(s)$$

Future reward triggered by an action a at state s

- Future reward at state s
- It is called Baseline

$$L_i(\theta_i) = \log(\pi_\theta(s,a)) \cdot Q(s,a) \quad \Longrightarrow \quad \begin{aligned} L_i(\theta_i) &= \log(\pi_\theta(s,a)) \cdot A(a) \\ &= \log(\pi_\theta(s,a)) \cdot (Q(s,a) - V(s)) \\ &= \log(\pi_\theta(s_t,a_t)) \cdot (r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \end{aligned}$$

Actor-Critic

- ❑ Actor-Critic is a policy gradient approach which updates a policy in each step
  1) Actor determines a policy
  2) Critic determines a value function for future reward

$$L_i(\theta_i) = \log(\pi_\theta(s_t, a_t)) \cdot (r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

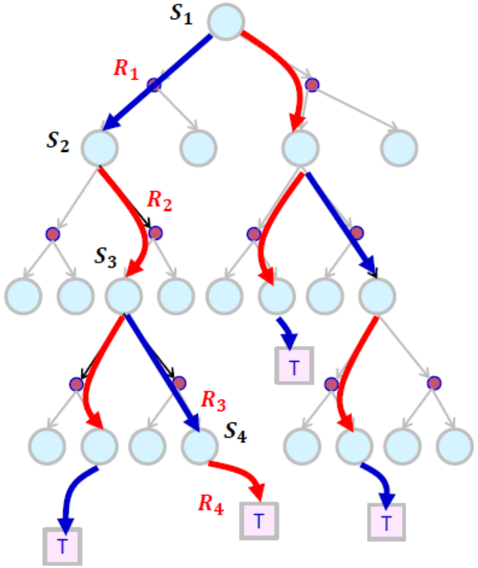
- We call it "Critic" because it *critically(?)* evaluates how good the action taken.
- V(s) needs to be found to calculate this part
- A neural network can be used to approximate this value

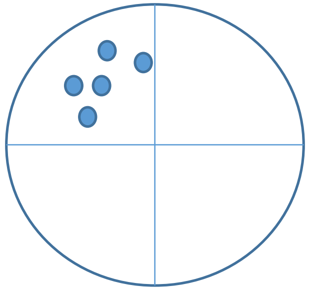
- We call it "Actor" because it determines its action policies!
- Policy needs to be determined based on the result from Critic
- A neural network can be used to approximate the policy

A3C: Asynchronous Advantage Actor-Critic

Two problems in Actor Critic

- ❑ High bias due to every one step update
- ❑ Exploration issue
    - DQN uses e-greedy approach to handle exploration issue.
    - However, policy gradient approach; Actor Critic does not have the mechanism.
    - Stochastic behavior of a policy function can handle the exploration issue partially.





High bias

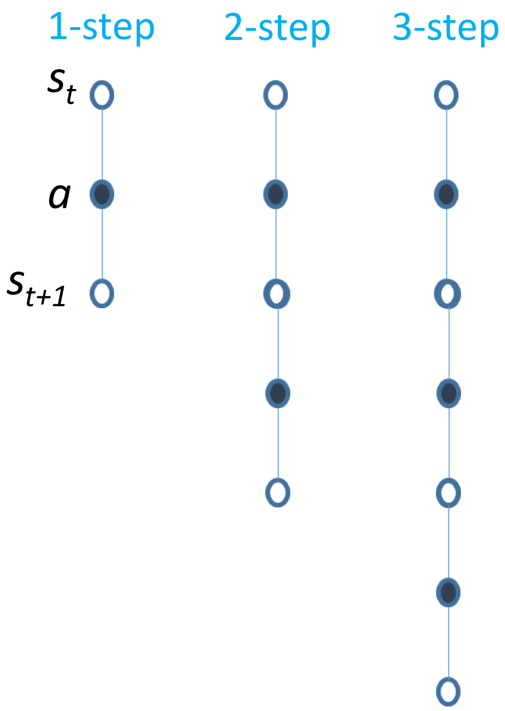
1) High Bias every one step update

- ❑ A3C introduces multi step updates to handle the problem
- ❑ There can be several variations!

$$L(\theta) = \log(\pi_\theta(s_t, a_t)) \cdot (r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

$$L(\theta) = \log(\pi_\theta(s_t, a_t)) \cdot (r_{t+1} + r_{t+2} + \gamma V(s_{t+2}) - V(s_t))$$

$$L(\theta) = \log(\pi_\theta(s_t, a_t)) \cdot (r_{t+1} + r_{t+2} + r_{t+3} + \gamma V(s_{t+3}) - V(s_t))$$



$$L(\theta) = \log(\pi_\theta(s_t, a_t)) \cdot (r_{t+1} + r_{t+2} + r_{t+3} + \gamma V(s_{t+3}) - V(s_t))$$

$$+ \log(\pi_\theta(s_t, a_t)) \cdot (r_{t+2} + r_{t+3} + \gamma V(s_{t+3}) - V(s_t))$$

$$+ \log(\pi_\theta(s_t, a_t)) \cdot (r_{t+3} + \gamma V(s_{t+3}) - V(s_t))$$

2) Exploration issue

❑ A3C includes “entropy of the policy π” to the loss function in order to improve exploration by discouraging premature convergence to suboptimal deterministic policies.

$$L(\theta) = \log(\pi_\theta(s_t, a_t)) \cdot (r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) + \beta H \pi_\theta(s_t, a_t)$$

- This term defines the probability distribution of actions at each state

- Entropy regularization term
- This term tries to uniformize the probability distribution of actions defined in the first term.
    - Entropy is maximized when all actions from the policy π are same.
    - It aims to occur all action with equal probability (exploration)

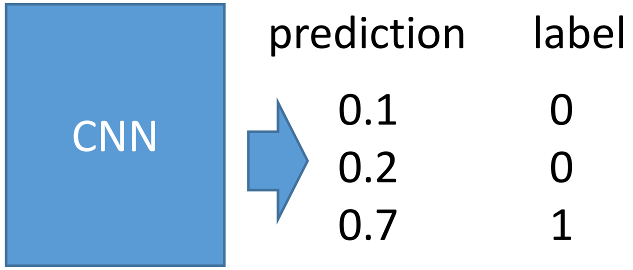
Backup Slides

Cross Entropy

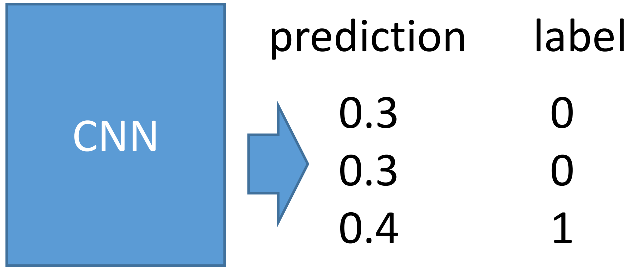
❑ Do you remember the Cross Entropy which is used to calculate the loss in CNN?

- prediction: predicted label which is the output from the previous layer
  - e.g., [0.1, 0.2, 0.7]
- label: true label, one-hot encoded
  - e.g., [0,0,1]

$$H(p,q) = -\sum_{x} \underbrace{p(x)}_{\text{label}} \underbrace{\log q(\hat{x})}_{\text{prediction}} = -\log q(\hat{x}_{x \neq 0})$$



prediction	label
0.1	0
0.2	0
0.7	1



prediction	label
0.3	0
0.3	0
0.4	1

-0*log(0.1)-0*log(0.2)-1*log(0.7) = 0.375

-0*log(0.3)-0*log(0.3)-1*log(0.4) = 0.916

Good one has small error