REINFORCEMENT LEARNING

Shingchern D. You

Introduction to RL

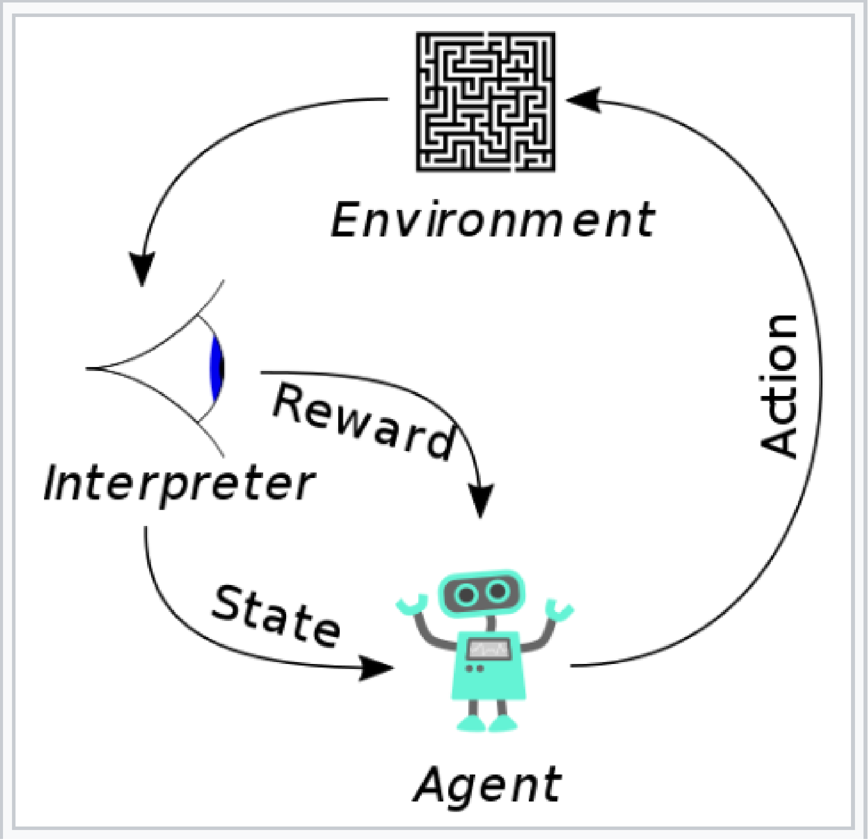
- One word of warning: RL is not a simple toy & may not be best solution to most problems
- Ref: Deep Reinforcement Learning Doesn't Work Yet (https://www.alexirpan.com/2018/02/14/rl-hard.html)
- Here I quote one sentence: **If You Just Care About Final Performance, Many Problems are Better Solved by Other Methods**

Introduction to RL

- Supervised learning needs "labeled data" for training
- Unsupervised learning performs clustering without additional knowledge
- Reinforcement learning (RL) uses "rewards" for learning
- One well-known example of RL is **Alpha-GO**

Introduction to RL

- Typical scenario (picture credit: Wikipedia)



Introduction to RL

- Agent has sensors to sense the states of environment
  - State example: agent in room 1,..,5
- Agent can perform actions
- Agent received a reward for each action (though reward can be zero)

Classification of RL

- Model free RL
  - Model free means no specific (built-in) environmental model used during training
  - Well known algorithms: Q learning, SARSA, policy gradients
- Model-based RL
  - Model-based method contains a virtual environmental model

Classification of RL

- Policy-based RL
  - The agent's action selection is modeled as a map called policy
  - Gives probability of taking action a when in state s
  - There are also non-probabilistic policies
  - Algorithm: policy gradients
- Value-based RL
  - Produce a value for each action
  - Algorithm: Q learning, SARSA
  - Not work for continuous actions

Classification of RL

- On-policy RL
  - Agent must be present to learn
  - Algorithm: SARSA
- Off-policy RL
  - Agent can learn from experiences of any one
  - Q learning

Classification of RL

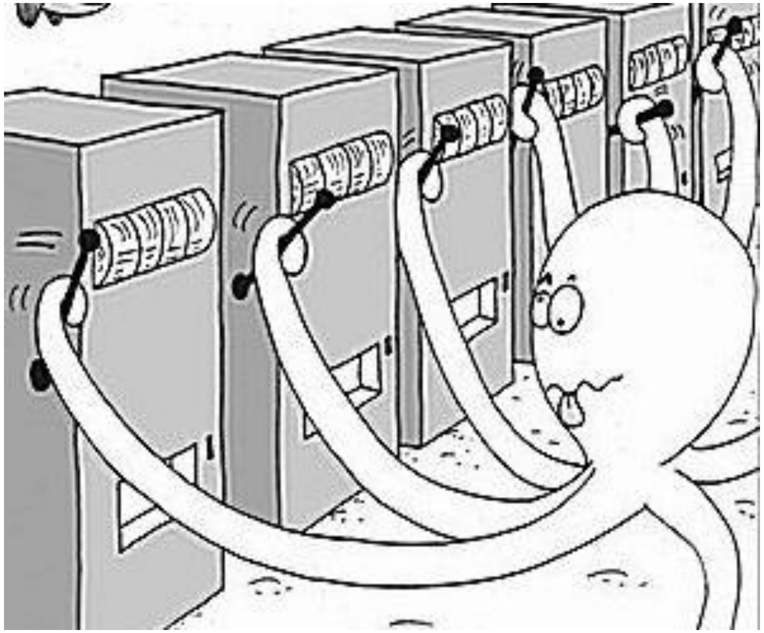
- Monte-Carlo updating
  - Rewards received at the end of one session or episode (e.g., one game of chess)
  - Algorithm: Monte-Carlo learning
- Temporal difference update
  - Update estimate after each action or time step (e.g., one move in a chess game)
  - Algorithm: Q learning

Classification of RL

- Episodic task
  - There is a terminal state and the task can terminate
- Continuous task
  - Task continues forever (no terminal state)

Exploration & exploitation

- Multi-armed bandit problem
- One-armed bandit = slot machine (Picture credit: https://blogs.mathworks.com/loren/2016/10/10/multi-armed-bandit-problem-and-exploration-vs-exploitation-trade-off/)



Exploration & exploitation

- Exploration & exploitation
  - Exploration: Random action to explore the environment
  - Exploitation: Best use of previously learned knowledge to max rewards
- Trade off in a multi-armed bandit problem
  - Doing exploration miss the chance to earn more from previous knowledge
  - Doing exploitation miss the chance to know another arm may have better rewards

Markov decision process

- Similar to Markov chain but with action and reward
- Recall 1^{st}-order Markov chain
- In a chess game, next move depends only on present checkerboard status

Markov decision process

- Ref: wiki
- A set of environment & agent sets, S
- A set of agent actions, A
- State transition $T(s, a, s')$
- $R(s, a, s')$ is immediate reward from s to s' by taking action a
- Start state
- Terminal state

Markov decision process

- In the following, we sometimes use short notations
  - State $s_t = s$, $s_{t+1} = s'$
  - In s_t, if action a_t is taken & clock ticks, reward R_{t+1} is received and state changes to s_{t+1}
  - Therefore, $R(s, a, s') = R_{t+1} = R$

Multi-armed bandit

- Want to compute average reward q_{t+1} after observing rewards $R_1, \ldots, R_{t+1}$

$$q_{t+1} = \frac{1}{t+1} \sum_{i=1}^{t+1} R_i$$

- But we can write

$$q_t = \frac{1}{t} \sum_{i=1}^{t} R_i$$

Multi-armed bandit

- With a couple steps of derivations, we have

$$q_{t+1} = q_t + \frac{1}{t+1}(R_{t+1} - q_t)$$

- In practical case, we may use the following for on-line learning

$$q_{t+1} \leftarrow q_t + \eta(R_{t+1} - q_t)$$

where η is called learning rate

Bellman equation

- How about the expected rewards in the future

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

- But then $G_k \to \infty$ if $T \to \infty$
- We need a discount rate: γ (gamma)

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

$$= \sum_{i=0}^{T-t-1} \gamma^i R_{t+1+i}$$

Bellman equation

- $V_\pi(s_t)$: expected value of G at states_t with policy π
- Conceptually (not using probability), performing action a_t at state s_t will receive future rewards as

$$V_\pi(s_t) = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$
$$= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots)$$
$$= R_{t+1} + \gamma V_\pi(s_{t+1})$$

where s'=s_{t+1} is next state

Bellman equation

- Therefore, the max $V_\pi(s_t)$ for all π, denoted as $V^*(s_t)$, can be obtained as

$$V^*(s_t) = \max_{a_t} E[R_{t+1} + \gamma V^*(s_{t+1})]$$

- This equation is known as Bellman equation

Temporal difference learning

- Recall we have

$$q_{t+1} \leftarrow q_t + \eta(R_t - q_t)$$

and

$$V_\pi(s_t) = R_{t+1} + \gamma V_\pi(s_{t+1})$$

- By combining these two equations, we have

$$V(s_t) \leftarrow V(s_t) + \eta[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

- This equation is TD (temporal difference) learning

Meanings of symbols

- V is a function of state only (state value)
- However, we know that actions affect rewards
- Define Q as a function of state and action (action value)
- $V^*(s_t) = \max_{a_t}(Q^*(s_t, a_t))$
- Use Q in place of V in TD learning, we have SARSA (State–action–reward–state–action) algorithm

Q learning

- If we use action value Q in the Bellman equation, we have Q learning

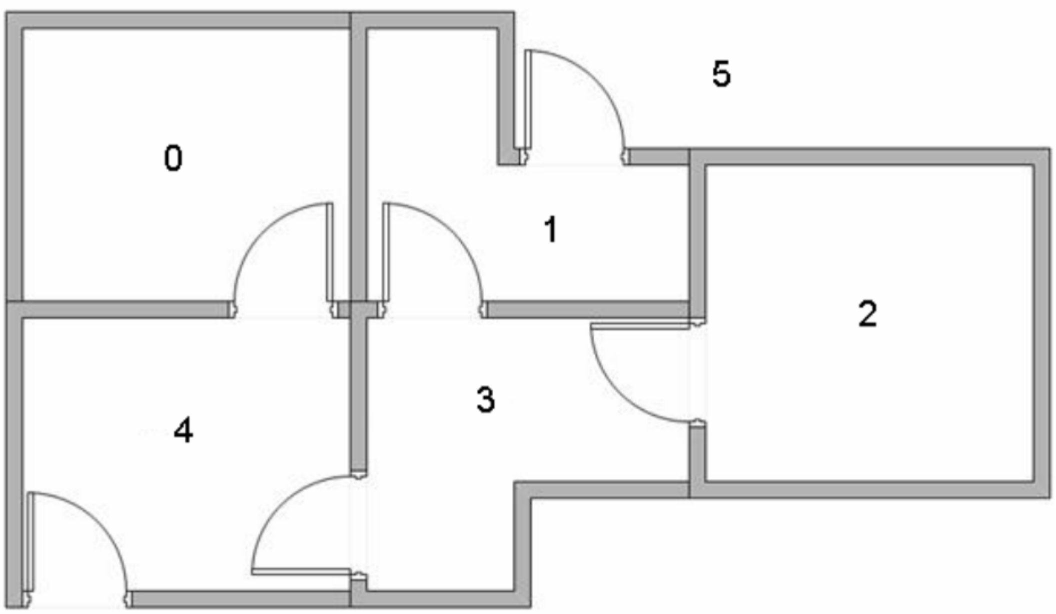
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left[R_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Q learning algorithm

- Initialize all $Q(s, a)$ arbitrarily
- For each episode
  - Initialize state s
  - Repeat
    - Choose a using policy derived from Q, e.g., ϵ-greedy
    - Take action a and observe reward R and next state s'
    - $Q(s, a) \leftarrow Q(s, a) + \eta \left[R + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    - $s \leftarrow s'$
  - Until s is terminal state

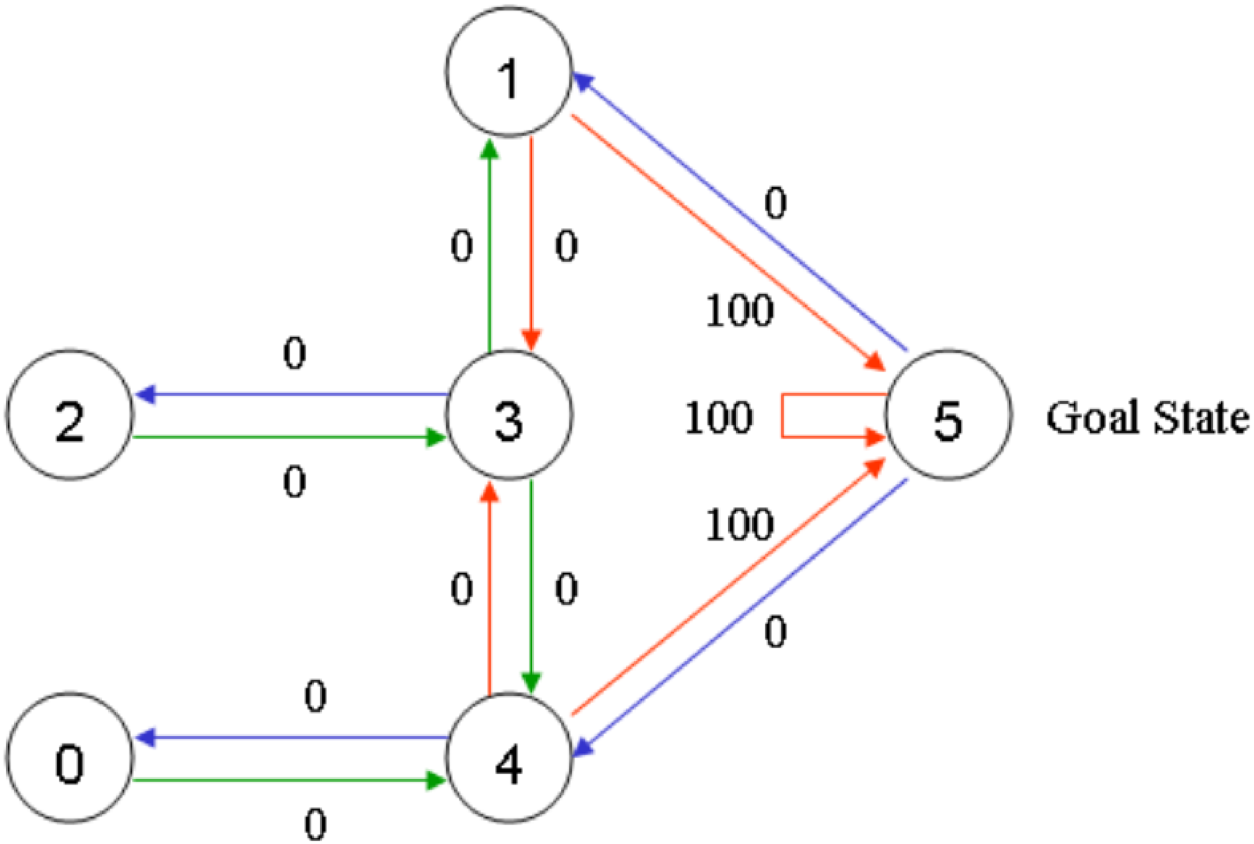
Numerical example of Q learning

- From: http://mnemstudio.org/path-finding-q-learning-tutorial.htm
- A building with four rooms (outside space is room 5)
- Want to escape out of the rooms (big reward!)



Numerical example of Q learning

- Represented in graph with assigned rewards



Numerical example of Q learning

- Basic assumption of RL: Reward R available for every action (no matter how)
- In this example, one action means "moving from one room to adjacent next room"
- If agent can sense environment (in a room or outside space), it is OK to assume rewards available
- **If Q learning is to be used for playing video games, how does the agent get rewards?**

Numerical example of Q learning

- Init Q table
- $$Q = \begin{array}{cc} & \text{Action } (0,..,5) \\ \text{State } (0,..,5) & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Numerical example of Q learning

- Episode 1 (step 1):
- Initial state (randomly chosen): s = room 1
- Action: randomly choose a = 5 (going to room 5 by exploration)
- Update
- $Q(s,a) \leftarrow Q(s,a) + \eta \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$
- Let $\eta = 0.1, \gamma = 0.8$

Numerical example of Q learning

- $Q(s,a)$ = Q(1,5) = 0
- R(in room 1 take action 5) = R(1,5) = 100
- $\max_{a'} Q(s',a')$ means to find the largest Q value for all possible action with state s'
  - Taking action 5, s' = 5
  - At state 5, possible actions are 1, 4, 5
  - Max {Q(5,1), Q(5,4), Q(5,5) } = 0

Numerical example of Q learning

- $Q(1,5) \leftarrow 0 + 0.1 * (100 + 0.8 * 0 - 0)$
- Therefore, $Q(1,5) = 10$
- New Q table is

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- **As s = 5 is terminal state, episode 1 ends**

Numerical example of Q learning

- Episode 2 (step 1):
- Initial state: s = room 3
- Action: randomly choose a = 1 (going to room 1 by exploration)
- R(3,1) = 0 (room 3 to take action 1)
- Present $Q(s,a)=Q(3,1)=0$

Numerical example of Q learning

- $\max_{a'} Q(s', a')$ means to find the largest Q value for all possible action with state s'
  - Taking action 1, s' = 1
  - At state 1, possible actions are 3, 5
  - Max {Q(1,3), Q(1,5) } = max(0,10)=10
- Therefore, $Q(3,1) = 0 + 0.1 * (0 + 0.8 * 10 - 0) = 0.8$

Numerical example of Q learning

- New Q table is

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- We can proceed to do more steps and more episodes

Q-learning and SARSA

- Q-learning is an off-policy learning because we update Q with $\max_{a'} Q(s', a')$
  - But, the agent may not choose action a' in next move (e.g., in episode 2, Q(3,1) is updated without actually taking action 5 in room 1)
- We mentioned SARSA before
- SARSA is an on-policy learning
  - Update Q entries only if agent takes the actions

SARSA learning algorithm

- Initialize all $Q(s,a)$ arbitrarily
- For each episode
  - Initialize state s
  - Choose a using policy derived from Q, e.g., ϵ-greedy
  - Repeat
    - Take action a, observe R and s'
    - Choose a' from s' using policy derived from Q, e.g., ϵ-greedy
    - $Q(s,a) \leftarrow Q(s,a) + \eta[R + \gamma Q(s',a') - Q(s,a)]$
    - $s \leftarrow s'$, $a \leftarrow a'$ // a' will be used in next move
  - Until s is terminal state

Q learning and SARSA

- Update in SARSA
  - Agent start in state 1, perform action 1 determined in previous iteration, and get reward 1
  - Now agent in state 2, determine action 2 and get reward 2
  - Update Q of action 1 performed in state 1
- Update in Q-learning
  - Agent start in state 1, perform action 1, and get reward 1
  - Look and see the maximum possible reward for all actions in state 2
  - Use max reward to update Q for action 1 in state 1

Q learning and SARSA

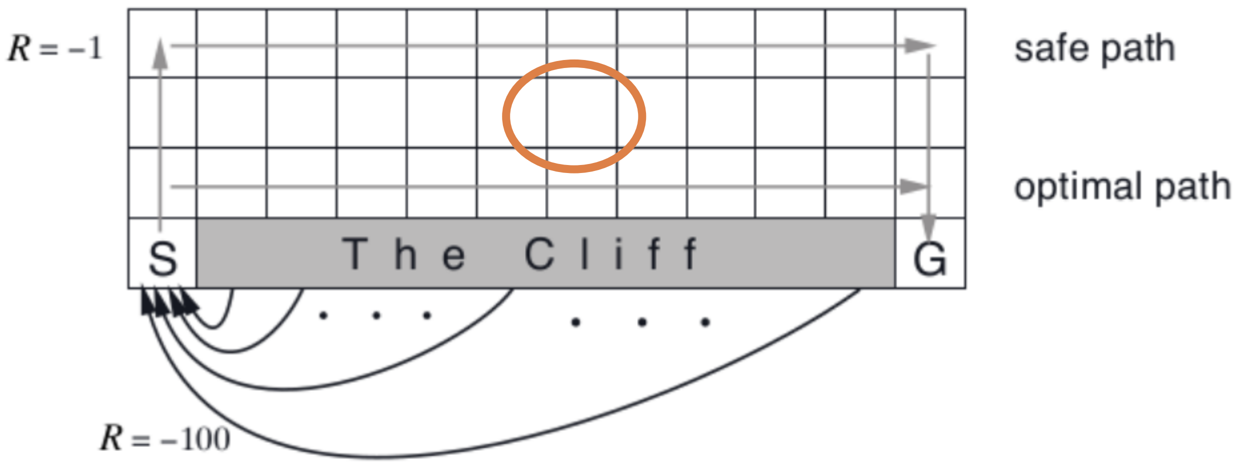
- Difference is in the way the future reward is found
  - In Q-learning it's simply the highest possible action that can be taken from state 2
  - In SARSA, it's the value of the actual action that will be taken
- In Q-learning, next action to perform in next iteration may not equal to a' in max $Q(s', a')$ used to update Q because of ϵ-greedy (some randomness)

Q learning and SARSA

- SARSA take into account the control policy by which the agent is moving, and incorporate that into its update of action values
- Q-learning simply assumes that an optimal policy is being followed
- A good article to explain the difference (with demo) is at https://studywolf.wordpress.com/2013/07/01/reinforcement-learning-sarsa-vs-q-learning/

Q learning and SARSA

- Example of mouse, cheese, and cliff



Q-learning update

- Present state is s_k, action is south, & Reward $= 0$
- s_m has action values (E,W,N,S) of (0,0,0,-100)
- Therefore Q(s_k,south) ← 0 (let $\eta = 1$)

	s_k	
	s_m	
	Cliff	

Q-learning update

- It seems safe to go to s_m
- But, when the mouse is in s_m, due to exploration, it has some chance $(1/4)$ to fall in cliff

SARSA update

- Present state is s_k, action is south, & Reward = 0
- s_m has action values (E,W,N,S) of (0,0,0,-100)
- Taking action of south (by exploration)
- Therefore Q(s_k,south) ← -90 (let $\eta = 1, \gamma = 0.9$)
- During exploitation, going south at state s_k will be discouraged (due to negative Q value)
- Will not fall in cliff by one step of exploration
- **This EX does not mean SARSA is always better**

Deep Q learning

- Sometimes we may have too many states
  - For example, playing video games
  - How may states do we have
- Can we use CNN to replace Q table
  - CNN produces Q values for all actions given present state
  - Can use screen shots as inputs to CNN
  - However, we still need rewards (**not mentioned in most papers how to get it**)

Deep Q learning references

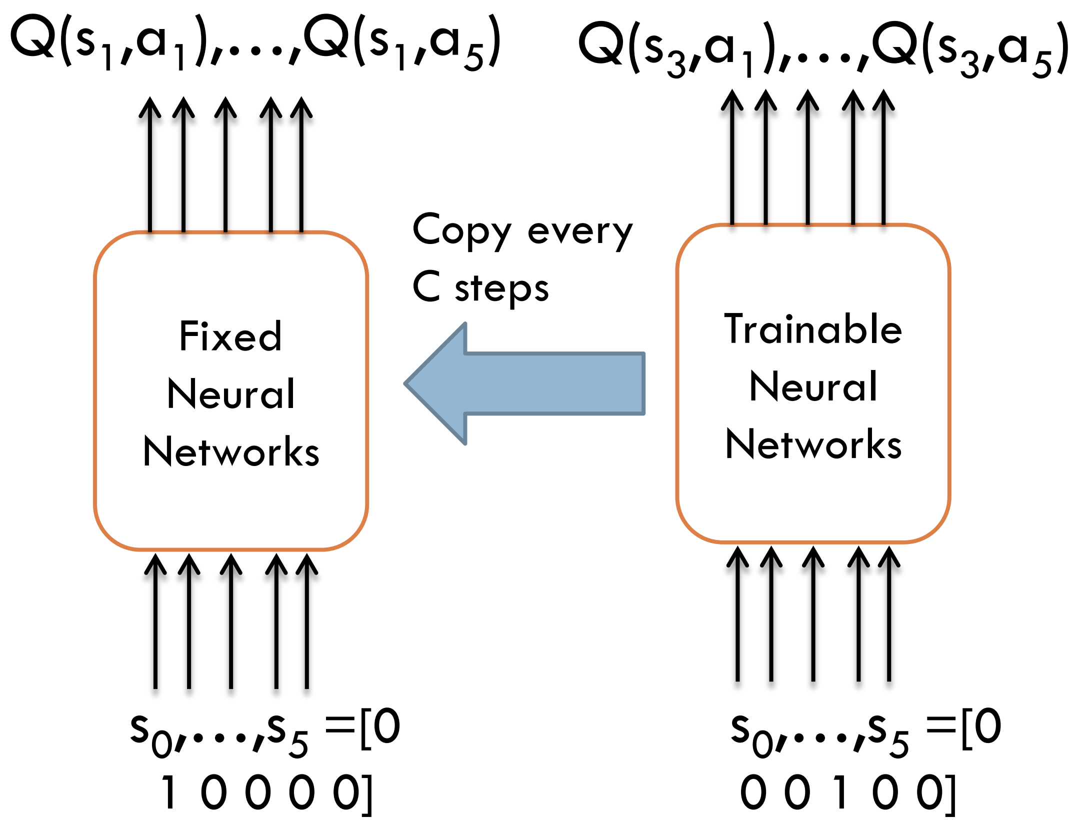
- DeepMind paper: https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
- https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/4-3-DQN3/ (in Chinese)

Deep Q learning concept

- Use CNN to replace Q table
- Need to overcome some problems
  - Use two sets of neural networks to avoid learning problems (**fixed Q targets**)
  - Avoid forgotten old knowledge with **experience replay** (stored previous episodes in buffer and randomly replay)

Deep Q learning concept

- Use NN in place of Q in our previous example



Deep Q learning concept

- How to train "trainable NN"
- We need desired (target) output & a loss function
- Loss function is mean-squared error
- If $\eta = 1$, target output become $R + \gamma \max_{a'} Q(s', a')$
  - Reward R is from somewhere (assuming always available)
  - $Q(s', a')$ is from "fixed NN"

Deep Q learning concept

- For example, if agent is in room 3
  - Input to trainable NN is [0 0 0 1 0 0]
  - Suppose O/P from trainable NN is [0 0.2 0 0.1 0.1 0]
  - In the exploitation mode agent will choose action 1 (i.e., go to room 1)
  - Obtain R (R = 0 in this example)
  - Store necessary info for experience replay later

Deep Q learning concept

- Perform backprop (no random mini-batch selection to simplify discussion)
  - Agent in state 1 now
  - Use fixed NN with input [0 1 0 0 0 0] (find Q for room 1)
  - Suppose O/P of fixed NN is [0 0.2 0.3 0 0.1 1.0]
  - Compute $R + \gamma \max_{a'} Q(s', a') = 0.99$ (γ set to 0.99)
  - Target output = [0 0.99 0 0 0 0] --- symbol in paper is y
  - Modify the output of trainable NN as [0 0.2 0 0 0 0] = ϕ --- in the paper
  - Error =$(y - \phi)^2$

Deep Q learning concept

- Why do we want to modify target output and actual output before computing MSE
  - Recall what we did in Q learning
    $$Q(3,1) \leftarrow 0 + 0.1 * (0 + 0.8 * 10 - 0) = 0.8$$
  - We only update one entry in Q table
  - In CNN, state is input and action value is output
  - Therefore, we should only update this particular state-action combination
  - So, other outputs are set to zero (not to learn)

Deep Q learning algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

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 ε select a random action a_t

    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

    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 $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

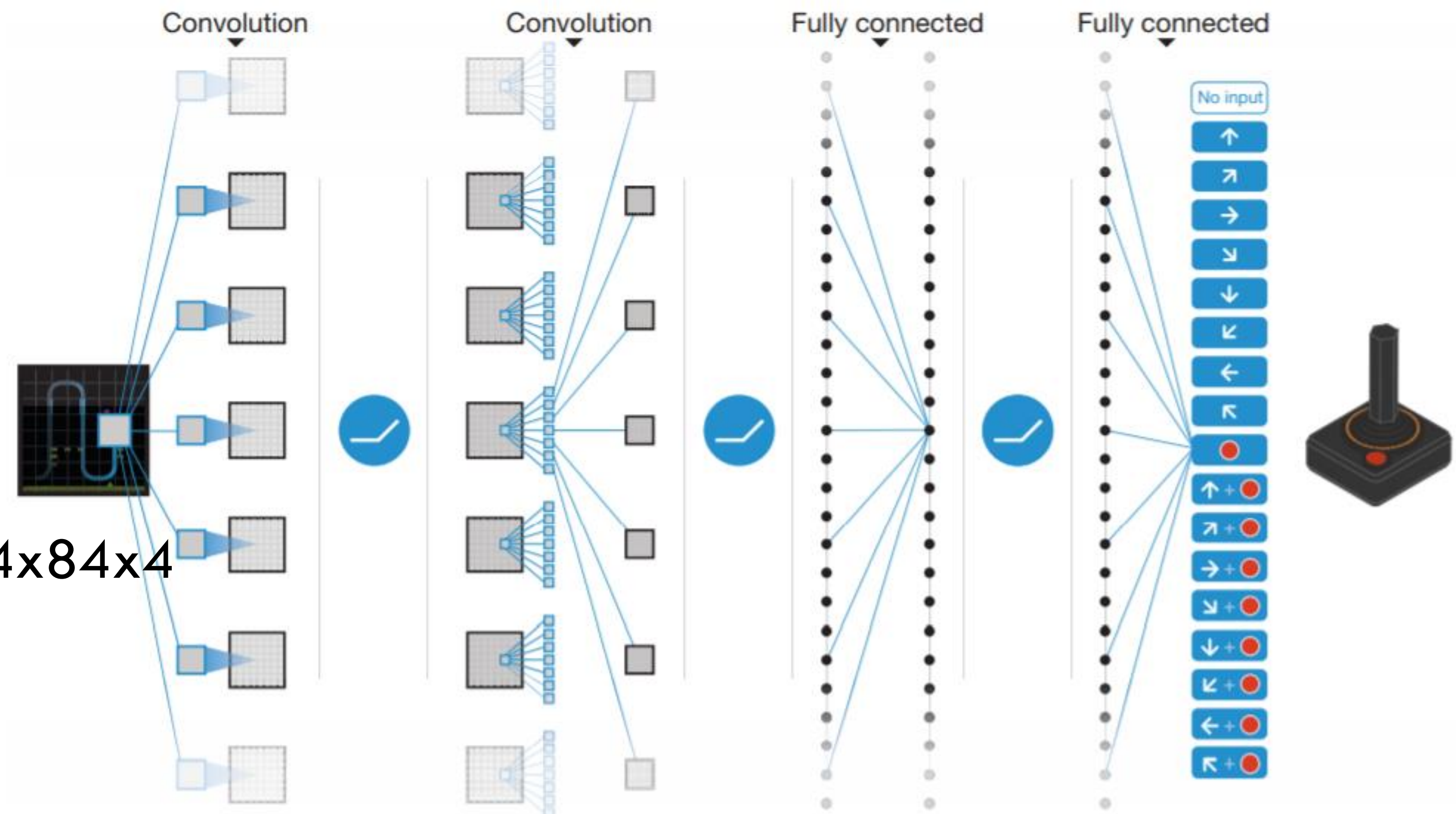
  End For

End For

Deep Q neural networks

- Basic structure is CNN
- To deal with temporal info, use **four consecutive images** as one set of input
- Output is action values for present input image set
- Number of output nodes = number of actions

Deep Q neural networks



84x84x4

What is not covered

- Policy gradients
- A3C
- Model-based learning