Reinforcement Learning

A presentation by Yara Mohamadi

Contents

1 What is Reinforcement Learning?

2 Markov Decision Process

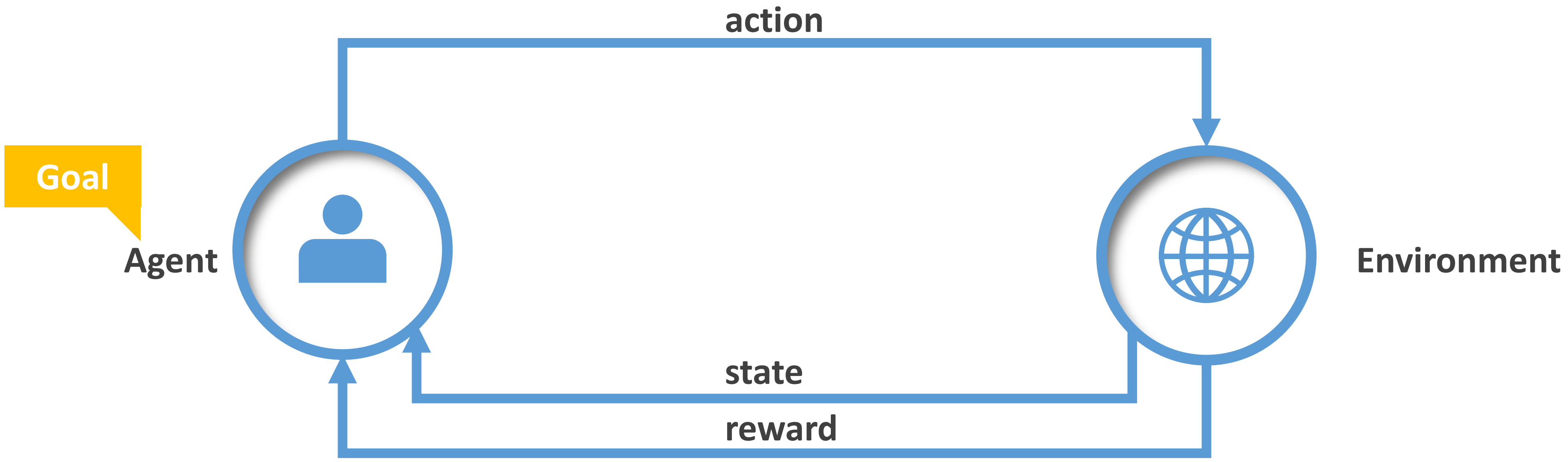
3 Q-Learning

4 Policy Gradients

5 Challenges & Future of RL

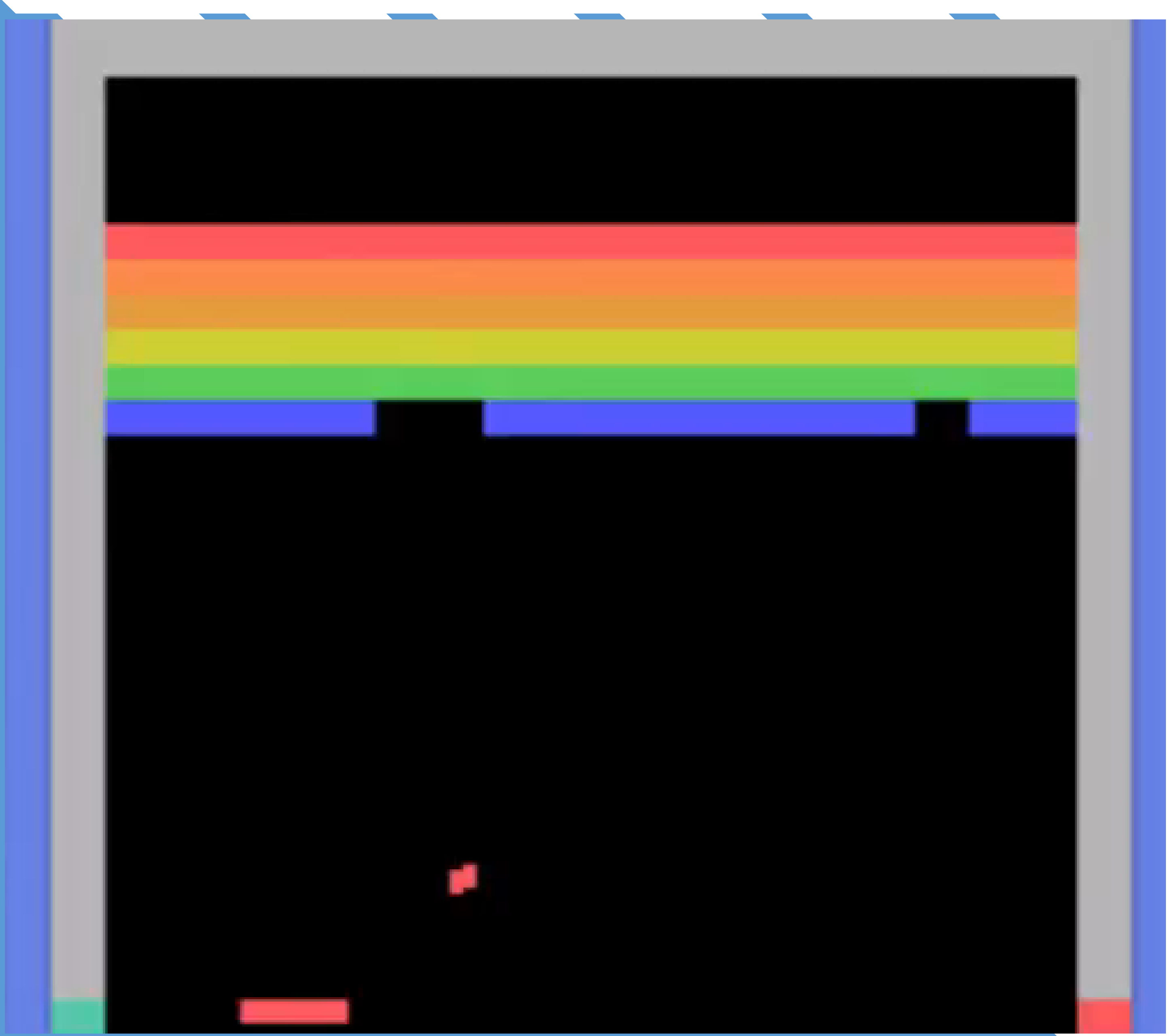
6 Question Answering

Reinforcement learning



Cart Pole

- **Objective:** balance a pole on top of a movable cart
- **State:** angle, angular speed, position, horizontal velocity
- **Action:** horizontal force applied on the cart
- **Reward:** 1 at each time step if the pole is upright



ATARI games

- **Objective**: complete the game with the highest score
- **State**: raw pixel inputs of the game state
- **Action**: game controls

- **Reward** score increase/decrease at each time step

Contents

1 What is Reinforcement Learning?

2 Markov Decision Process

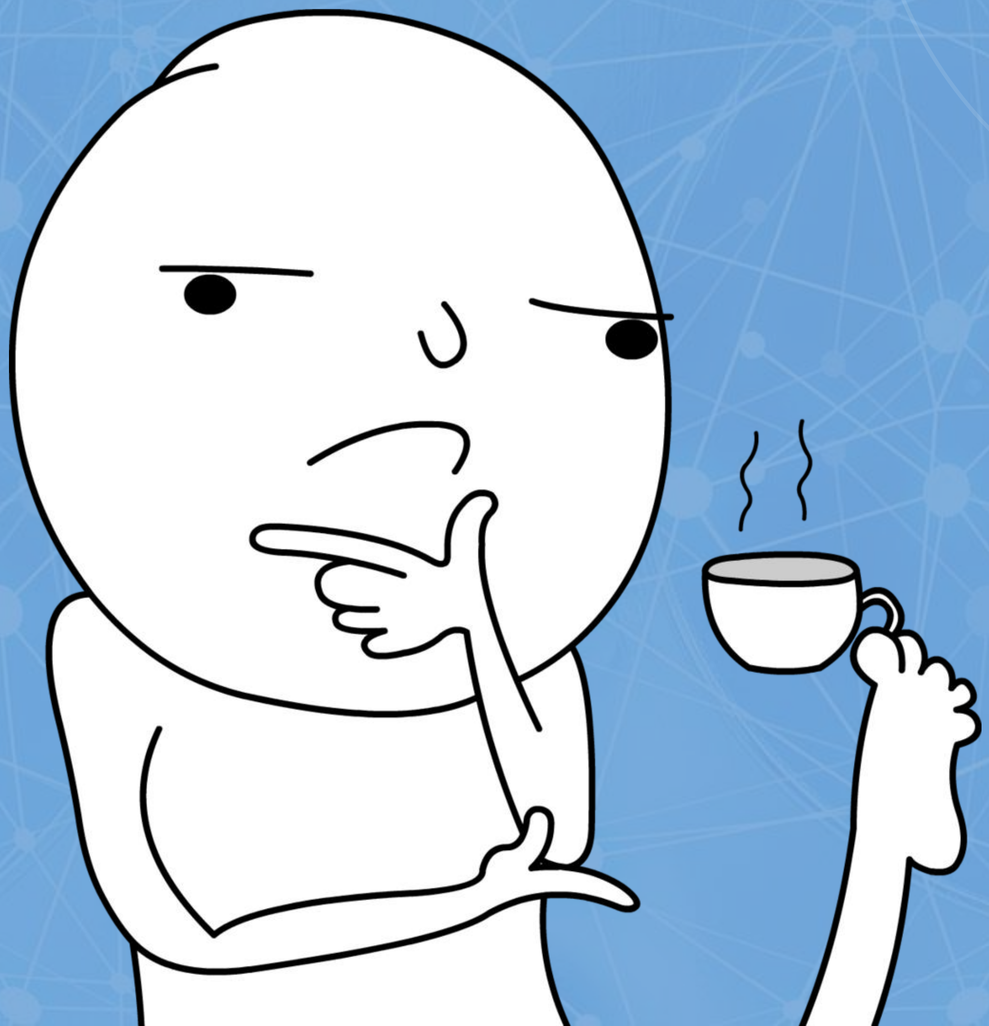
3 Q-Learning

4 Policy Gradients

5 Challenges & Future of RL

6 Question Answering

But how do you formalize reinforcement learning?



Markov Decision Process (MDP)

- **Defined by**: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

- **S** : set of possible states
- **A** : set of possible actions
- **R** : distribution of reward given (state, action) pair
- **P** : transition probability i.e. distribution over next state given (state, action) pair
- **Gamma** : discount factor

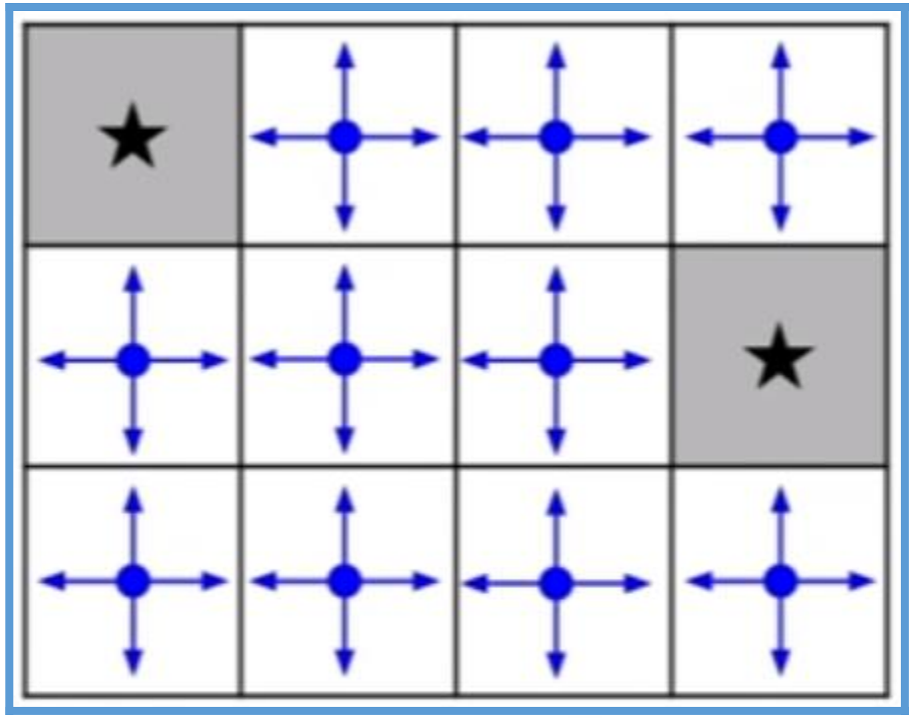
MDP Algorithm

- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

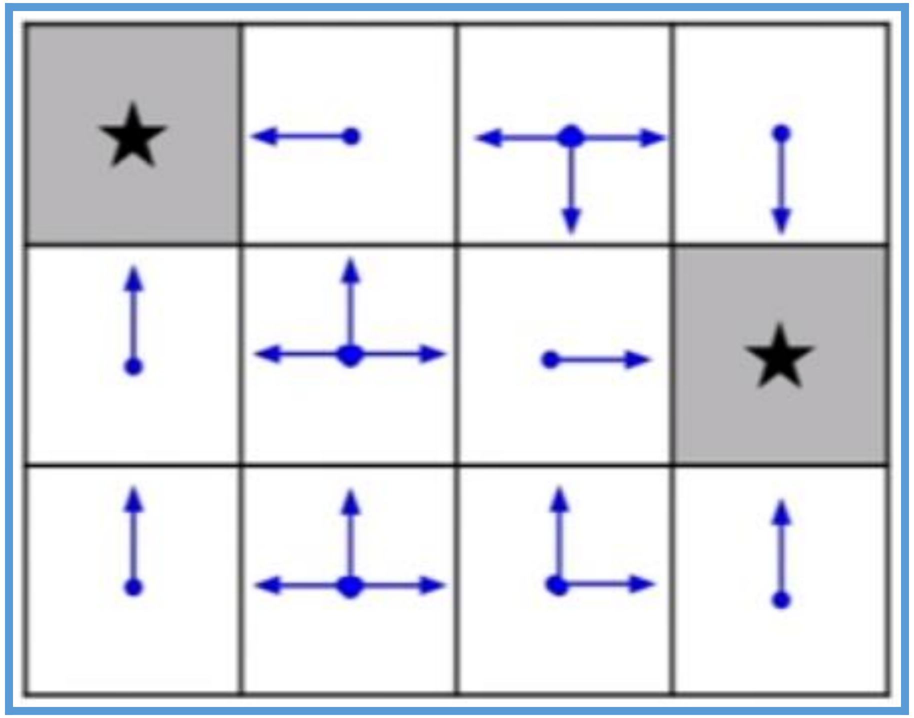
Policy definition (**π**)

- produces sample trajectories : (s0, a0, r0, s1, a1, r1, …)
- **Objective**: find π^* that maximizes $\sum_{t>0} \gamma^t r_t$

Grid World



Random policy



Optimal policy

Handling Randomness

Maximize the expected sum of rewards.

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$$

Contents

1 What is Reinforcement Learning?

2 Markov Decision Process

3 Q-Learning

4 Policy Gradients

5 Challenges & Future of RL

6 Summary

2 important values

Value function

- How good is a state?
- Expected cumulative reward from following the policy from s

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

Q-value function

- How good is a state-action pair?
- Expected cumulative reward from taking an action and then following the policy

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

$$Q^{*}(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Q-Learning

- Q-Value declaration:

  Q(state, action) = R(state, action) + Gamma * MAX[Q(next state, all actions)]

- **Intuition:** If the optimal state-action values for the next time-step (s', a') are known, then the optimal strategy is to take the action that maximizes the expected value of:

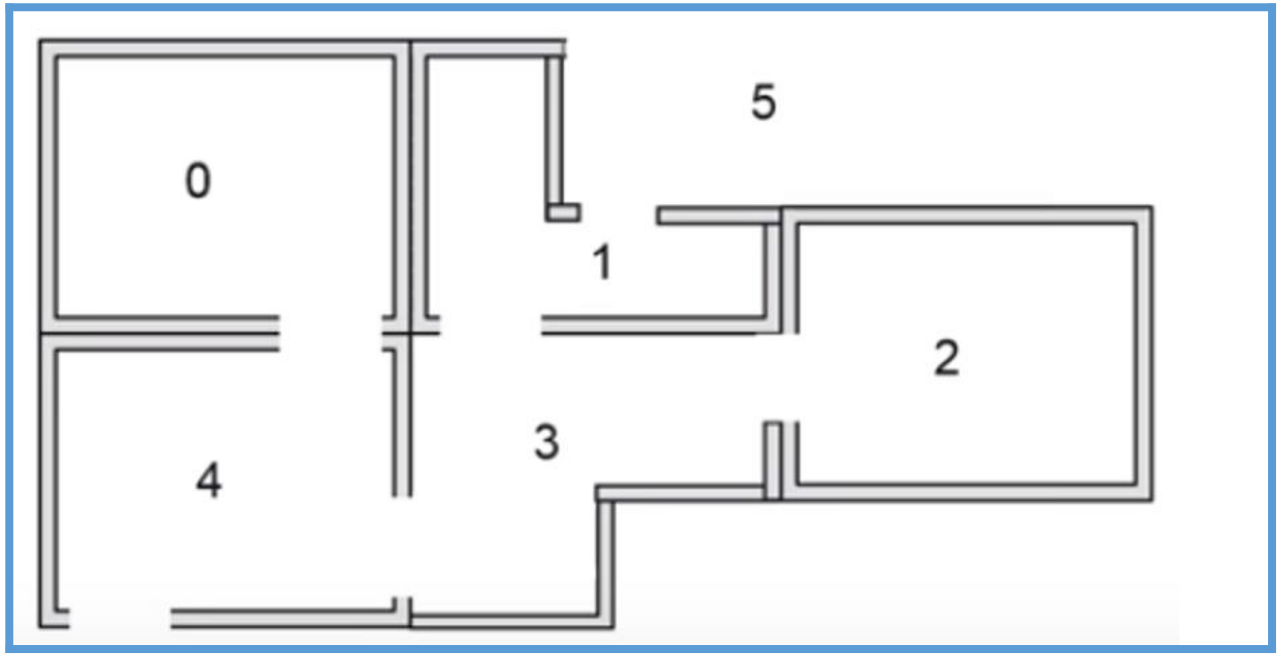
  r + Gamma (Q*(s', a'))

- Optimal policy corresponds to taking the best action in any state specified by Q*
- Value iteration algorithm:

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

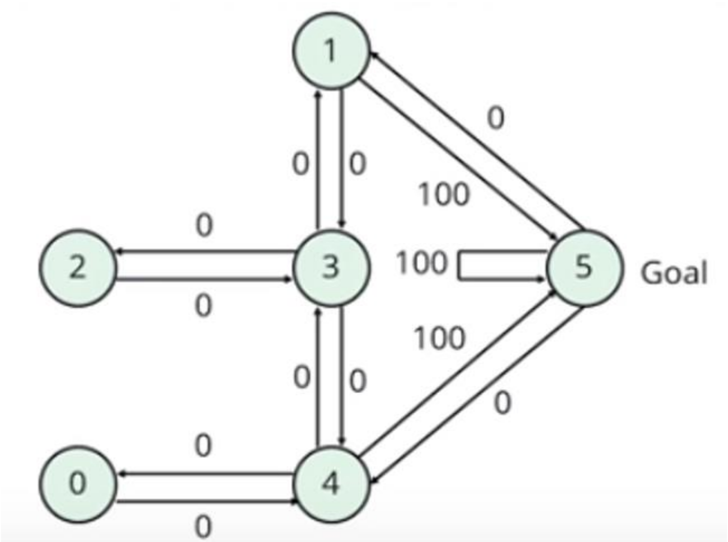
- Q -> Q* as i -> infinity

Example



- **Objective**: reach outside the building
- **State**: rooms
- **Action**: moving through doors
- **Reward:** a value for each door

Example



First episode:

Q(1,5) = R(1,5) + 0.8*max[Q(5,5), Q(5,1), Q(5,4)] = 100 + 0.8*[0, 0, 0] = 100

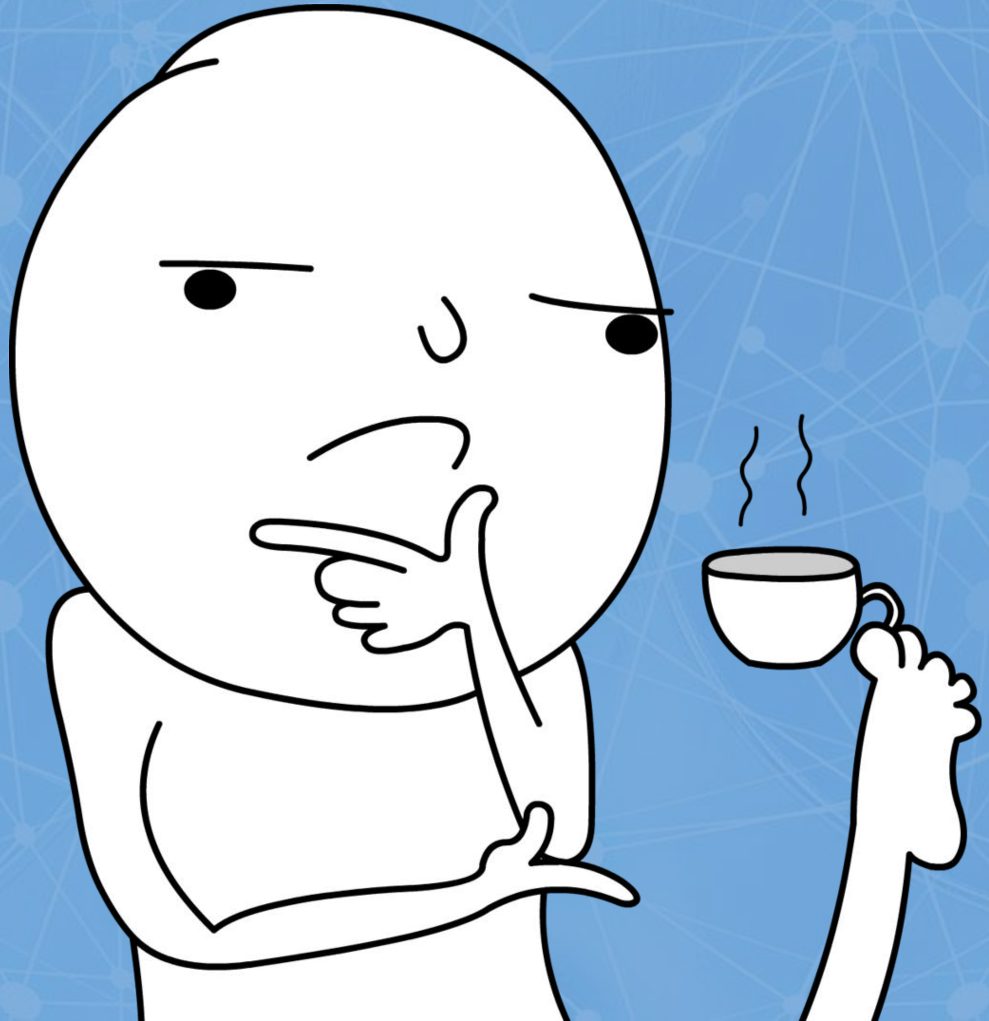
Second episode:

Q(3,1) = R(3,1) + 0.8*max[Q(1,3), Q(1,5)] = 0 + 0.8*[0,100] = 80

```
Trained Q matrix with gamma set to 0.2:
[[  0.    0.    0.    0.   20.   20.8]
 [  0.    0.    0.    4.    0.  100. ]
 [  0.    0.    0.    4.    0.    0. ]
 [  0.   20.    0.8   0.   20.    0. ]
 [  0.   20.    0.8   0.    0.  100. ]
 [  0.   20.    0.    0.   20.  100. ]]
Selected path:
[2, 3, 1, 5]

Trained Q matrix with gamma set to 0.8:
[[  0.    0.    0.    0.   80.   80.2]
 [  0.    0.    0.   64.    0.  100. ]
 [  0.    0.    0.   64.    0.    0. ]
 [  0.   80.   51.2   0.   80.    0. ]
 [  0.   80.   51.2   0.    0.  100. ]
 [  0.   80.    0.    0.   80.  100. ]]
Selected path:
[2, 3, 4, 5]
```

Hold up! There is a problem with this

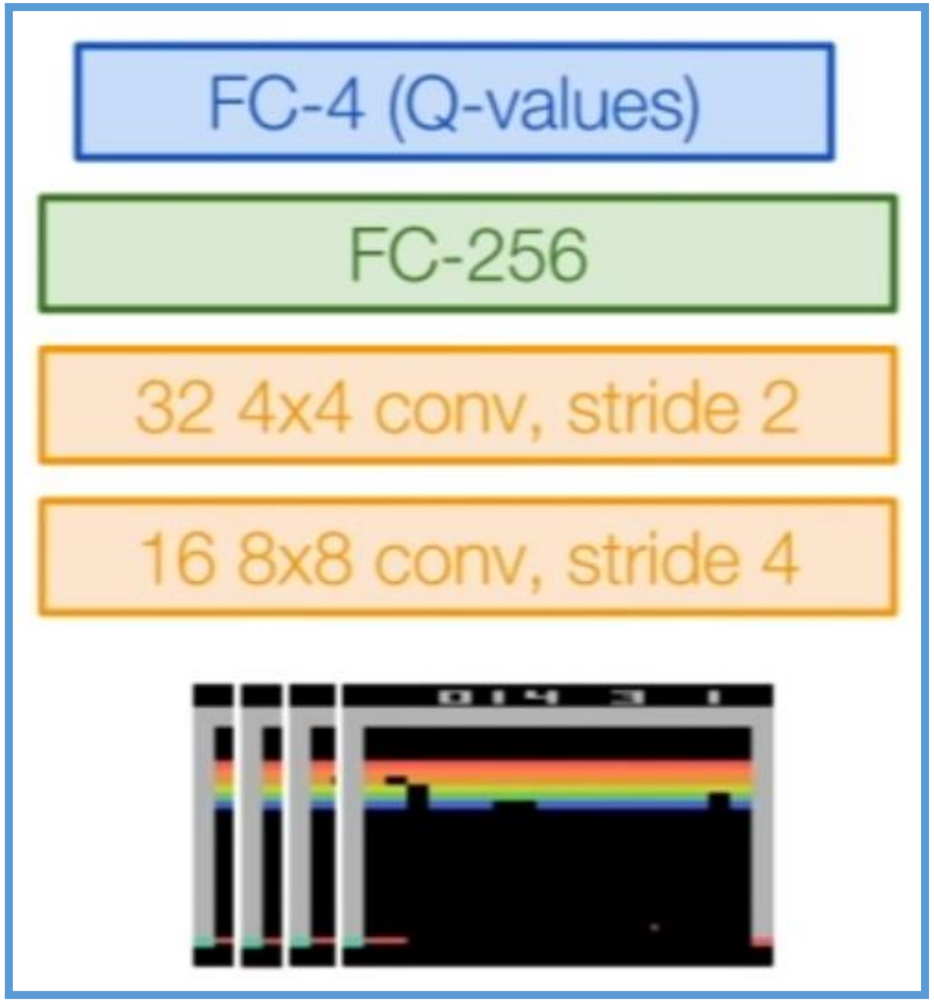


Deep Q-Learning

- Approximate Q:

$$Q(s, a; \theta) \approx Q^*(s, a)$$

- Loss function: minimize the error of our estimated Q
- ATARI example:

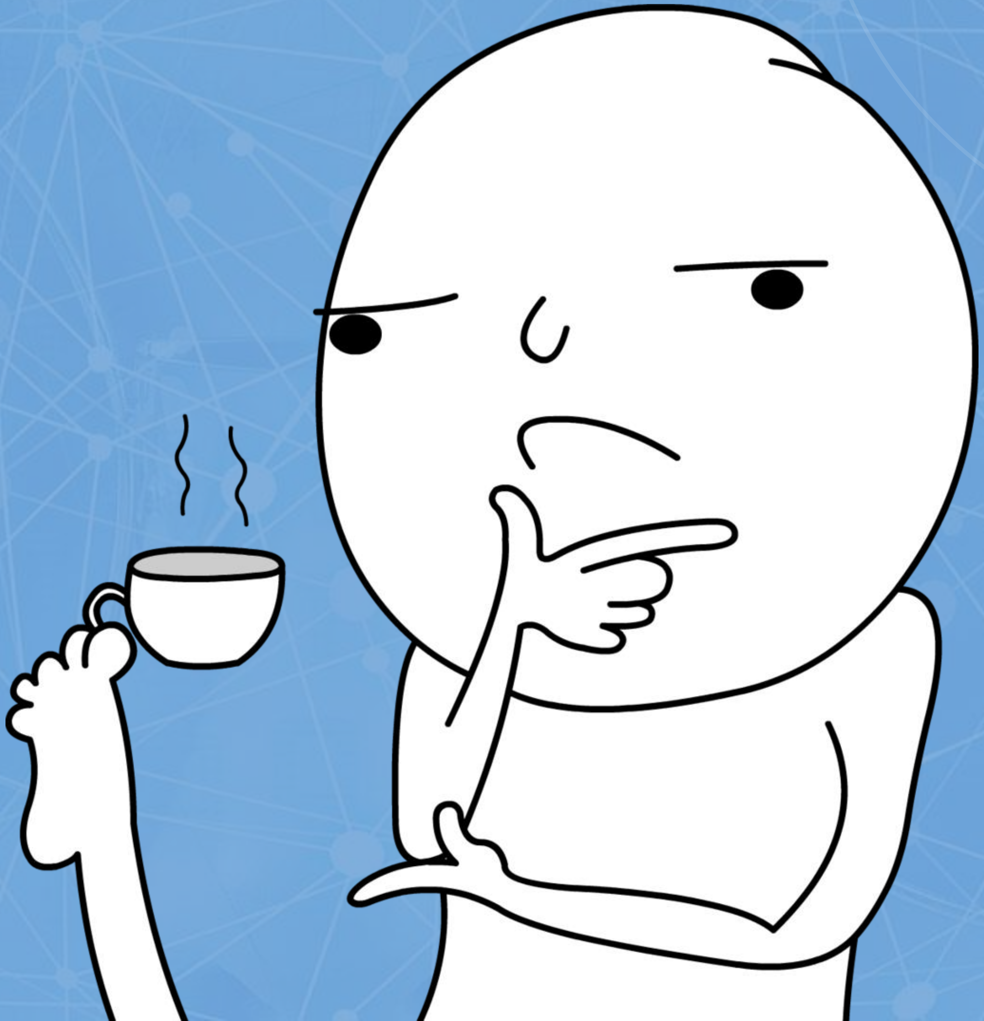


Learning from batches of consecutive samples is problematic

- Samples are correlated
- Current Q-network parameters determine next training samples

Use experience replay

- Update a memory of transitions (st, at, rt, st+1) as you play
- Train on random batches of replay memory



Deep Q-Learning algorithm

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Contents

1 What is Reinforcement Learning?

2 Markov Decision Process

3 Q-Learning

4 Policy Gradients

5 Challenges & Future of RL

6 Question Answering

Policy Gradients

- **Policies:** $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$
- **Policy Value**: $J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$
- **Goal**: find optimal policy $\theta^* = \arg\max_\theta J(\theta)$

Reinforce algorithm

Value of a policy:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

$$\tau = (s_0, a_0, r_0, s_1, \ldots)$$

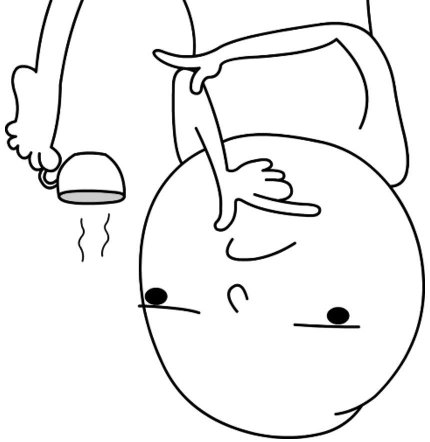
Differentiate it:

$$\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

Fix dependency:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Intuition



- If r(tau) is high, push up the probabilities of the actions seen
- If r(tau) is low, push down the probabilities of the actions seen

- simple, but it works!
- Unbiased estimator
- High variance due to hard credit assignment
- We need lots of samples

Can we help our estimator?

3 ideas

1st idea:

- Use cumulative future rewards

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

2nd idea:

- Ignore delayed effects

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

3rd idea:

- Add a baseline function
- Is the reward better or worse than what you expect it to be?

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- We don't know Q and V, but we can Q-Learn it!

Actor-Critic Algorithm

- Train an actor (the policy) and a critic (the Q-function)

- actor chooses action, critic gives feedback (how good it thinks an action is good)
- Critic only learns (state, action) pairs generated by the policy

- **Remark**: advantage function $A^{\pi}(s, a)$

Actor-Critic Algorithm

Initialize policy parameters θ, critic parameters ϕ

For iteration=1, 2 … **do**

Sample m trajectories under the current policy

$\Delta\theta \leftarrow 0$

For i=1, …, m **do**

For t=1, … , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

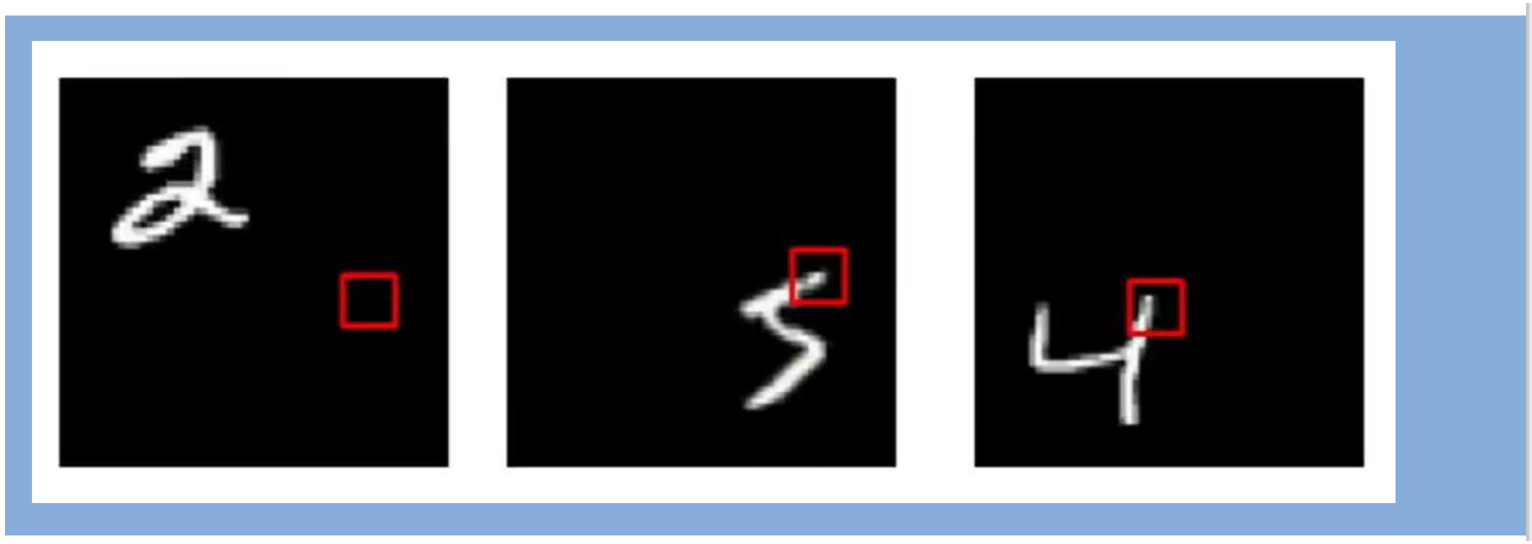
$$\theta \leftarrow \alpha\Delta\theta$$

$$\phi \leftarrow \beta\Delta\phi$$

End for

Example: Recurrent Attention Model (RAM)

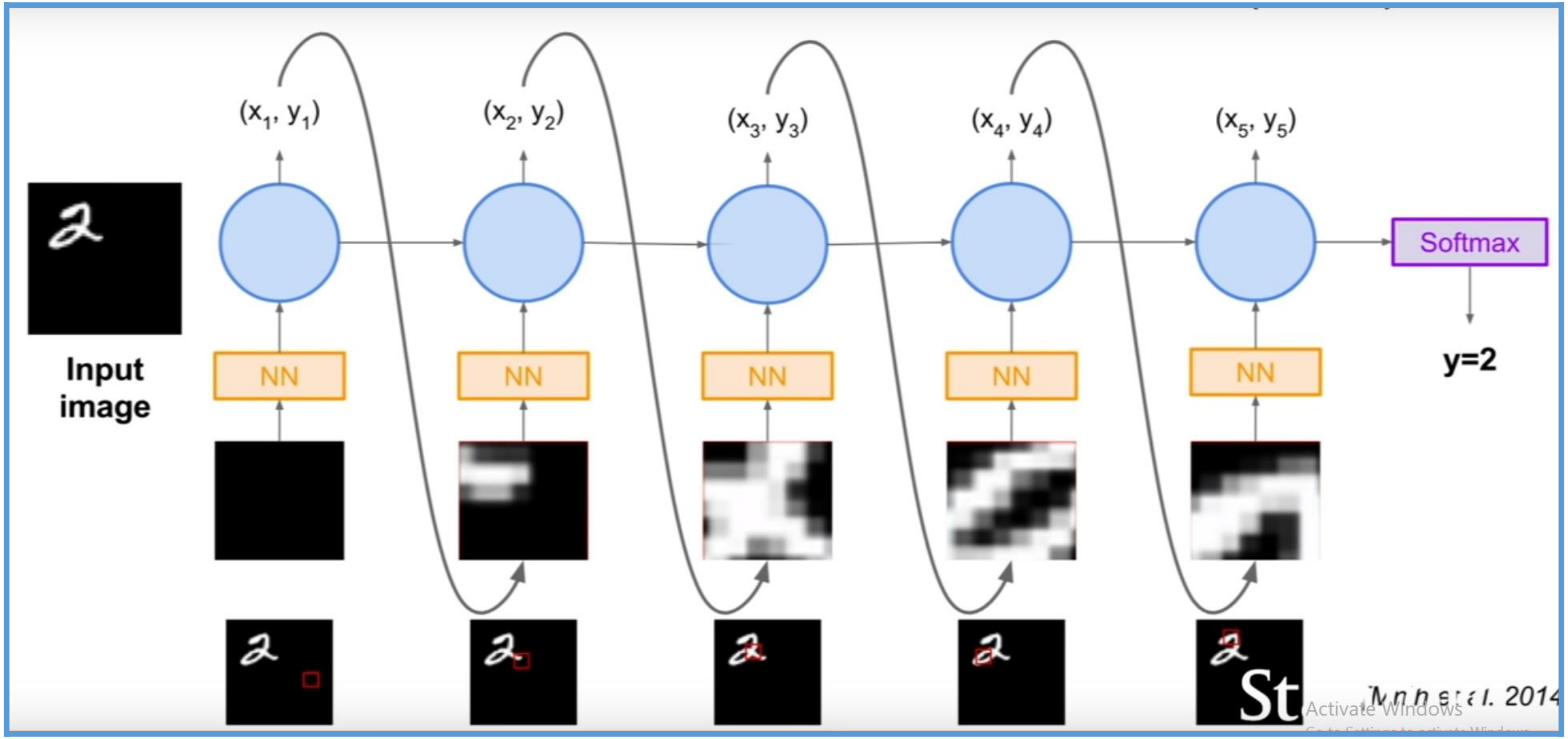
- **Objective**: Image Classification
- **State**: Glimpses seen so far
- **Action**: (x,y) coordinates of where to look next in image
- **Reward**: 1 at the final time-step if image correctly classified, 0 otherwise



Why reinforce?

- Inspiration from human perception and eye movements
- Saves computational resources
- Ignores irrelevant parts of image

Example: Recurrent Attention Model (RAM)



Contents

1 What is Reinforcement Learning?

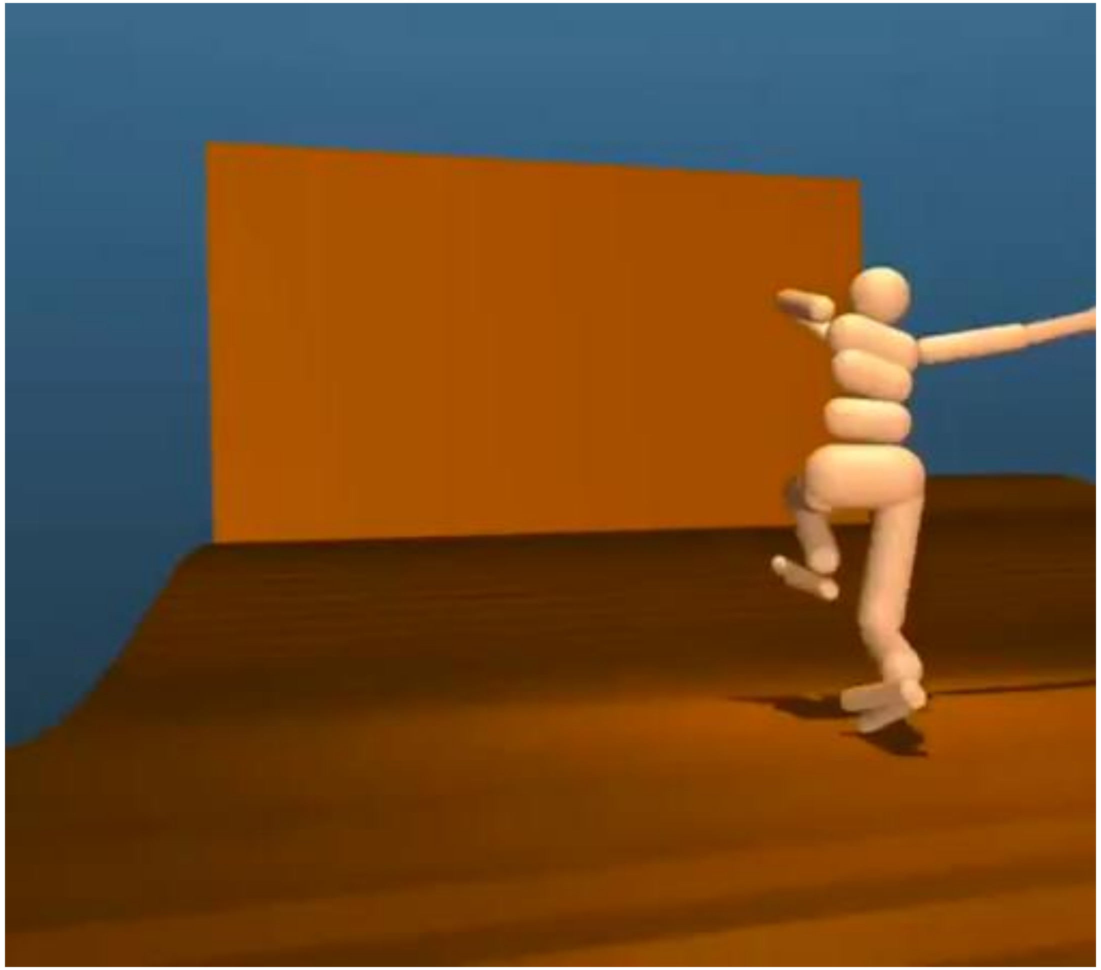
2 Markov Decision Process

3 Q-Learning

4 Policy Gradients

5 Challenges & Future of RL

6 Question Answering



“ the only way you can learn something is when you almost know it already. ”

“ afterall, maybe RL hasn’t advanced so much, and the reason might be because the emphasis is always performance on a single task, at whatever the training cost. ”

Thank you for your time

Any Questions?