

Monte Carlo Tree Search



Exploration reduces uncertainty

Model-free setting

- Unknown environment dynamics $p(s', r|s, a)$

Model-based setting

- Known model of the world
 - Distribution model: $p(s', r|s, a)$
 - Sample model: $(s', r) \sim G(s, a)$

Do we need exploration for model-based learning?



Exploration reduces uncertainty

Model-free setting

- Unknown environment dynamics $p(s', r|s, a)$

Model-based setting

- Known model of the world
 - Distribution model: $p(s', r|s, a)$
 - Sample model: $(s', r) \sim G(s, a)$

Do we need exploration for model-based learning?



Exploration reduces uncertainty

Model-free setting

- Unknown environment dynamics $p(s', r|s, a)$

Model-based setting

- Known model of the world
 - Distribution model: $p(s', r|s, a)$
 - Sample model: $(s', r) \sim G(s, a)$

Do we need exploration for model-based learning?

Still don't know the global values of any action!



We don't know global value estimates

Given the model of the world, what is the best action?

1. Immediate rewards are **insufficient**
2. **Time constraints** on recovering of global estimates



We don't know global value estimates

Given the model of the world, what is the best action?

1. Immediate rewards are **insufficient**
2. **Time constraints** on recovering of global estimates

Planning

Transform

- known immediate rewards & dynamics
into
- value / action-value estimates / explicit policy



Types of planning

Model-free – learn the policy by trial and error

Model-based – plan to obtain the policy

- Background planning (DP)
 - learning from simulated experience
 - improves estimates in arbitrary states
- Decision-time planning (heuristic search, MCTS)
 - focuses on current state
 - plans action selection for current state only
 - commit action only after planning is finished



Types of planning

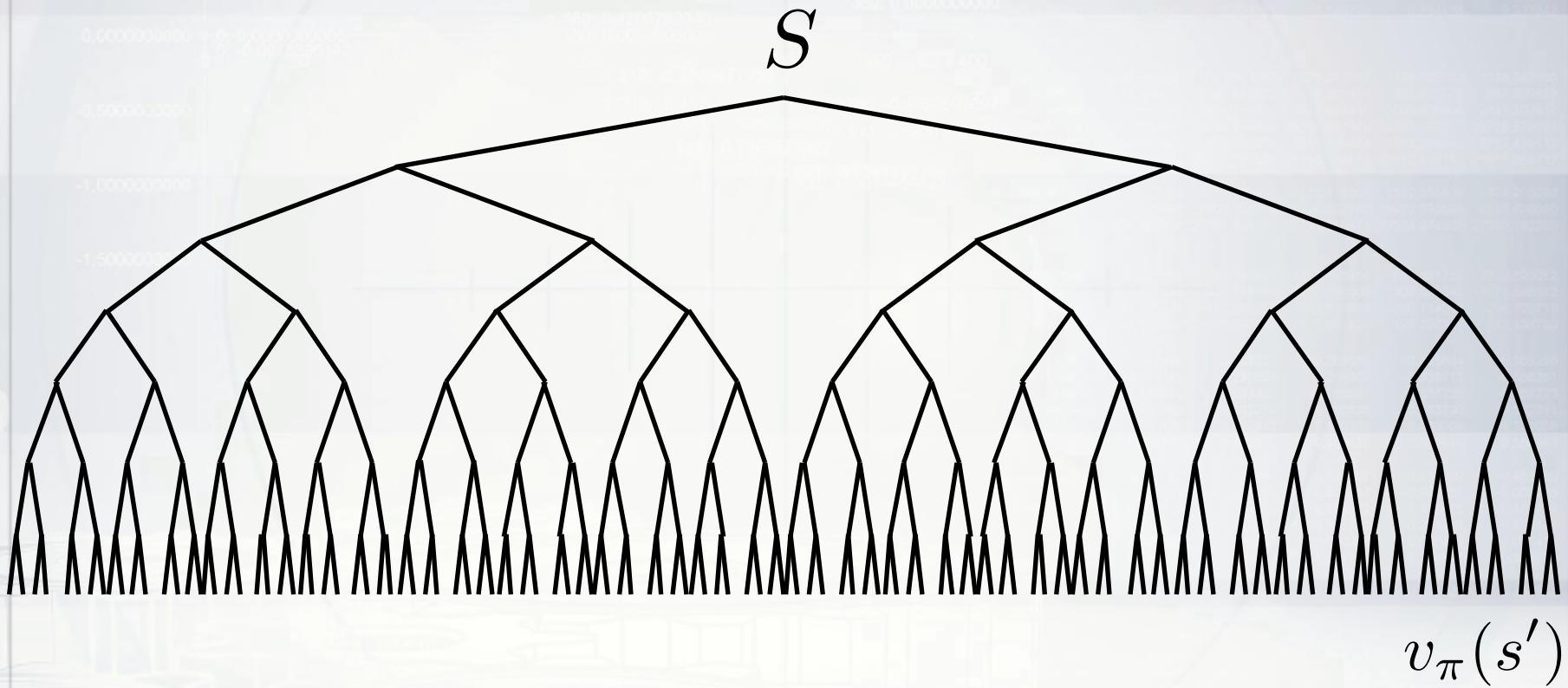
Model-free – learn the policy by trial and error

Model-based – plan to obtain the policy

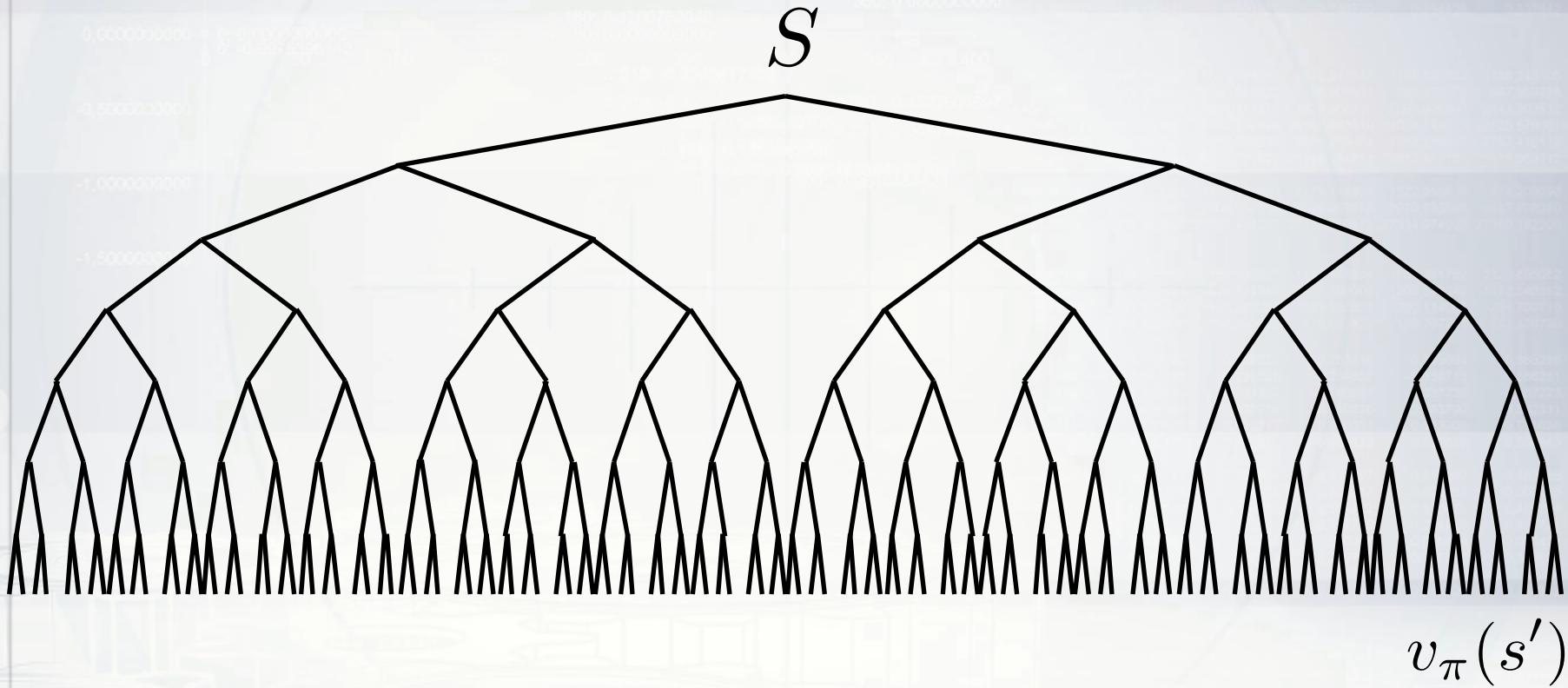
- Background planning (DP)
 - learning from simulated experience
 - improves estimates in arbitrary states
- Decision-time planning (heuristic search, MCTS)
 - focuses on current state
 - plans action selection for current state only
 - commit action only after planning is finished



Guided planning – heuristic search



Guided planning – heuristic search



Guided planning – heuristic search

Heuristic search – an umbrella term for many algorithms

- lookahead planning guided by heuristic function
 - select only “valuable” nodes to expand

Heuristic search for RL

- + resources are focused on valuable paths
- + nearest states contribute the most to the return
- bad results when the model is unreliable



Guided planning – heuristic search

Heuristic search – an umbrella term for many algorithms

- lookahead planning guided by heuristic function
 - select only “valuable” nodes to expand

Heuristic search for RL

- + resources are focused on valuable paths
- + nearest states contribute the most to the return
- bad results when the model is unreliable



Obtain value estimates with rollouts



Monte Carlo Tree Search in a glance

- Rollout algorithm which



Monte Carlo Tree Search – basic parts

Planning



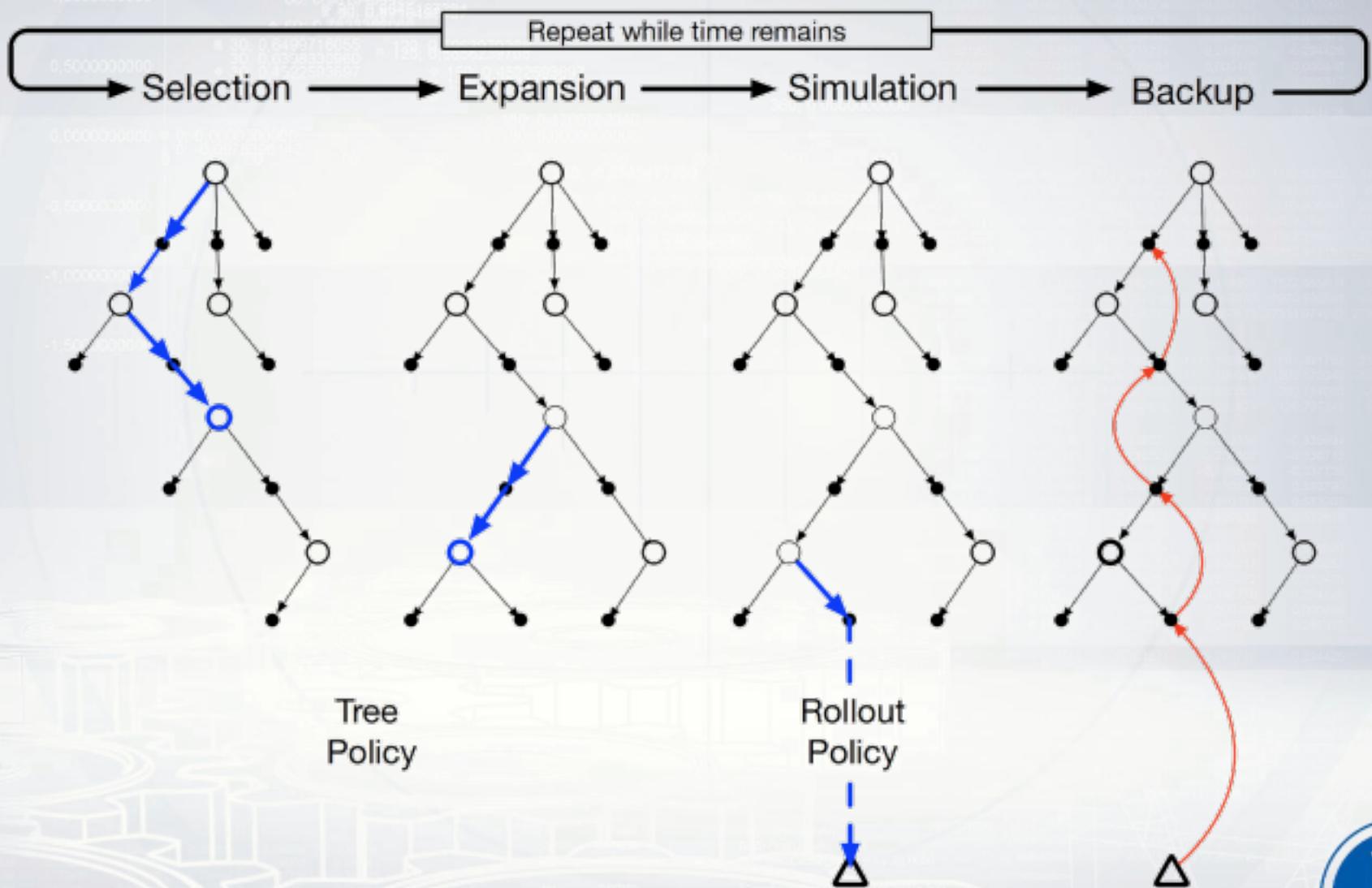
Planning

Planning

Авторство



Scheme of Monte Carlo Tree Search



Авторство? Своя картинка или перерисовать?



Recap: Approximate Q-learning

$$\begin{aligned}Q(s, a_0) \\Q(s, a_1) \\Q(s, a_2)\end{aligned}$$

$$\begin{array}{c}\text{model} \\ \Theta = \text{params}\end{array}$$

$$\begin{array}{c}\uparrow \\ \text{image}\end{array}$$

Q-values:

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \max_a \hat{Q}(s_{t+1}, a')$$

Objective:

$$L = (Q(s_t, a_t) - [r + \gamma \cdot \max_{a'} Q(s_{t+1}, a')])^2$$

Gradient step:

$$w_{t+1} = w_t - \alpha \cdot \frac{\delta L}{\delta w}$$



Recap: Approximate Q-learning

$$\begin{aligned}Q(s, a_0) \\Q(s, a_1) \\Q(s, a_2)\end{aligned}$$

$$\begin{array}{c}\text{model} \\ \Theta = \text{params}\end{array}$$

$$\begin{array}{c}\text{image}\end{array}$$

Q-values:

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \max_a \hat{Q}(s_{t+1}, a')$$

Objective:

$$L = (Q(s_t, a_t) - [r + \gamma \cdot \max_{a'} Q(s_{t+1}, a')])^2$$

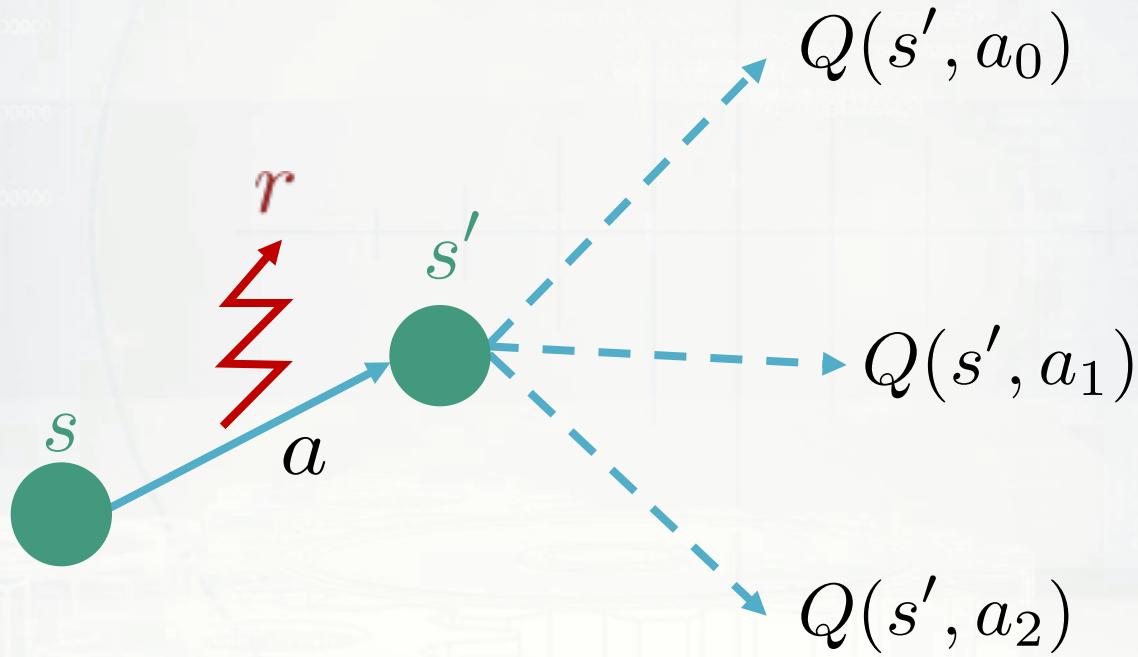
consider const

Gradient step:

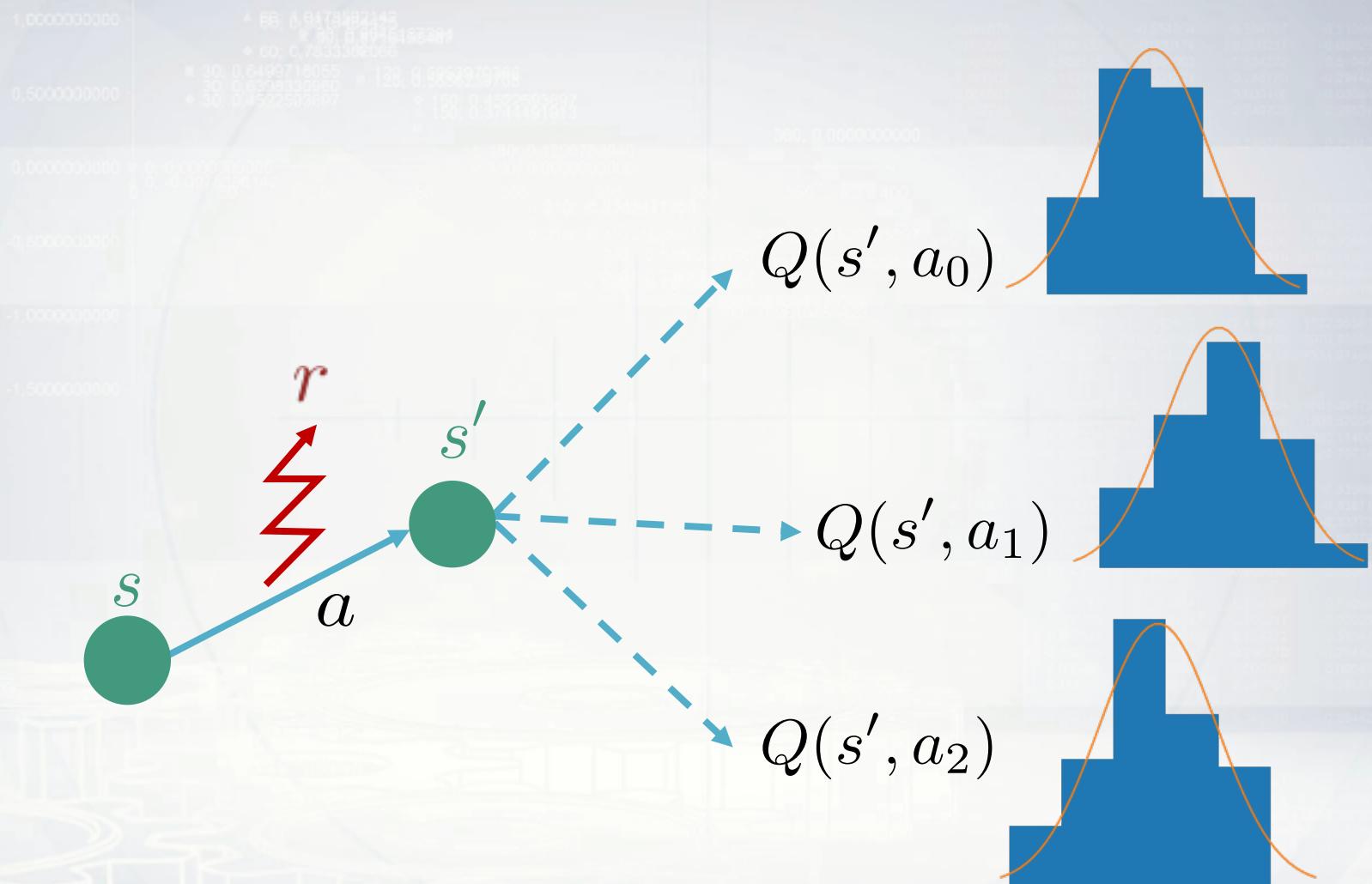
$$w_{t+1} = w_t - \alpha \cdot \frac{\partial L}{\partial w}$$



We have a problem

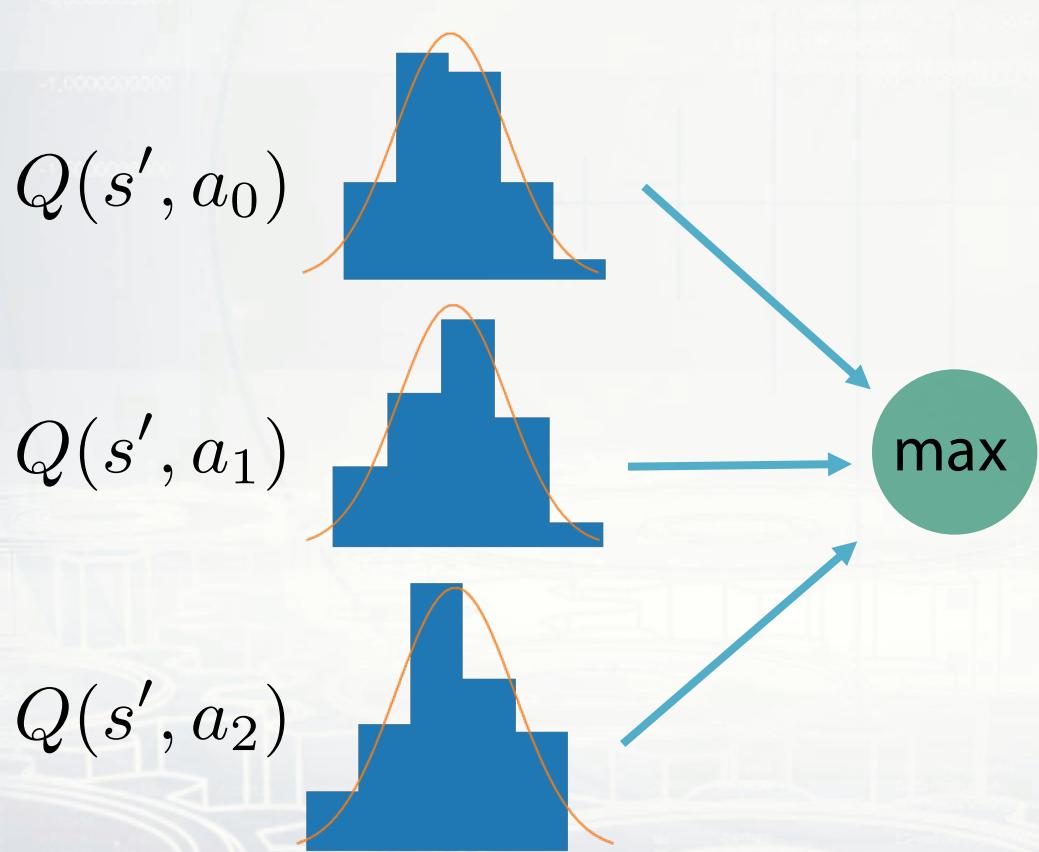


We have a problem



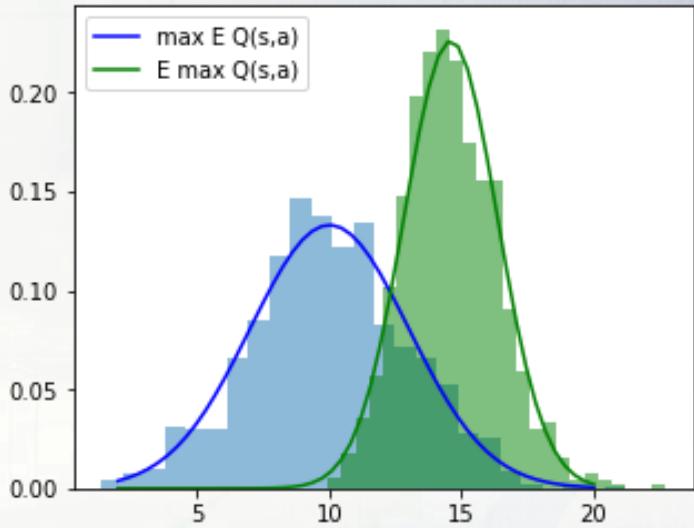
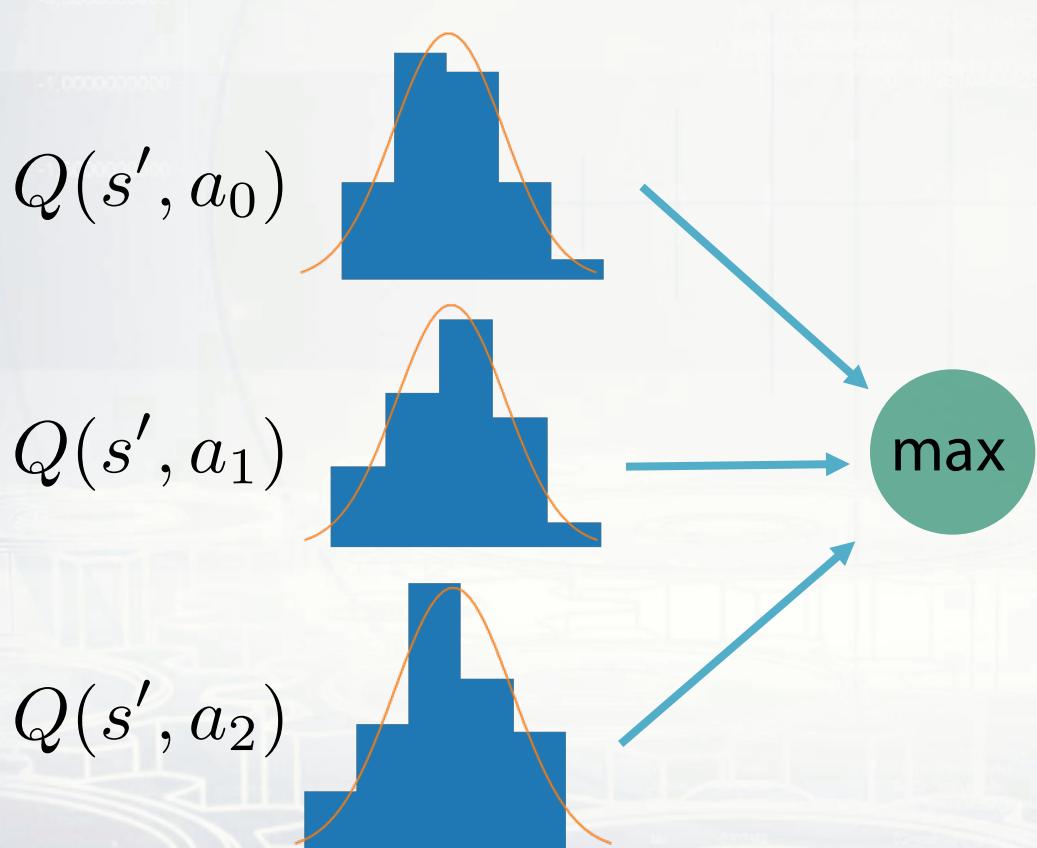
We have a problem

Q-value update: $\hat{Q}(s_t, a_t) = r + \gamma \cdot \max_{a'} \hat{Q}(s_{t+1}, a')$



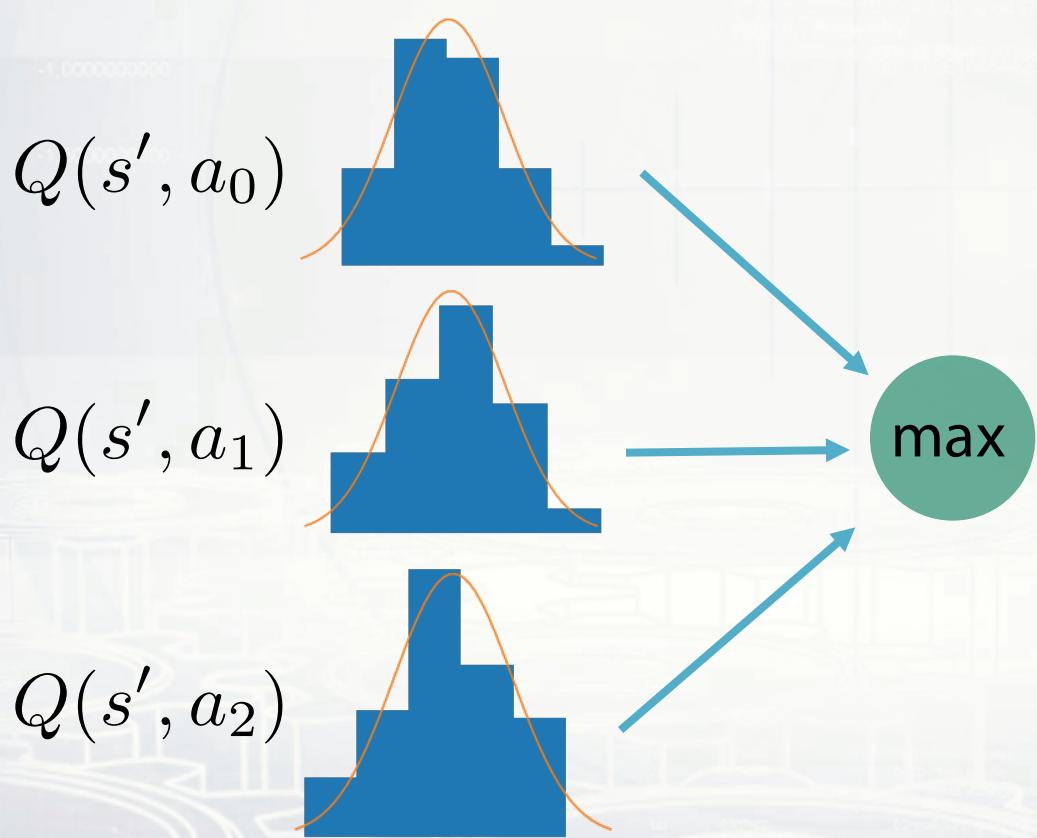
We have a problem

$$E_{s'} \max_{a'} \hat{Q}(s_{t+1}, a') \geq \max_{a'} E_{s'} \hat{Q}(s_{t+1}, a')$$



We have a problem

Q-value update: $\hat{Q}(s_t, a_t) = r + \gamma \cdot \max_{a'} \hat{Q}(s_{t+1}, a')$



What happens to
max $Q(s, a')$?



Double Q-learning

Idea: train two Q -functions separately, Q_1 and Q_2



Double Q-learning

Idea: train two Q -functions separately, Q_1 and Q_2

$$\hat{Q}_1(s_t, a_t) = r + \gamma \cdot \max_{a'} \hat{Q}_2(s_{t+1}, a')$$

$$\hat{Q}_2(s_t, a_t) = r + \gamma \cdot \max_{a'} \hat{Q}_1(s_{t+1}, a')$$



Double Q-learning

Idea: train two Q functions separately, Q_1 and Q_2

$$\hat{Q}_1(s_t, a_t) = r + \gamma \cdot \max_{a'} \hat{Q}_2(s_{t+1}, a')$$

$$\hat{Q}_2(s_t, a_t) = r + \gamma \cdot \max_{a'} \hat{Q}_1(s_{t+1}, a')$$



Double Q-learning todo:fix

Initialize $Q_1(s, a)$ and $Q_2(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily

Initialize $Q_1(\text{terminal-state}, \cdot) = Q_2(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε -greedy in $Q_1 + Q_2$)

 Take action A , observe R, S'

 With 0.5 probability:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left(R + \gamma Q_2(S', \operatorname{argmax}_a Q_1(S', a)) - Q_1(S, A) \right)$$

 else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha \left(R + \gamma Q_1(S', \operatorname{argmax}_a Q_2(S', a)) - Q_2(S, A) \right)$$

$S \leftarrow S'$

 until S is terminal

Figure 6.12: Double Q-learning.

Double Q-learning

Now consider DQN:

- Neural net policy
- Experience replay
- Target networks

Idea 1: train two networks separately.



Double Q-learning

Now consider DQN:

- Neural net policy
- Experience replay
- Target networks

Idea 1: train two networks separately.

Can we do better?



Double Q-learning

Idea 2: use target network instead of Q_2

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \max_{a'} \hat{Q}(s_{t+1}, a')$$



$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \hat{Q}(s_{t+1}, \operatorname{argmax}_{a'} \hat{Q}(s_{t+1}, a'))$$



Double Q-learning

Idea 2: use target network instead of Q_2

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \max_{a'} \hat{Q}(s_{t+1}, a')$$



$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \hat{Q}(s_{t+1}, \operatorname{argmax}_{a'} \hat{Q}(s_{t+1}, a'))$$



$$\hat{Q}(s_t, a_t) = r + \gamma \cdot Q^{old}(s_{t+1}, \operatorname{argmax}_{a'} \hat{Q}(s_{t+1}, a'))$$



We have a problem #2

Let's watch the video

<https://www.youtube.com/watch?v=UXurvvDY93o>

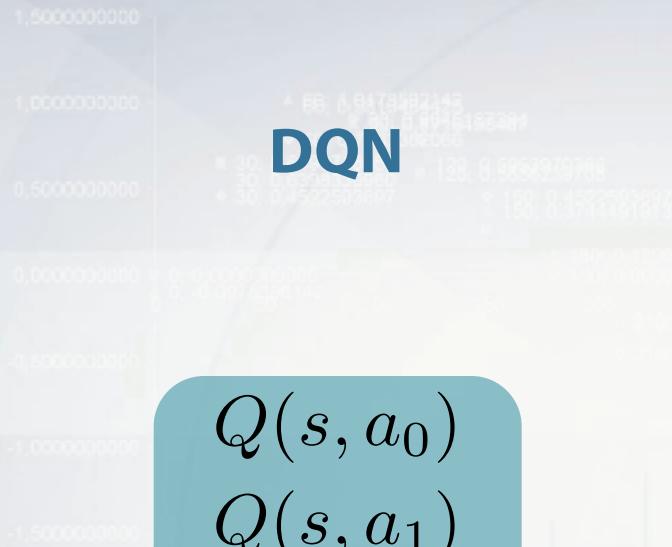


We have a problem #2

Defining $Q = V + A$

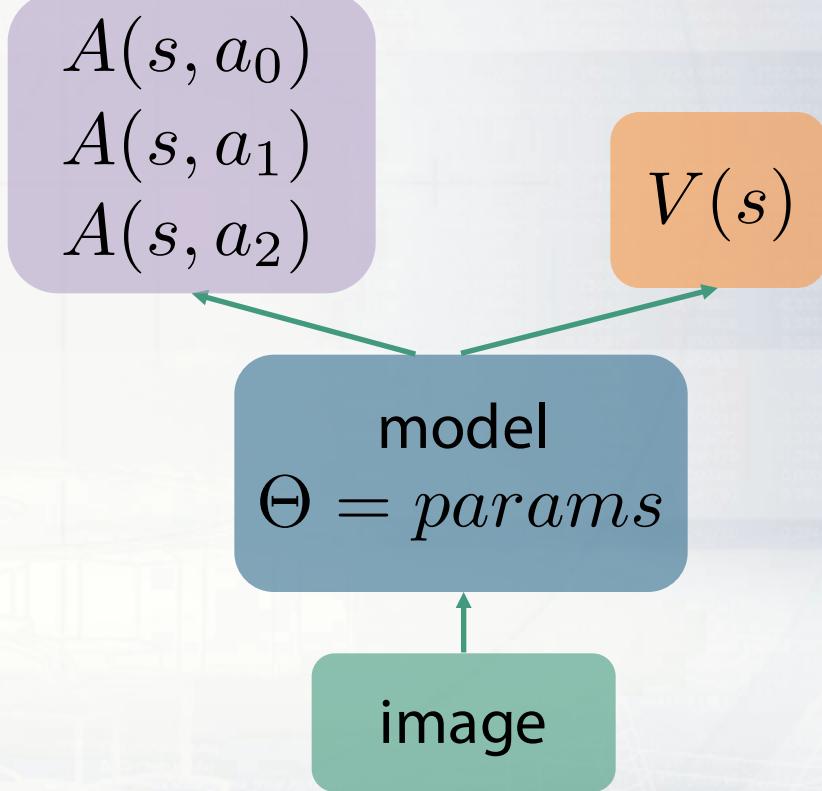


DQN



Dueling DQN

$$Q(s, a_0), Q(s, a_1), Q(s, a_2)$$

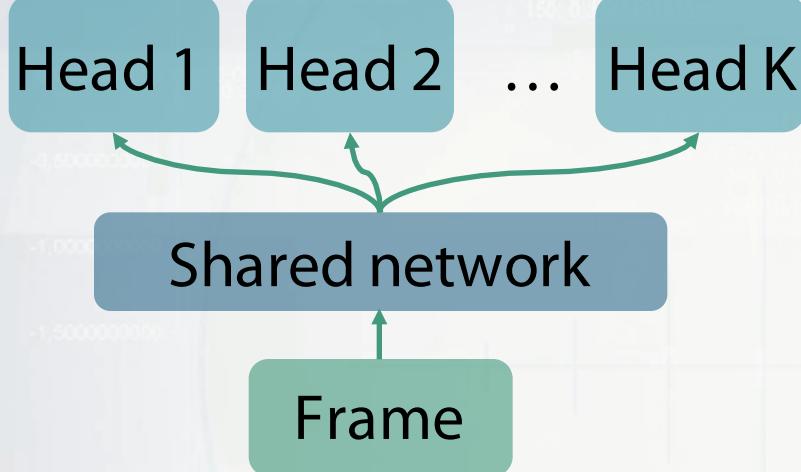


We have a problem #2

Substract mean vs max



Bootstrap DQN

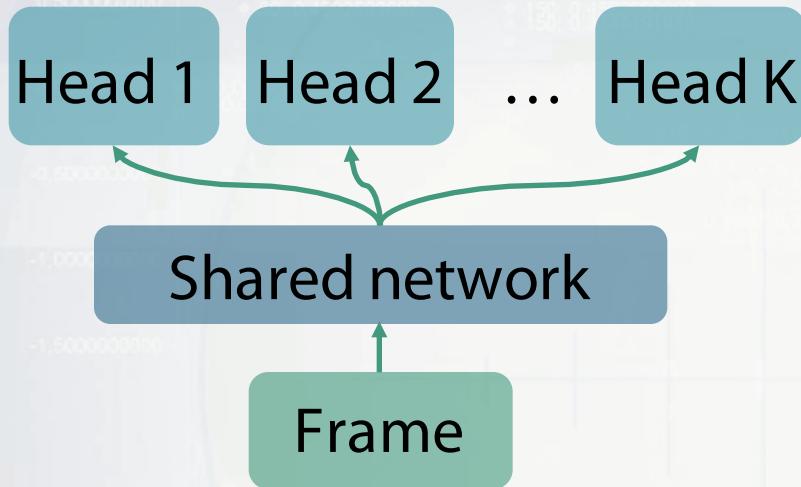


Idea: make exploration great again more clever. Usually we use w-greedy strategy to pick action.

Now we maintain K separate heads (for example, FC-layers) and shared body (for example, convolutional) weights.



Bootstrap DQN



Idea: make exploration great again more clever. Usually we use w-greedy strategy to pick action.

Now we maintain K separate heads (for example, FC-layers) and shared body (for example, convolutional) weights.

Every episode we pick a head randomly and train both this head and shared network on transitions from that episode.

It allow us to make deep exploration.

Often to explore agent should make several non-greedy actions in a row, e-greedy strategy does n't allow to do it.



Table of all things

Results table



Problem:

In most real environments agent observation does not hold all information about system state (e.g. human field of view).



Problem:

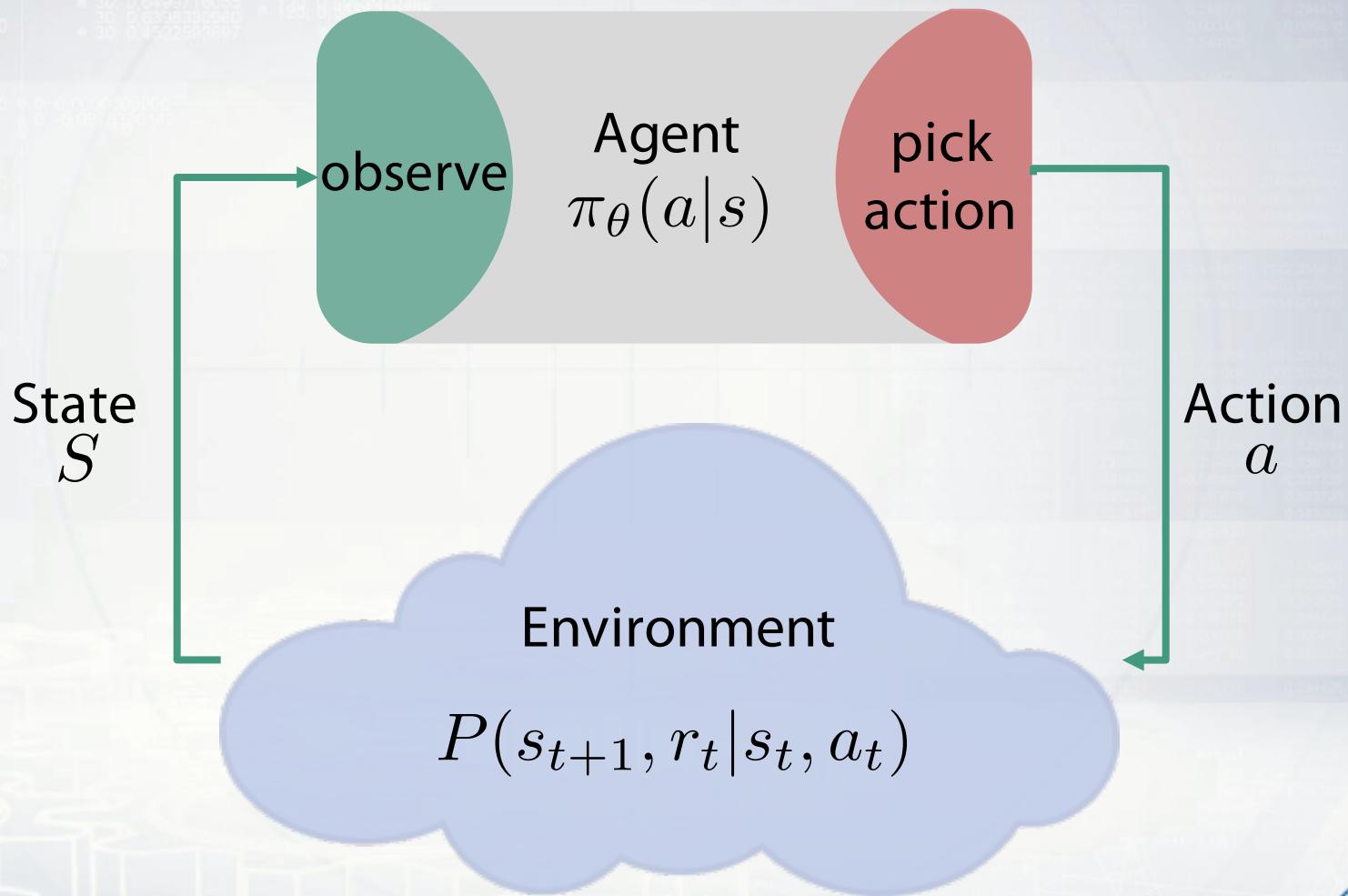
In most real environments agent observation does not hold all information about system state (e.g. human field of view).

Examples:

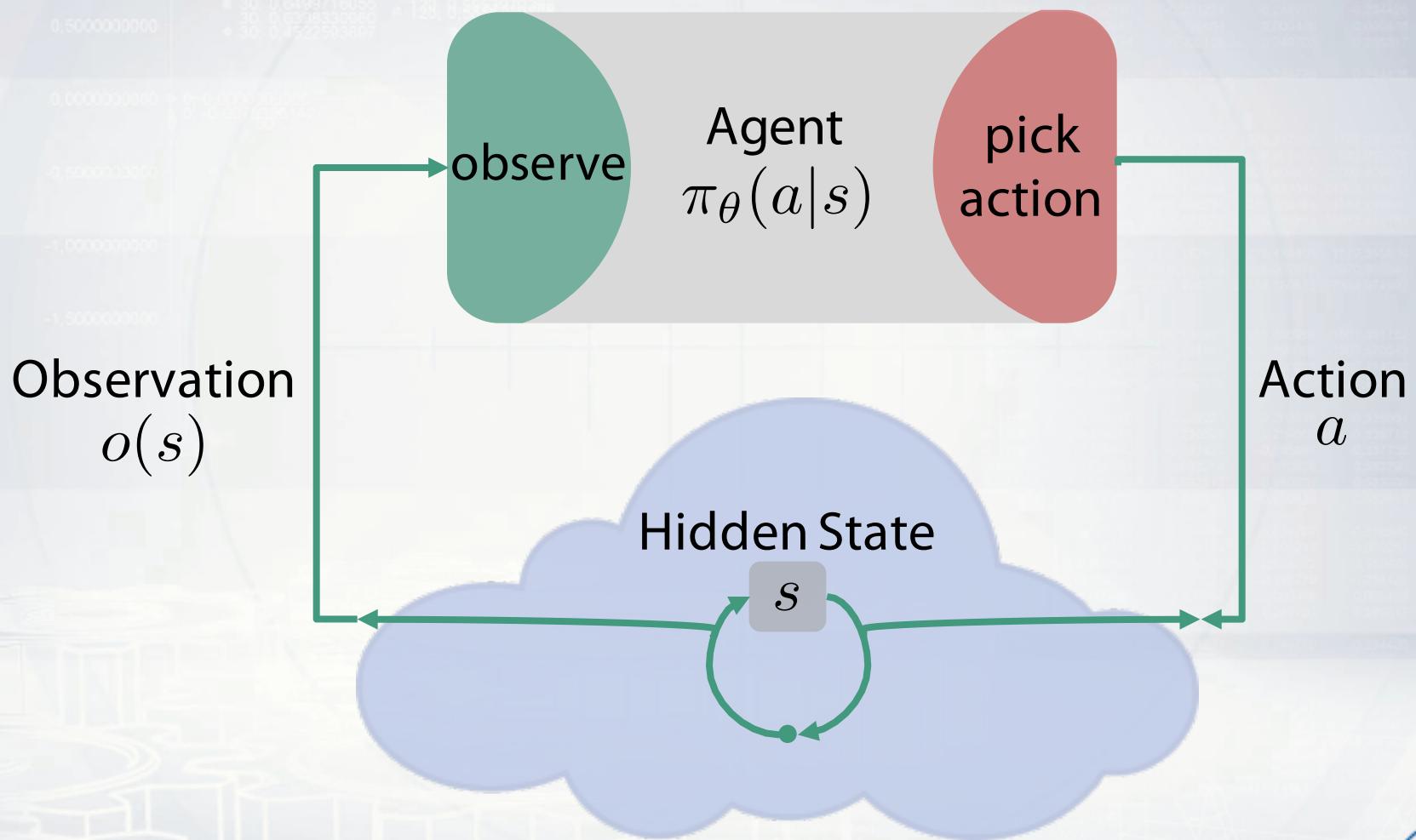
- Robotics: field of view
- Trading bots: market state
- Atari Breakout: ball velocity



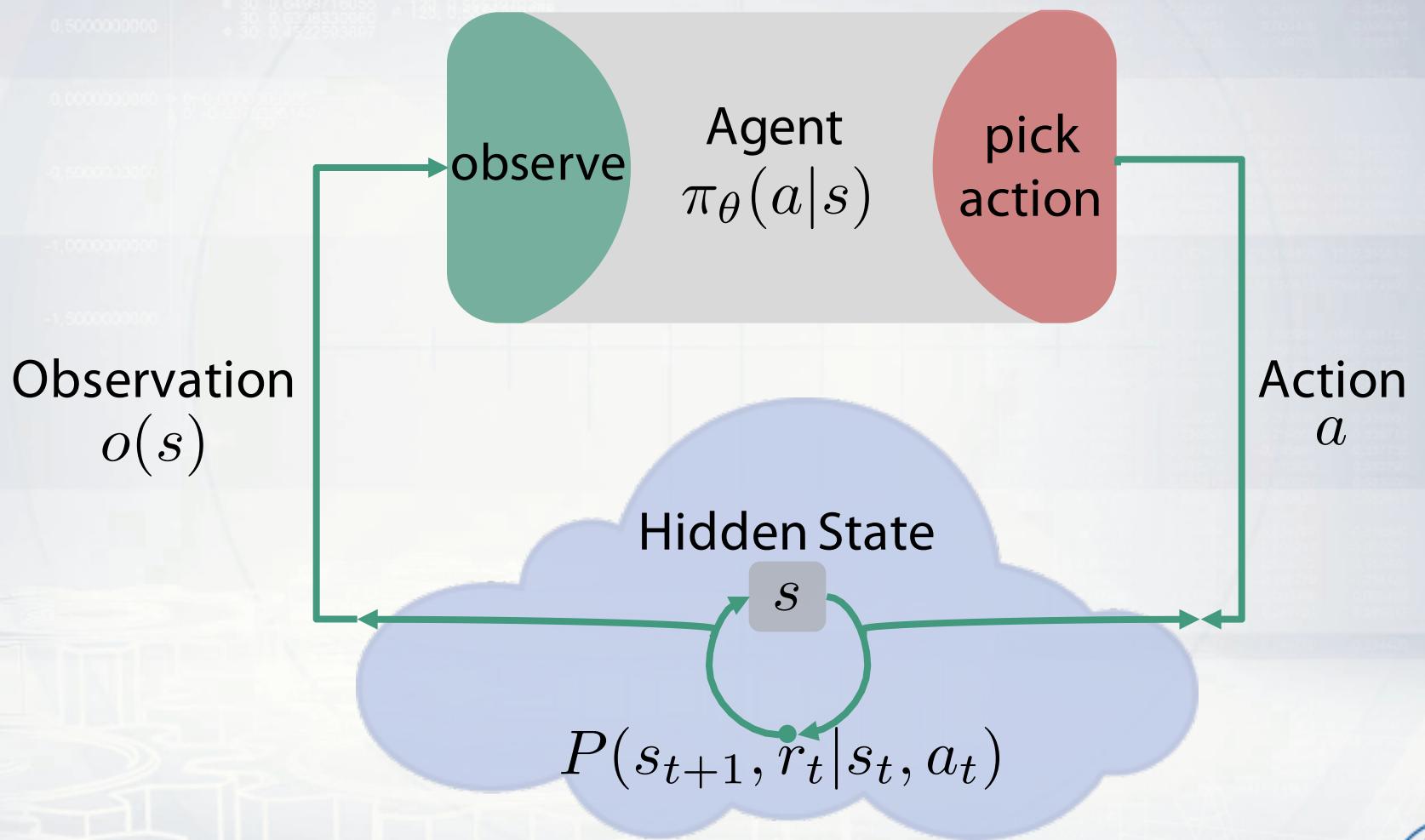
Recap: MDP



Partially Observable MDP



Partially Observable MDP

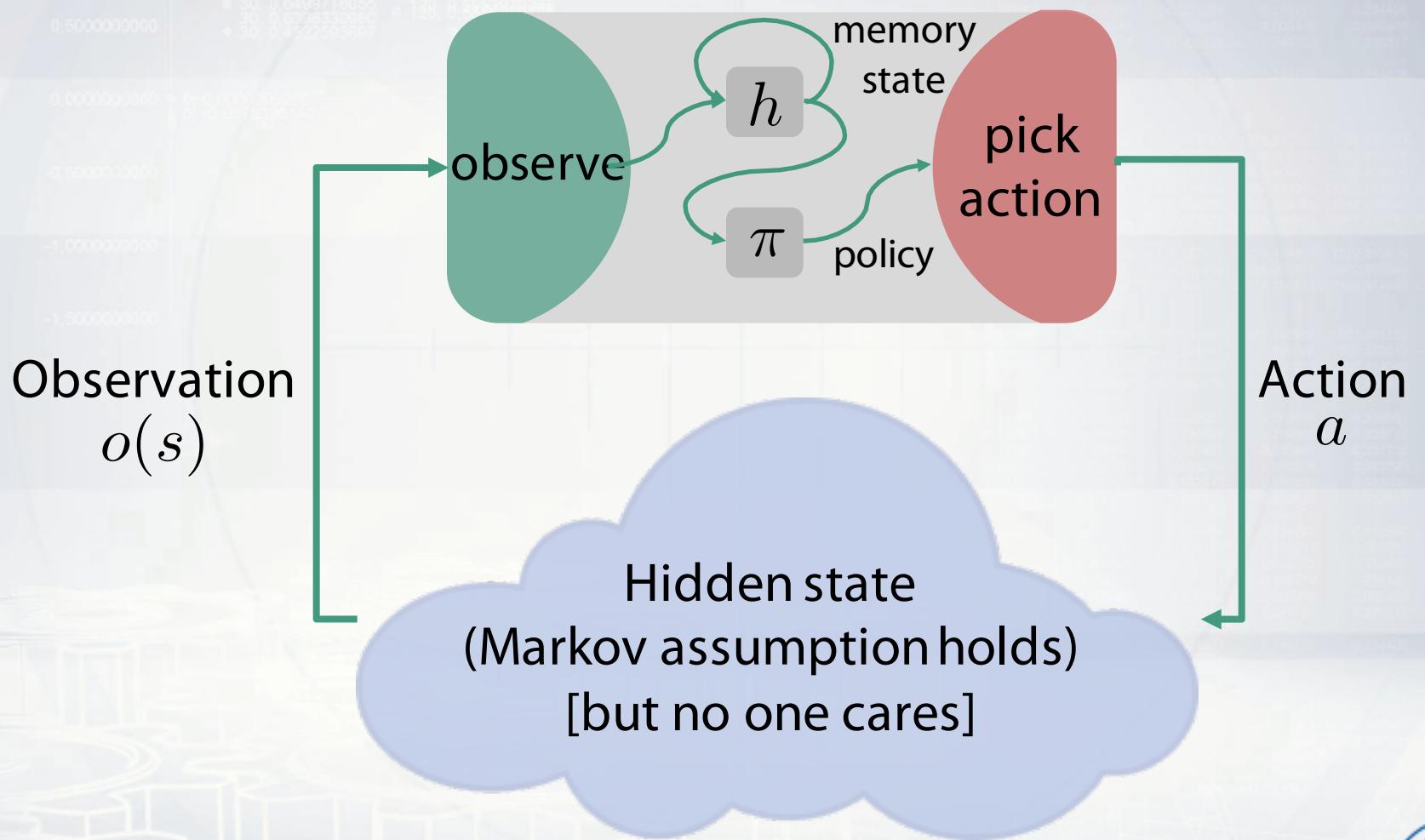


Partially Observable MDP

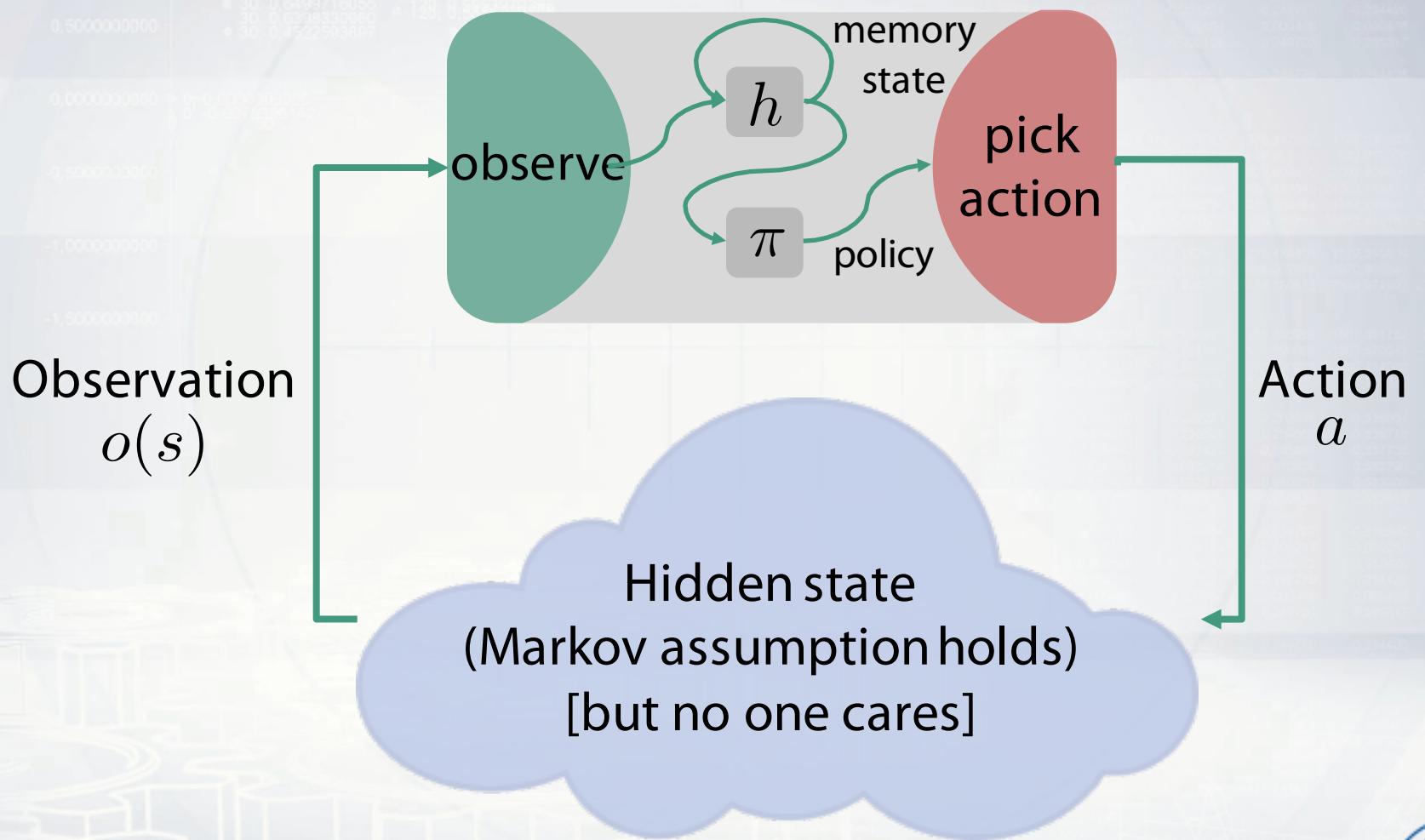
How do we train an agent in such process?



Partially Observable MDP



Partially Observable MDP



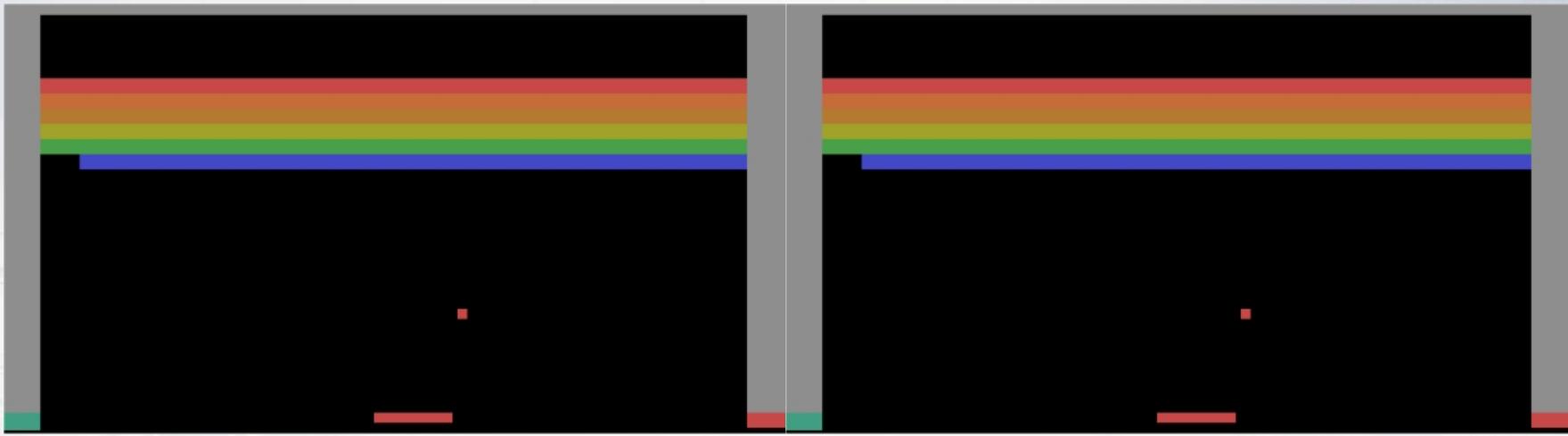
N-gram heuristic

Idea:

$$s_t \neq o(s_t)$$

$$s_t \approx (o(s_{t-n}), a_{t-n}, \dots, o(s_{t-1}), a_{t-1}, o(s_t))$$

e.g. ball movement in breakout

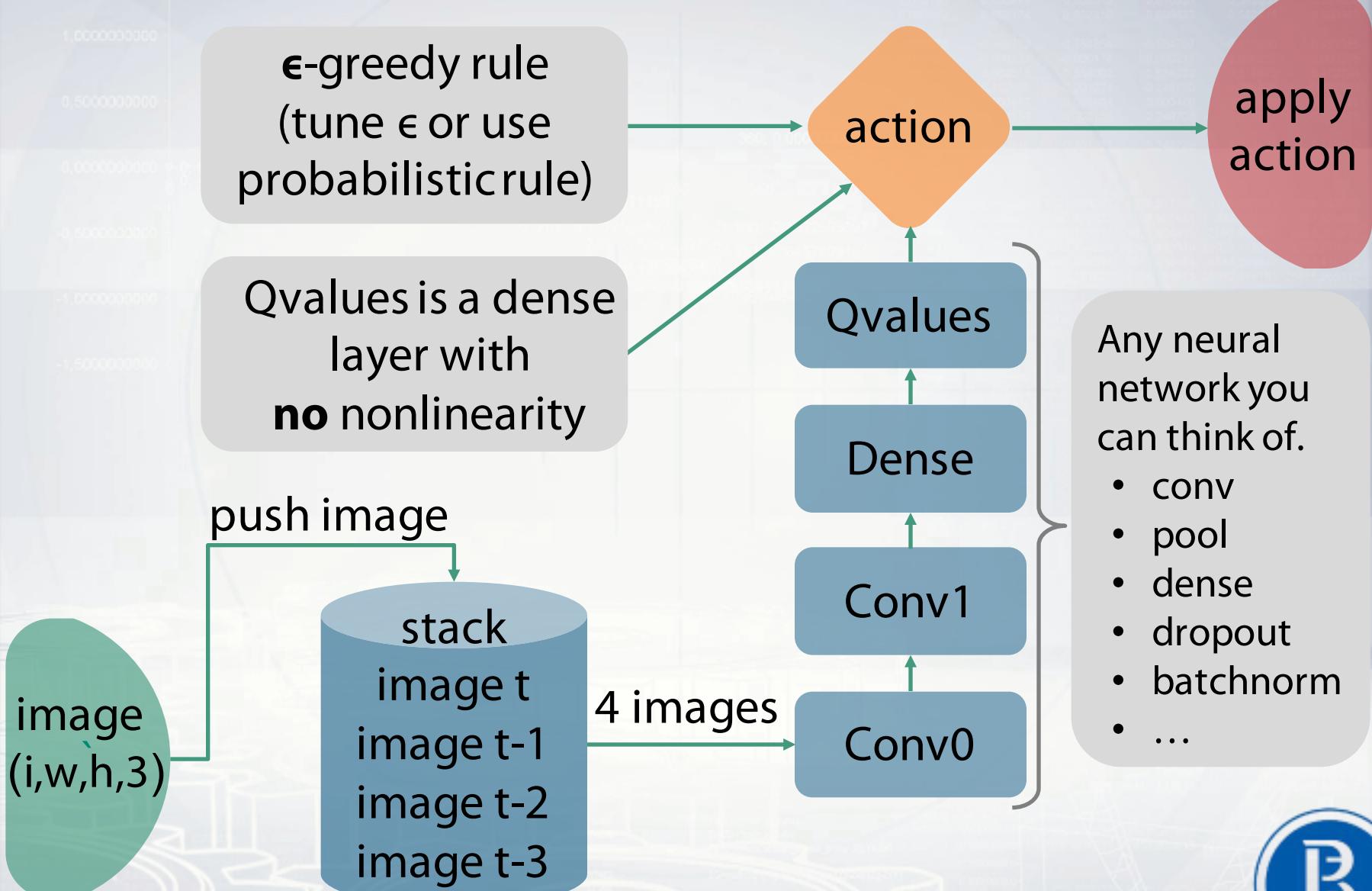


One frame

Several frames



4-frame DQN



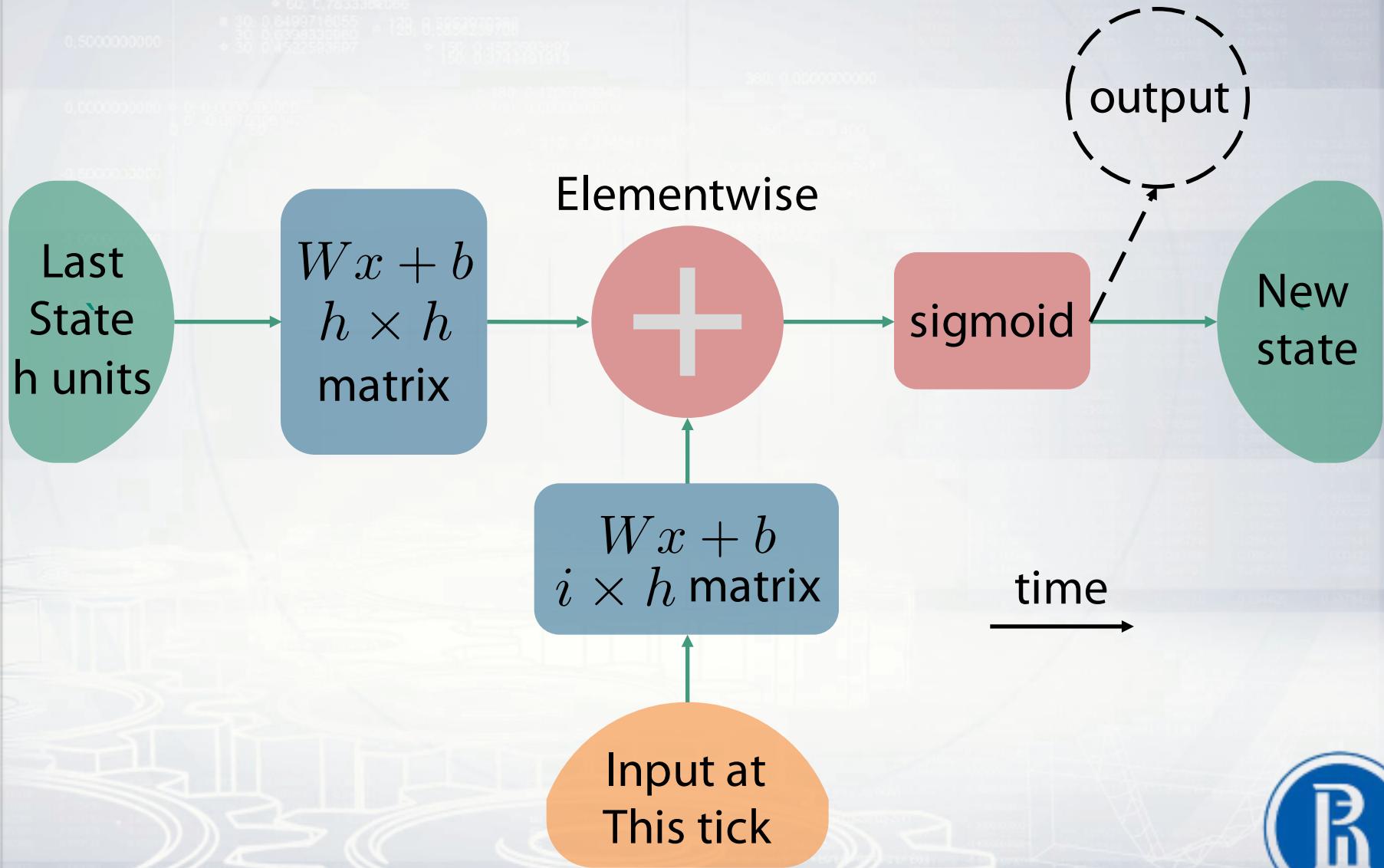
Can we do better?

- There's a hidden process
- You only see some part of it
- You want hidden states

Reminds you of something?



Recurrent neural networks



Recurrent neural networks

▲ 69, 0.8178582142
● 30, 0.8945183284
● 60, 0.7853362010
■ 30, 0.6499716055
● 30, 0.62598330860
● 30, 0.5222583687
■ 120, 0.585289788
● 120, 0.5154373873

300, 0.0000000000

0.0000000000 - 0.5000000000

-0.5000000000 - 1.0000000000

-1.0000000000 - 1.5000000000

-1.5000000000 - 2.0000000000

-2.0000000000 - 2.5000000000

-2.5000000000 - 3.0000000000

-3.0000000000 - 3.5000000000

-3.5000000000 - 4.0000000000

-4.0000000000 - 4.5000000000

-4.5000000000 - 5.0000000000

-5.0000000000 - 5.5000000000

-5.5000000000 - 6.0000000000

-6.0000000000 - 6.5000000000

-6.5000000000 - 7.0000000000

-7.0000000000 - 7.5000000000

-7.5000000000 - 8.0000000000

-8.0000000000 - 8.5000000000

-8.5000000000 - 9.0000000000

-9.0000000000 - 9.5000000000

-9.5000000000 - 10.0000000000

-10.0000000000 - 10.5000000000

-10.5000000000 - 11.0000000000

-11.0000000000 - 11.5000000000

-11.5000000000 - 12.0000000000

-12.0000000000 - 12.5000000000

-12.5000000000 - 13.0000000000

-13.0000000000 - 13.5000000000

-13.5000000000 - 14.0000000000

-14.0000000000 - 14.5000000000

-14.5000000000 - 15.0000000000

-15.0000000000 - 15.5000000000

-15.5000000000 - 16.0000000000

-16.0000000000 - 16.5000000000

-16.5000000000 - 17.0000000000

-17.0000000000 - 17.5000000000

-17.5000000000 - 18.0000000000

-18.0000000000 - 18.5000000000

-18.5000000000 - 19.0000000000

-19.0000000000 - 19.5000000000

-19.5000000000 - 20.0000000000

-20.0000000000 - 20.5000000000

-20.5000000000 - 21.0000000000

-21.0000000000 - 21.5000000000

-21.5000000000 - 22.0000000000

-22.0000000000 - 22.5000000000

-22.5000000000 - 23.0000000000

-23.0000000000 - 23.5000000000

-23.5000000000 - 24.0000000000

-24.0000000000 - 24.5000000000

-24.5000000000 - 25.0000000000

-25.0000000000 - 25.5000000000

-25.5000000000 - 26.0000000000

-26.0000000000 - 26.5000000000

-26.5000000000 - 27.0000000000

-27.0000000000 - 27.5000000000

-27.5000000000 - 28.0000000000

-28.0000000000 - 28.5000000000

-28.5000000000 - 29.0000000000

-29.0000000000 - 29.5000000000

-29.5000000000 - 30.0000000000

-30.0000000000 - 30.5000000000

-30.5000000000 - 31.0000000000

-31.0000000000 - 31.5000000000

-31.5000000000 - 32.0000000000

-32.0000000000 - 32.5000000000

-32.5000000000 - 33.0000000000

-33.0000000000 - 33.5000000000

-33.5000000000 - 34.0000000000

-34.0000000000 - 34.5000000000

-34.5000000000 - 35.0000000000

-35.0000000000 - 35.5000000000

-35.5000000000 - 36.0000000000

-36.0000000000 - 36.5000000000

-36.5000000000 - 37.0000000000

-37.0000000000 - 37.5000000000

-37.5000000000 - 38.0000000000

-38.0000000000 - 38.5000000000

-38.5000000000 - 39.0000000000

-39.0000000000 - 39.5000000000

-39.5000000000 - 40.0000000000

-40.0000000000 - 40.5000000000

-40.5000000000 - 41.0000000000

-41.0000000000 - 41.5000000000

-41.5000000000 - 42.0000000000

-42.0000000000 - 42.5000000000

-42.5000000000 - 43.0000000000

-43.0000000000 - 43.5000000000

-43.5000000000 - 44.0000000000

-44.0000000000 - 44.5000000000

-44.5000000000 - 45.0000000000

-45.0000000000 - 45.5000000000

-45.5000000000 - 46.0000000000

-46.0000000000 - 46.5000000000

-46.5000000000 - 47.0000000000

-47.0000000000 - 47.5000000000

-47.5000000000 - 48.0000000000

-48.0000000000 - 48.5000000000

-48.5000000000 - 49.0000000000

-49.0000000000 - 50.0000000000

-50.0000000000 - 51.0000000000

-51.0000000000 - 52.0000000000

-52.0000000000 - 53.0000000000

-53.0000000000 - 54.0000000000

-54.0000000000 - 55.0000000000

-55.0000000000 - 56.0000000000

-56.0000000000 - 57.0000000000

-57.0000000000 - 58.0000000000

-58.0000000000 - 59.0000000000

-59.0000000000 - 60.0000000000

-60.0000000000 - 61.0000000000

-61.0000000000 - 62.0000000000

-62.0000000000 - 63.0000000000

-63.0000000000 - 64.0000000000

-64.0000000000 - 65.0000000000

-65.0000000000 - 66.0000000000

-66.0000000000 - 67.0000000000

-67.0000000000 - 68.0000000000

-68.0000000000 - 69.0000000000

-69.0000000000 - 70.0000000000

-70.0000000000 - 71.0000000000

-71.0000000000 - 72.0000000000

-72.0000000000 - 73.0000000000

-73.0000000000 - 74.0000000000

-74.0000000000 - 75.0000000000

-75.0000000000 - 76.0000000000

-76.0000000000 - 77.0000000000

-77.0000000000 - 78.0000000000

-78.0000000000 - 79.0000000000

-79.0000000000 - 80.0000000000

-80.0000000000 - 81.0000000000

-81.0000000000 - 82.0000000000

-82.0000000000 - 83.0000000000

-83.0000000000 - 84.0000000000

-84.0000000000 - 85.0000000000

-85.0000000000 - 86.0000000000

-86.0000000000 - 87.0000000000

-87.0000000000 - 88.0000000000

-88.0000000000 - 89.0000000000

-89.0000000000 - 90.0000000000

-90.0000000000 - 91.0000000000

-91.0000000000 - 92.0000000000

-92.0000000000 - 93.0000000000

-93.0000000000 - 94.0000000000

-94.0000000000 - 95.0000000000

-95.0000000000 - 96.0000000000

-96.0000000000 - 97.0000000000

-97.0000000000 - 98.0000000000

-98.0000000000 - 99.0000000000

-99.0000000000 - 100.0000000000

-100.0000000000 - 101.0000000000

-101.0000000000 - 102.0000000000

-102.0000000000 - 103.0000000000

-103.0000000000 - 104.0000000000

-104.0000000000 - 105.0000000000

-105.0000000000 - 106.0000000000

-106.0000000000 - 107.0000000000

-107.0000000000 - 108.0000000000

-108.0000000000 - 109.0000000000

-109.0000000000 - 110.0000000000

-110.0000000000 - 111.0000000000

-111.0000000000 - 112.0000000000

-112.0000000000 - 113.0000000000

-113.0000000000 - 114.0000000000

-114.0000000000 - 115.0000000000

-115.0000000000 - 116.0000000000

-116.0000000000 - 117.0000000000

-117.0000000000 - 118.0000000000

-118.0000000000 - 119.0000000000

-119.0000000000 - 120.0000000000

-120.0000000000 - 121.0000000000

-121.0000000000 - 122.0000000000

-122.0000000000 - 123.0000000000

-123.0000000000 - 124.0000000000

-124.0000000000 - 125.0000000000

-125.0000000000 - 126.0000000000

-126.0000000000 - 127.0000000000

-127.0000000000 - 128.0000000000

-128.0000000000 - 129.0000000000

-129.0000000000 - 130.0000000000

-130.0000000000 - 131.0000000000

-131.0000000000 - 132.0000000000

-132.0000000000 - 133.0000000000

-133.0000000000 - 134.0000000000

-134.0000000000 - 135.0000000000

-135.0000000000 - 136.0000000000

-136.0000000000 - 137.0000000000

-137.0000000000 - 138.0000000000

-138.0000000000 - 139.0000000000

-139.0000000000 - 140.0000000000

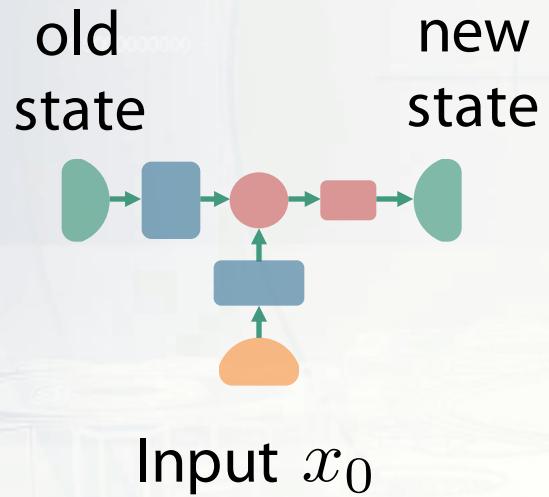
-140.0000000000 - 141.0000000000

-141.0000000000 - 142.0000000000

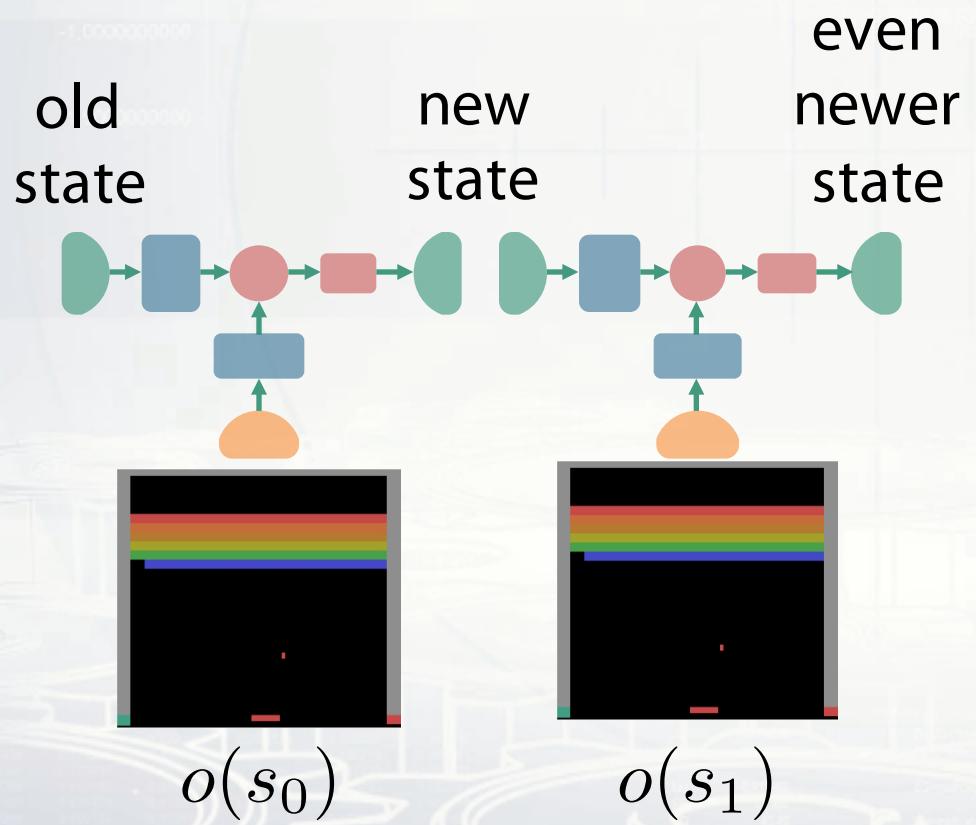
-142.0000000000 - 143.0000000000

-143.000000000

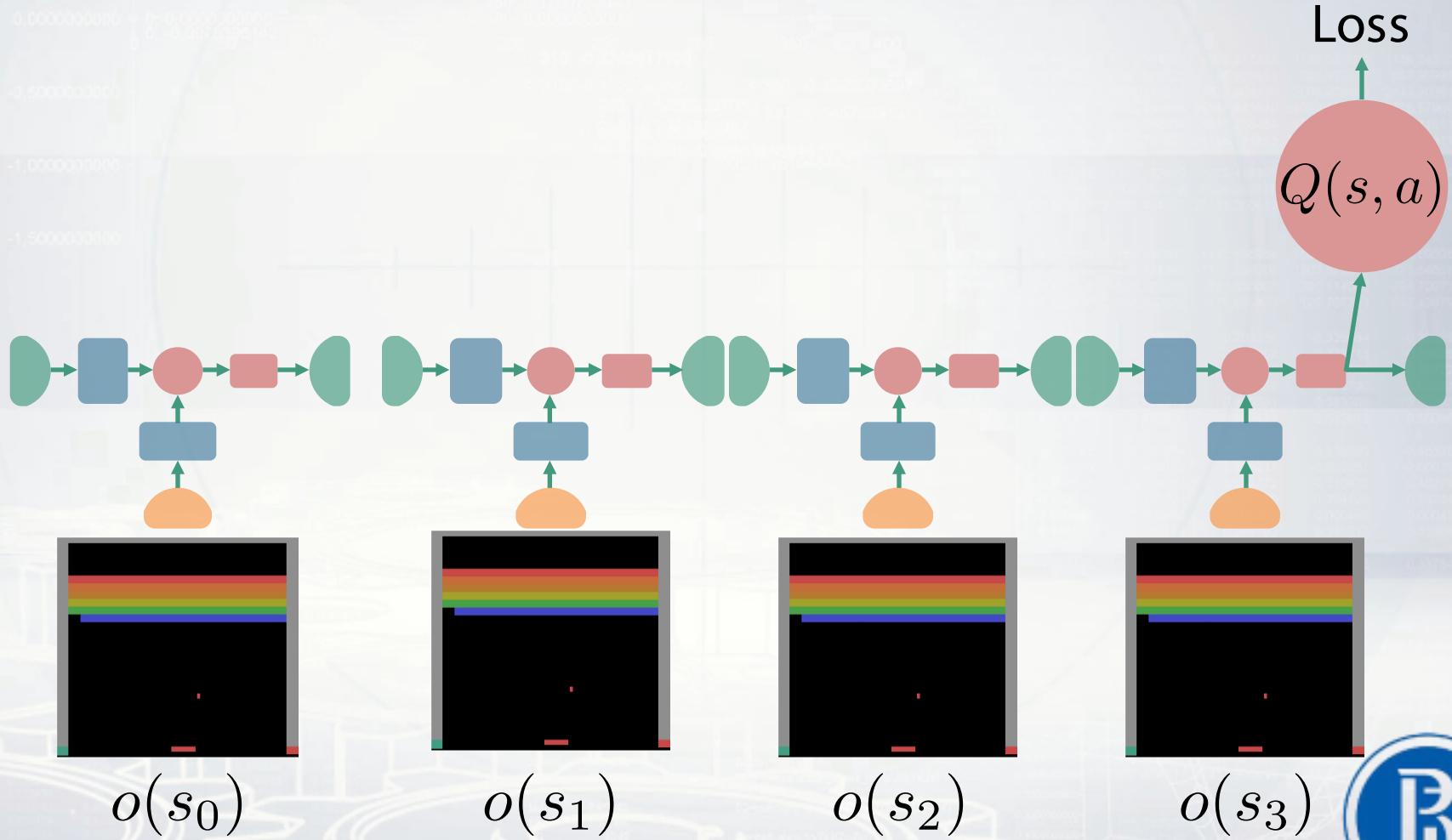
Recurrent neural networks



Recurrent neural networks



Recurrent neural networks

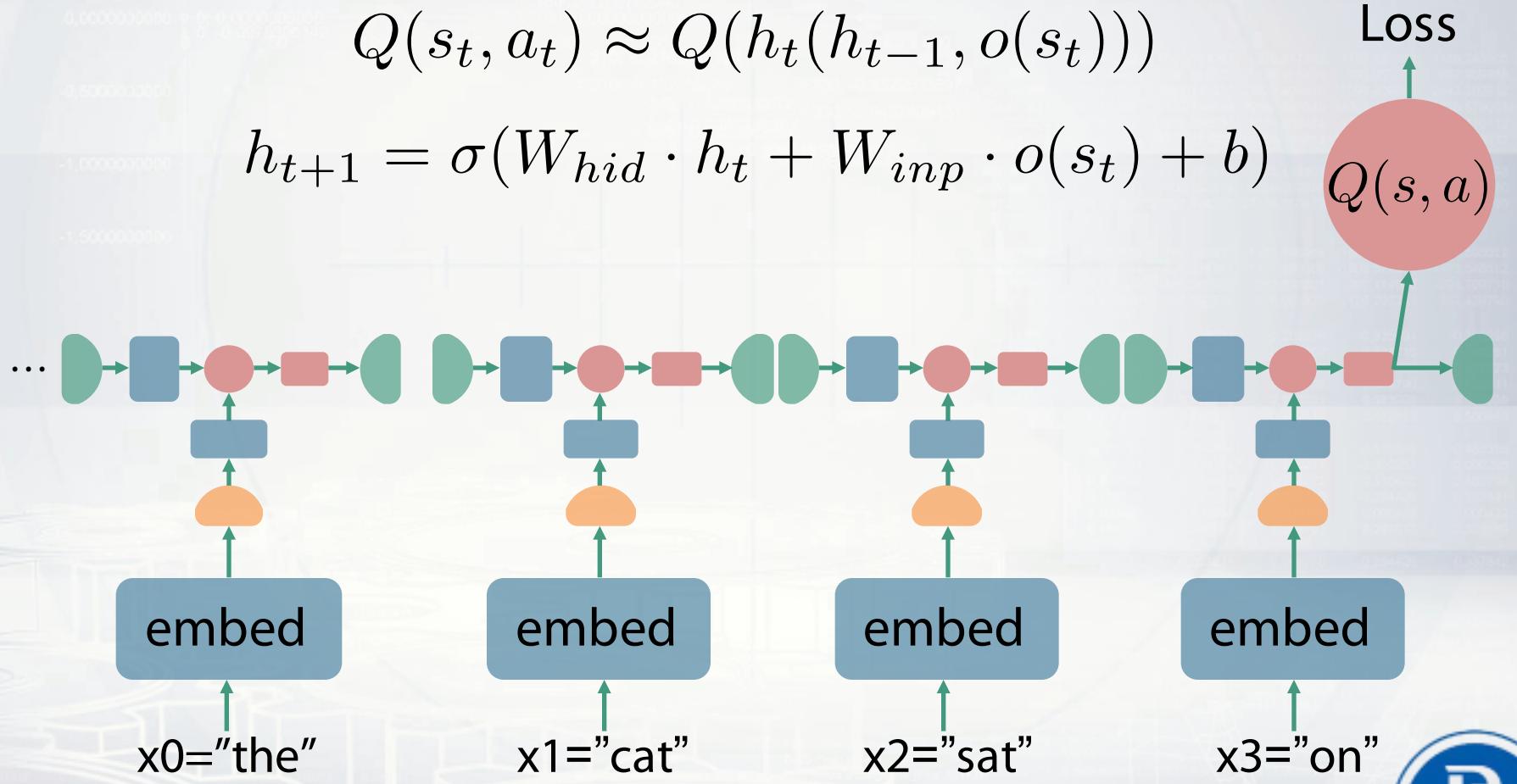


Recurrent neural networks

$$L = (Q(s_t, a_t) - [r + \gamma \cdot \max_{a'} Q(s_{t+1}, a')])^2$$

$$Q(s_t, a_t) \approx Q(h_t(h_{t-1}, o(s_t)))$$

$$h_{t+1} = \sigma(W_{hid} \cdot h_t + W_{inp} \cdot o(s_t) + b)$$



Recurrent neural networks

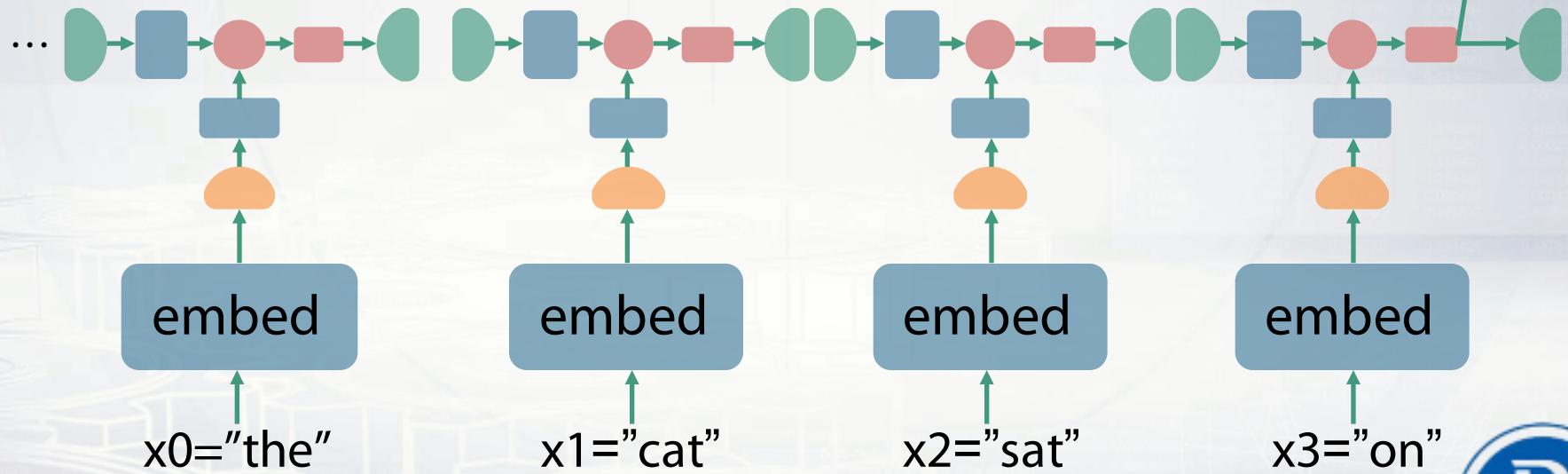
$$L = (Q(s_t, a_t) - [r + \gamma \cdot \max_{a'} Q(s_{t+1}, a')])^2$$

$$Q(s_t, a_t) \approx Q(h_t(h_{t-1}, o(s_t)))$$

$$h_{t+1} = \sigma(W_{hid} \cdot h_t + W_{inp} \cdot o(s_t) + b)$$

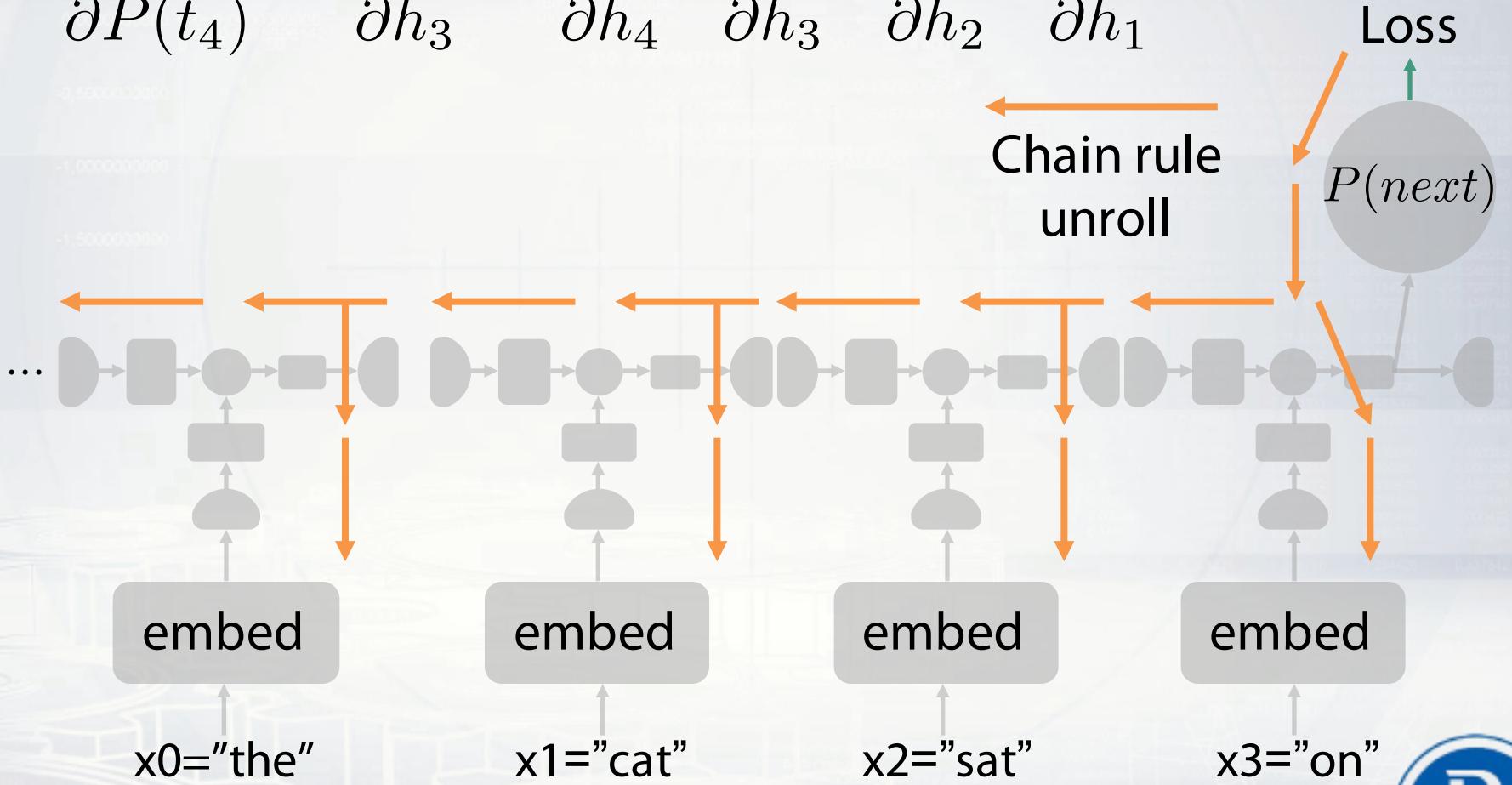


How do we train it?



Backpropagation through time

$$\frac{\partial Loss}{\partial P(t_4)} \cdot \frac{\partial P(t_4)}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_4} \cdot \frac{\partial h_2}{\partial h_3} \cdot \frac{\partial h_1}{\partial h_2} \cdot \frac{\partial h_0}{\partial h_1} \cdot \dots$$



Idea:

- use LSTM on top of DQN
- predict Q-values from hidden state
- trains on partial-trajectories

<image here>



Results



Todo add table