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: $(s', r) \backsim G(s, a)$

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: $(s', r) \backsim G(s, a)$

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: $(s', r) \backsim G(s, a)$

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

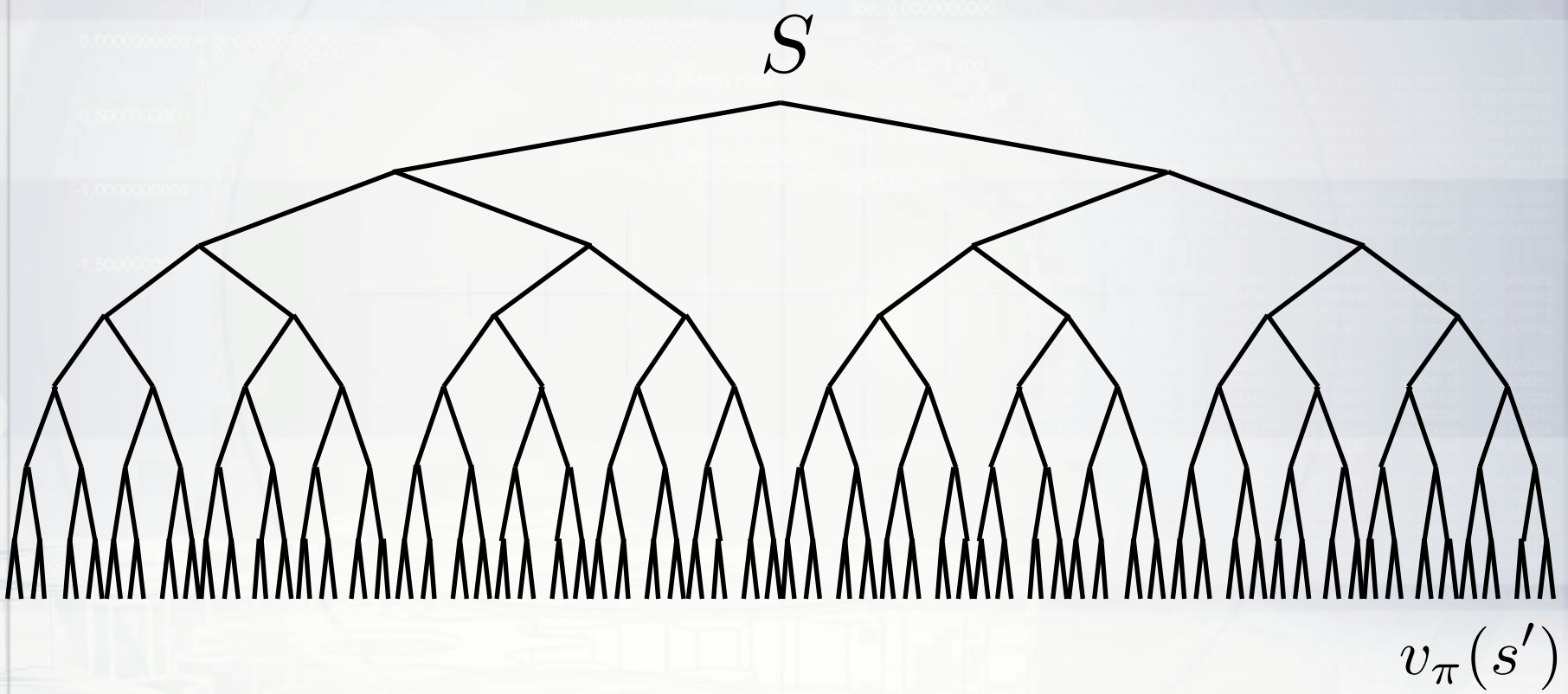
Types of planning

Model-free – **learn** the policy by trial and error

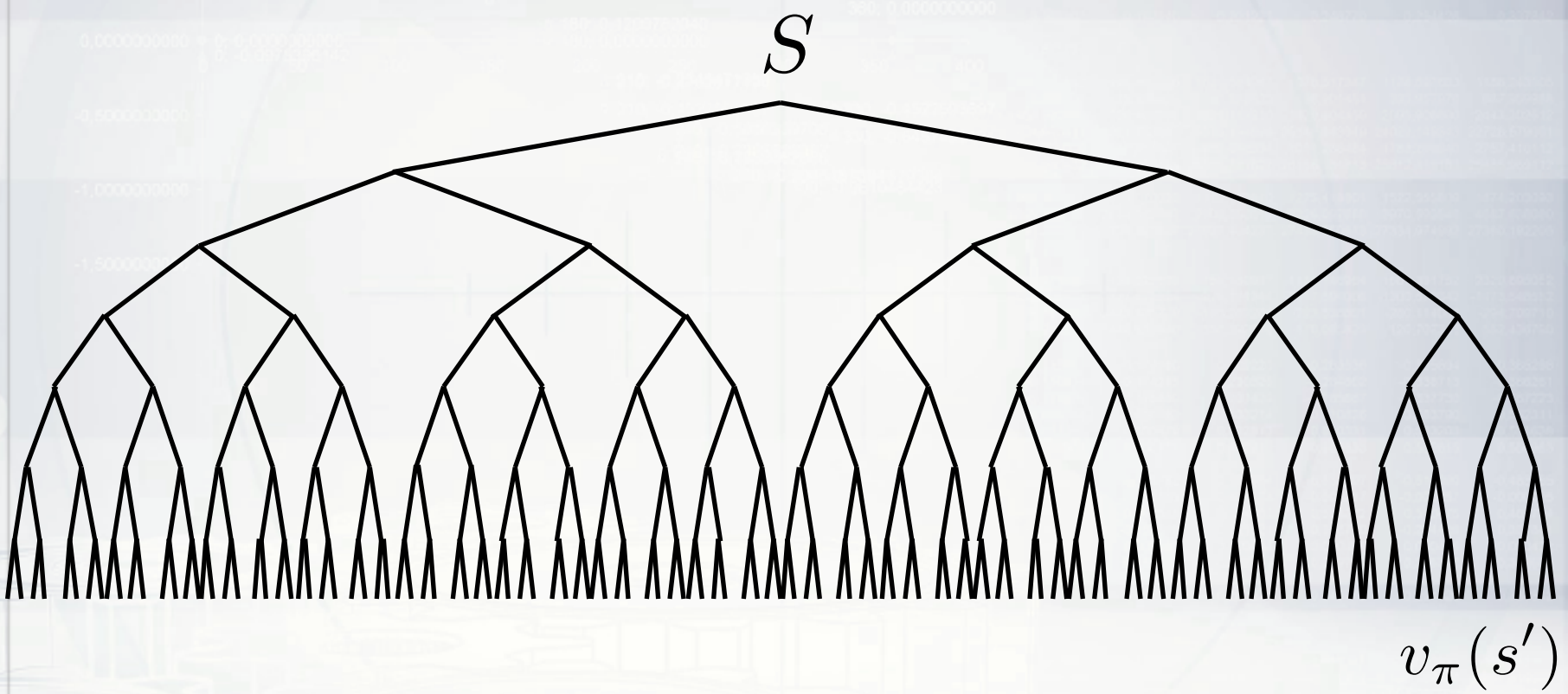
Model-based – **plan** to obtain the policy

- 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

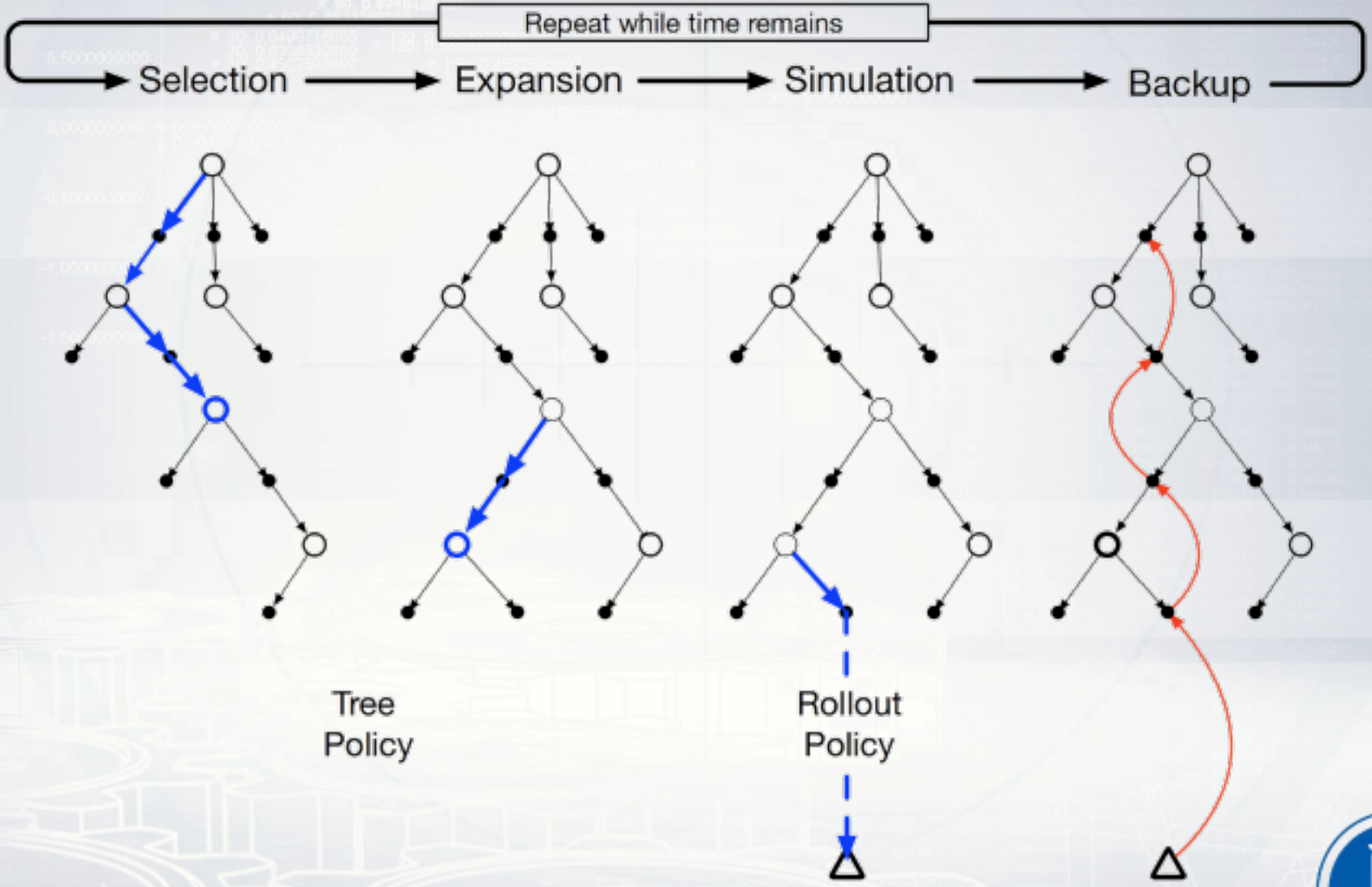
- Rollout algorithm which

Monte Carlo Tree Search – basic parts

Planning

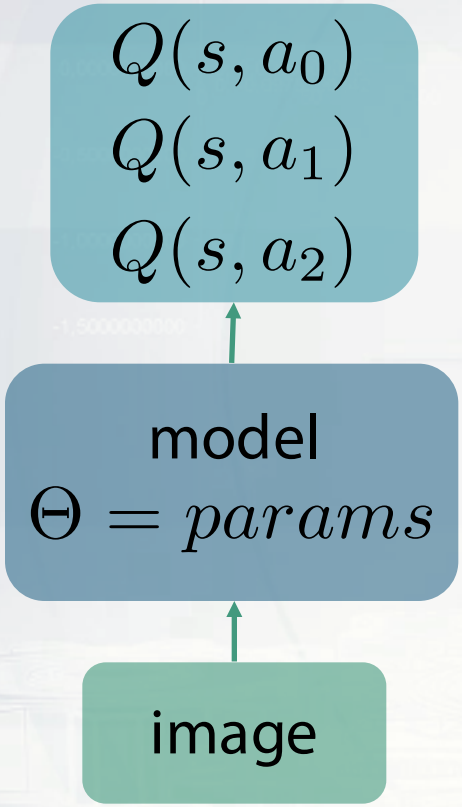
Planning

Scheme of Monte Carlo Tree Search



Авторство? Своя картинка или перерисовать?

Recap: Approximate Q-learning



Q-values:

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

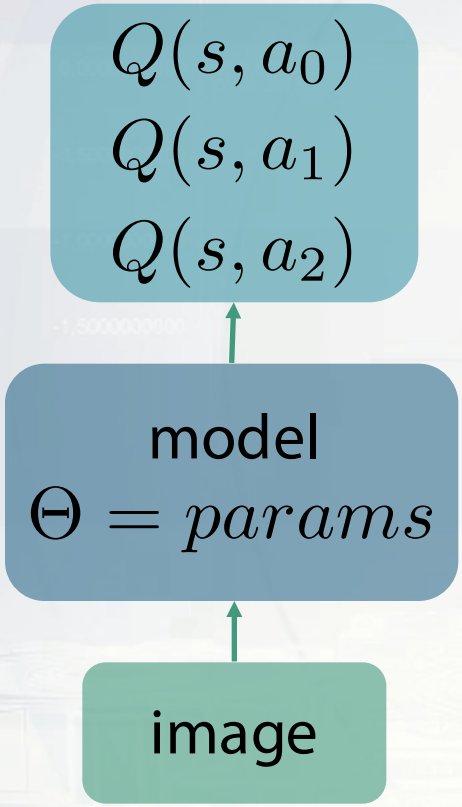
Objective:

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

Gradient step:

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

Recap: Approximate Q-learning



Q-values:

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

Objective:

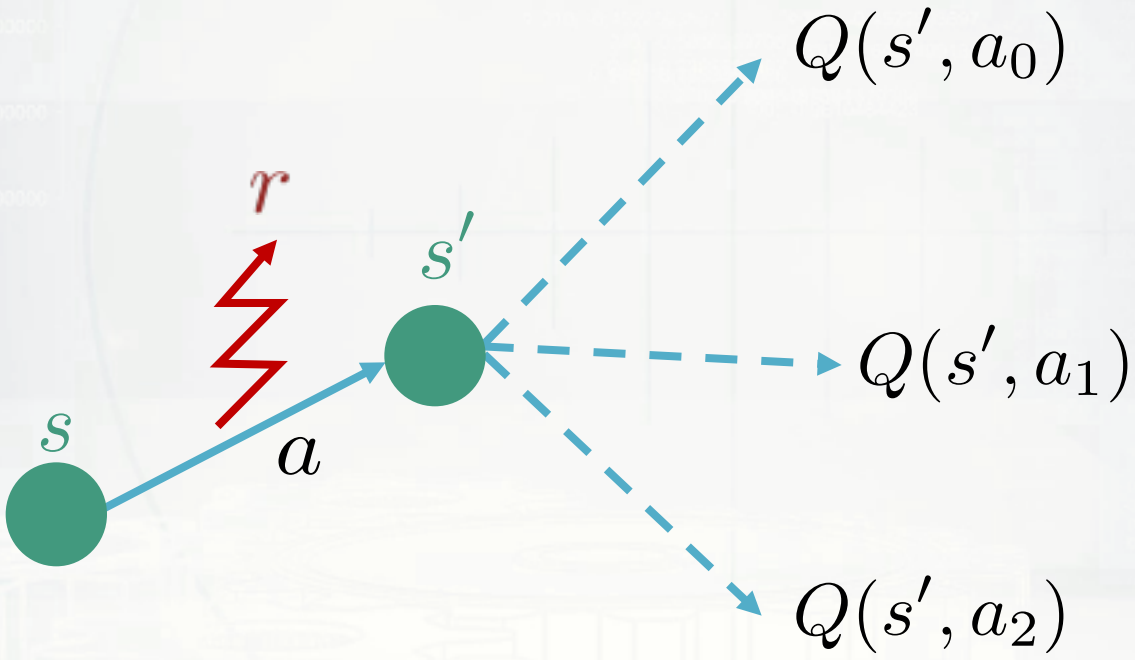
$$L = (Q(s_t, a_t) - [r + \gamma \cdot max_{a'} Q(s_{t+1}, a')])^2$$

consider const

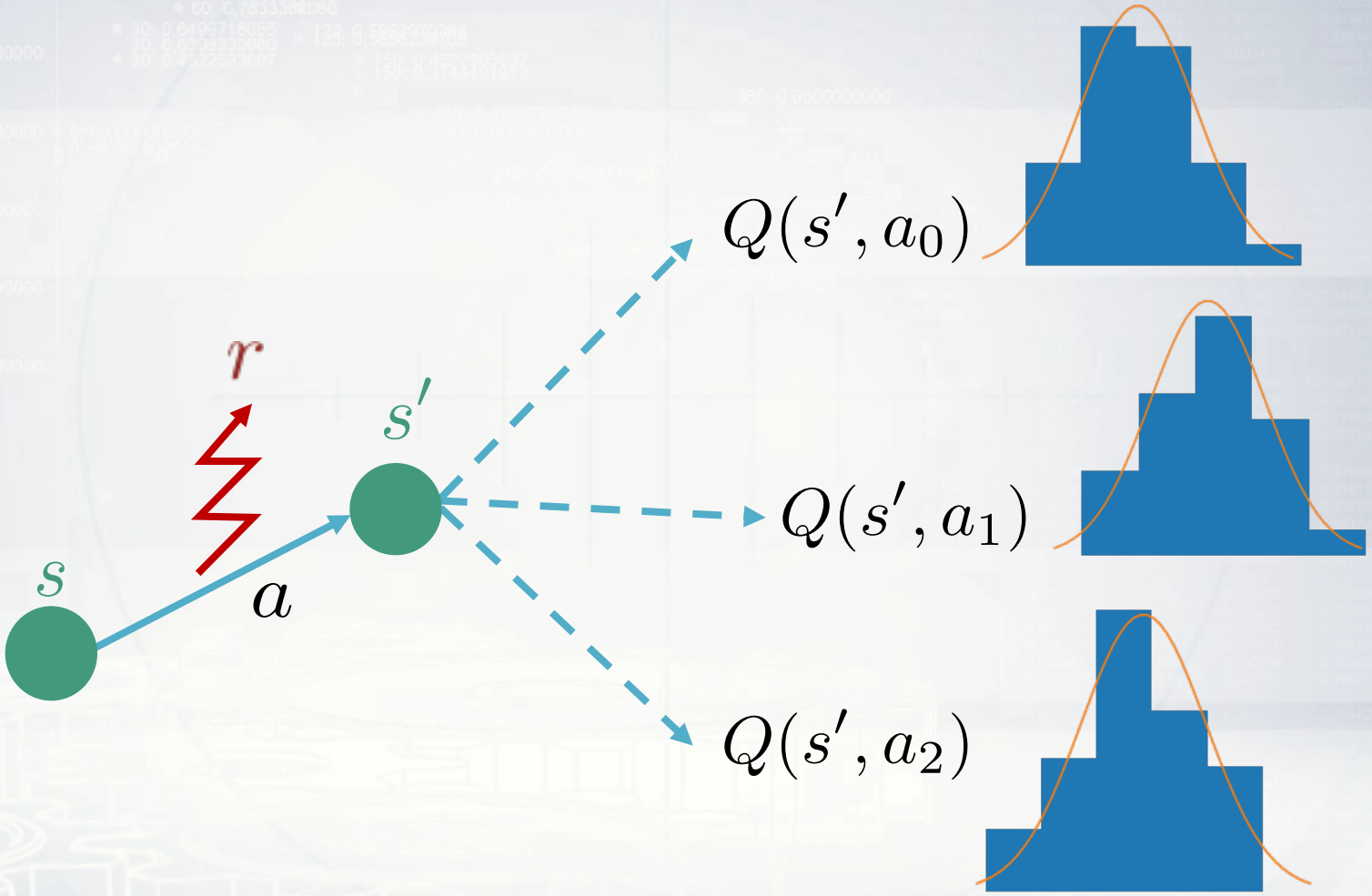
Gradient step:

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

We have a problem

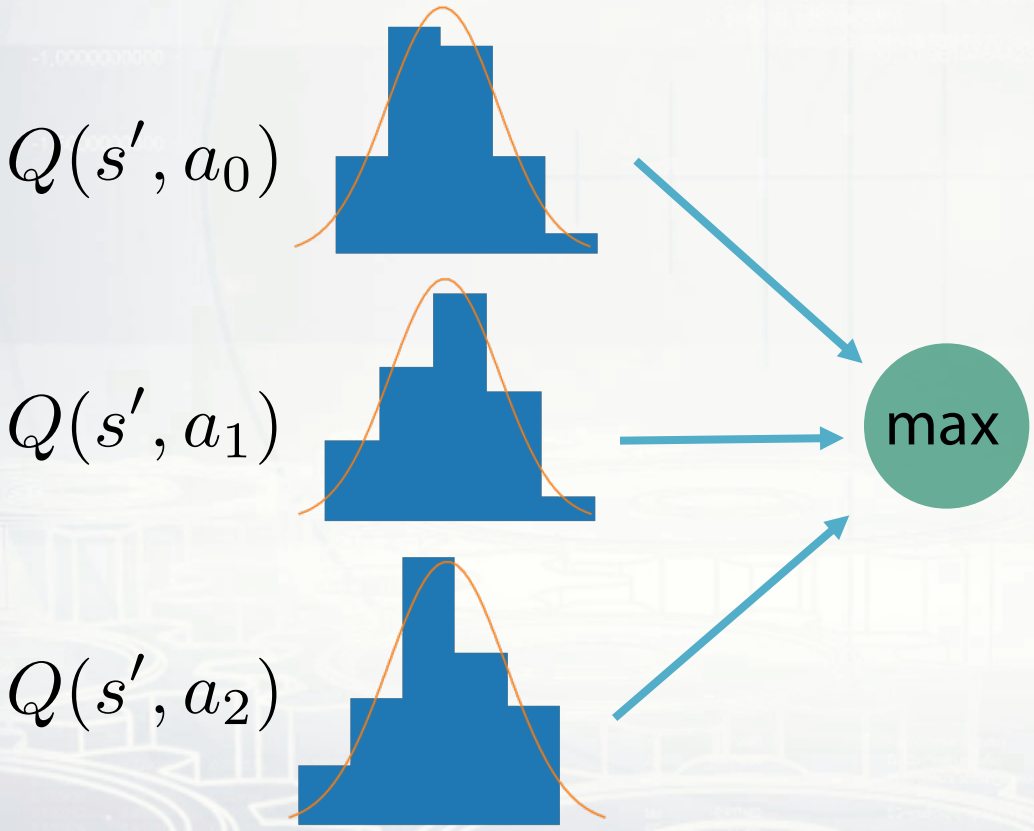


We have a problem



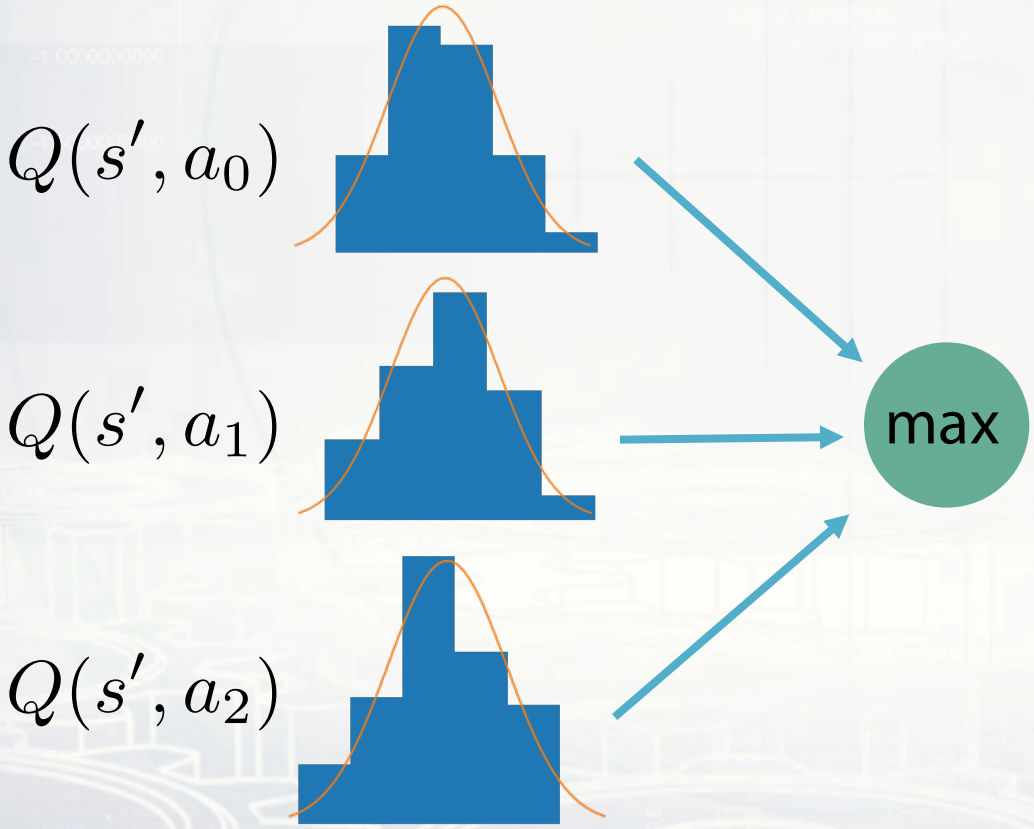
We have a problem

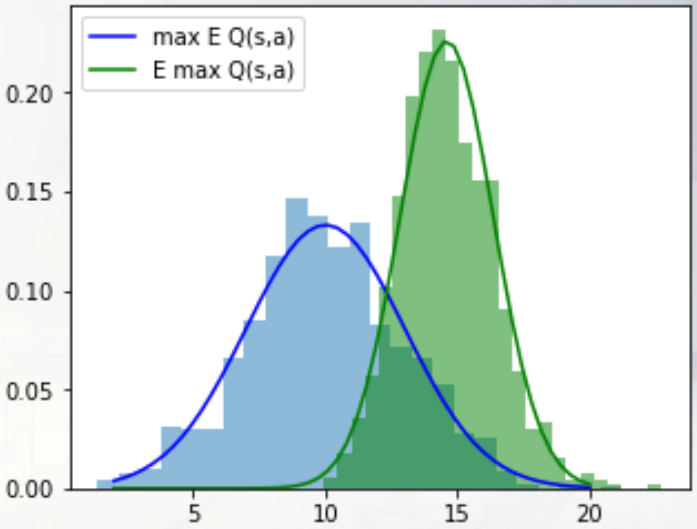
Q-value update: $\hat{Q}(s_t, a_t) = r + \gamma \cdot max_{a'} \hat{Q}(s_{t+1}, a')$



We have a problem

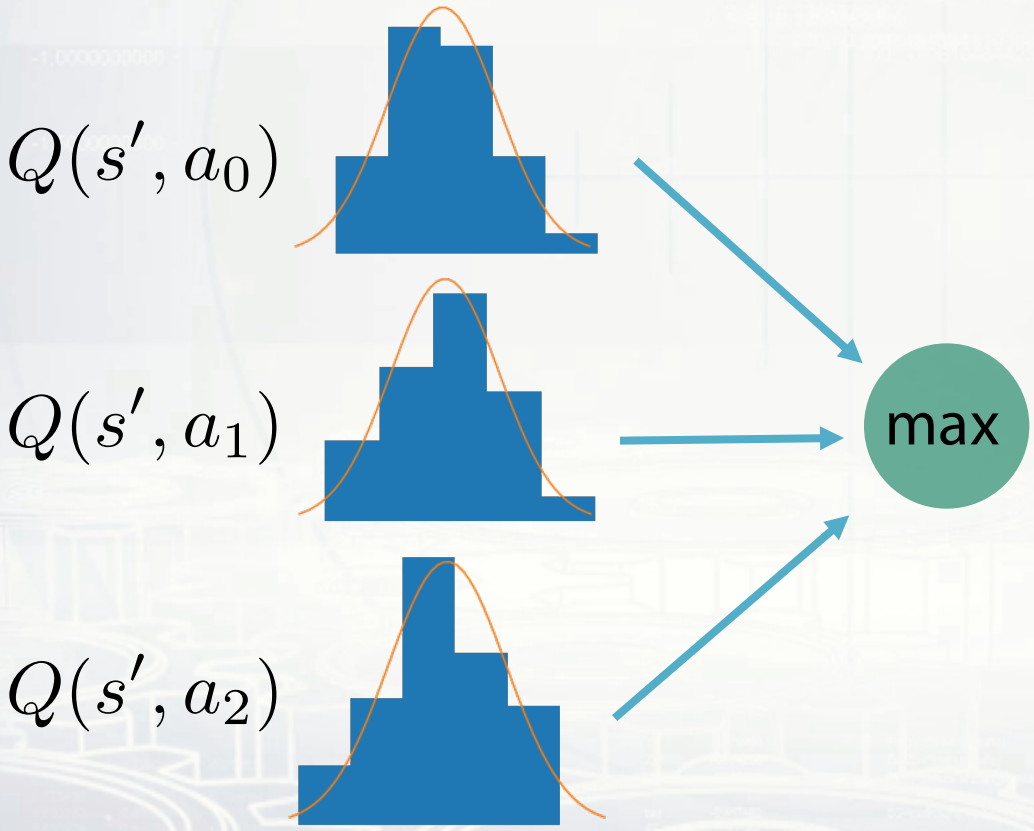
$$\underset{s'}{E}\ \max_{a'}\ \hat{Q}(s_{t+1}, a') \geq \max_{a'}\ \underset{s'}{E}\ \hat{Q}(s_{t+1}, a')$$





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 $Q_1(\textit{terminal-state},\cdot) = Q_2(\textit{terminal-state},\cdot) = 0$
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 probabilility:

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

    else:

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

    $S \leftarrow S'$
  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

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

$$\downarrow$$

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

Double Q-learning

Idea 2: use target network instead of Q_2

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

$$\downarrow$$

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

$$\downarrow$$

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

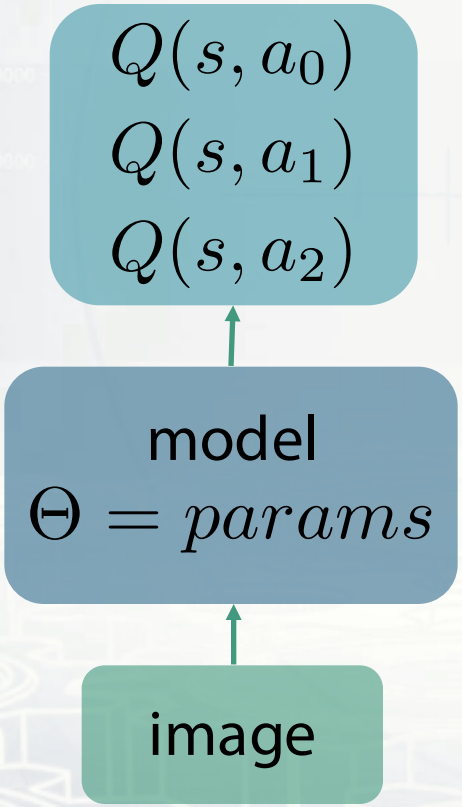
We have a problem #2

Let's watch the video
https://www.youtube.com/watch?v=UXurvvDY93o

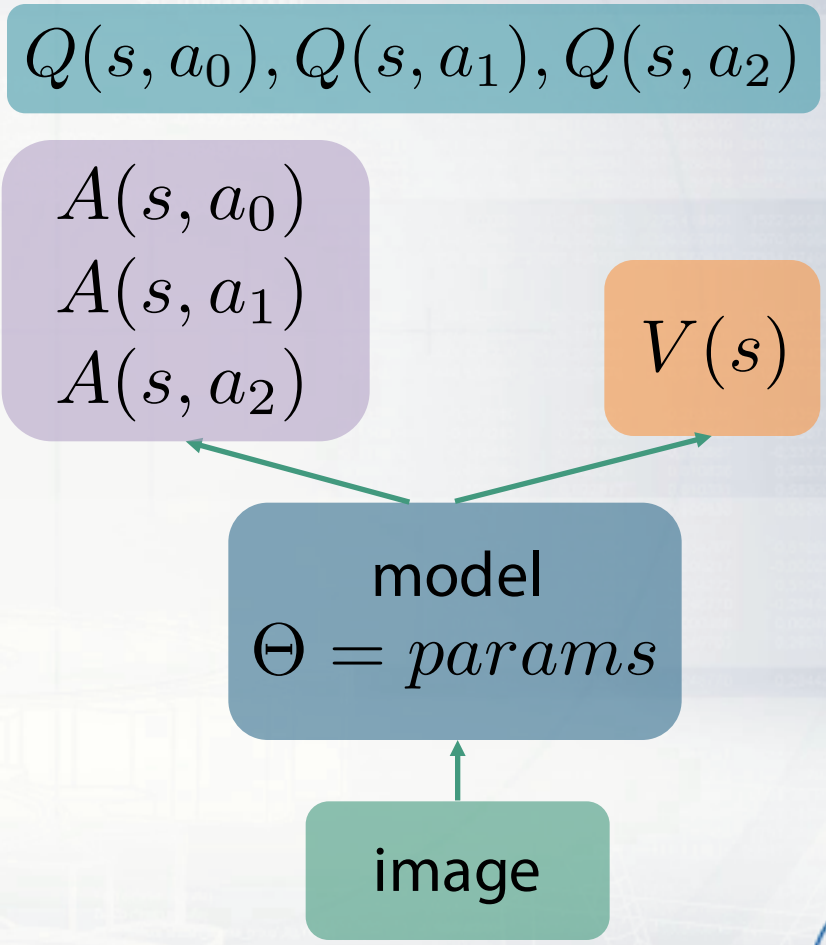
We have a problem #2

Defining $Q = V + A$

DQN



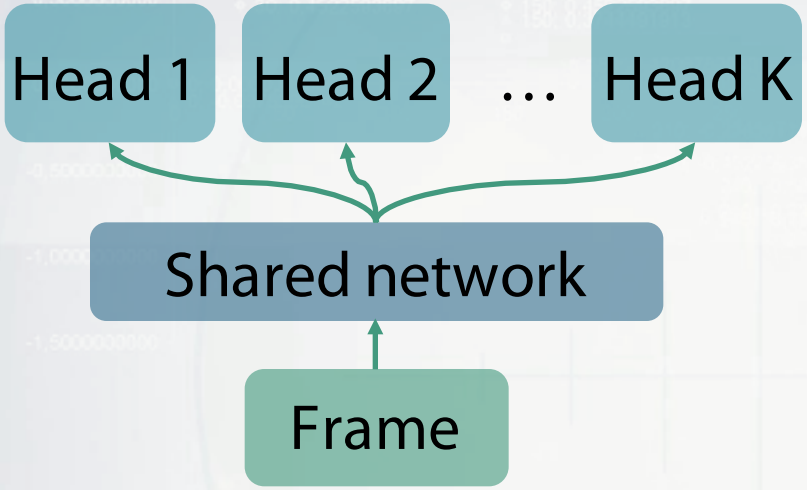
Dueling DQN



We have a problem #2

Substract mean vs max

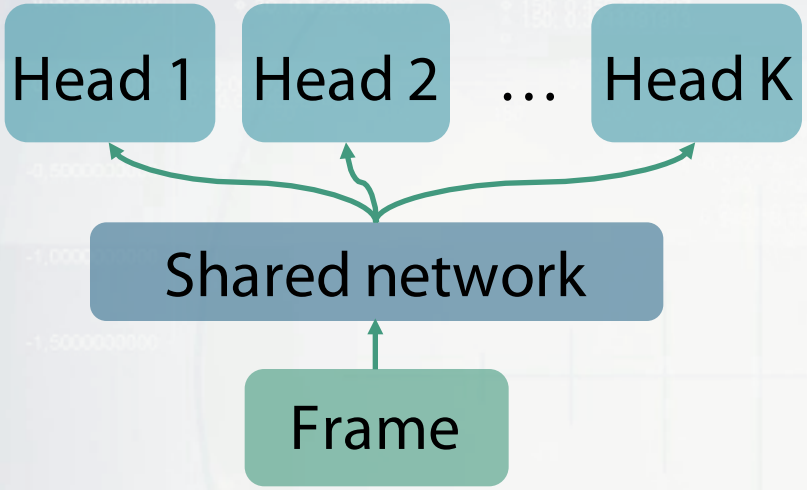
Bootstrap DQN



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).

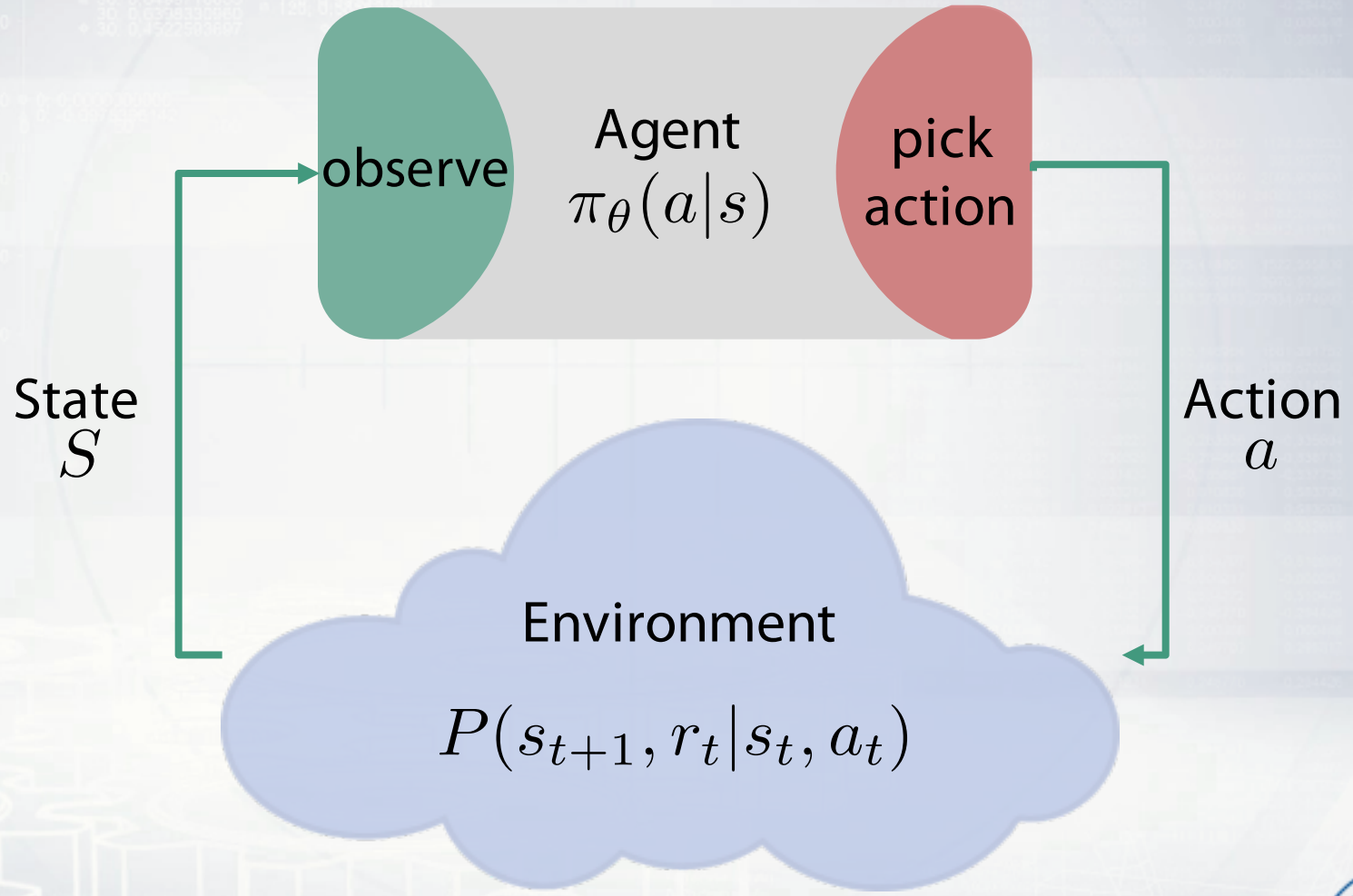
Problem:

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

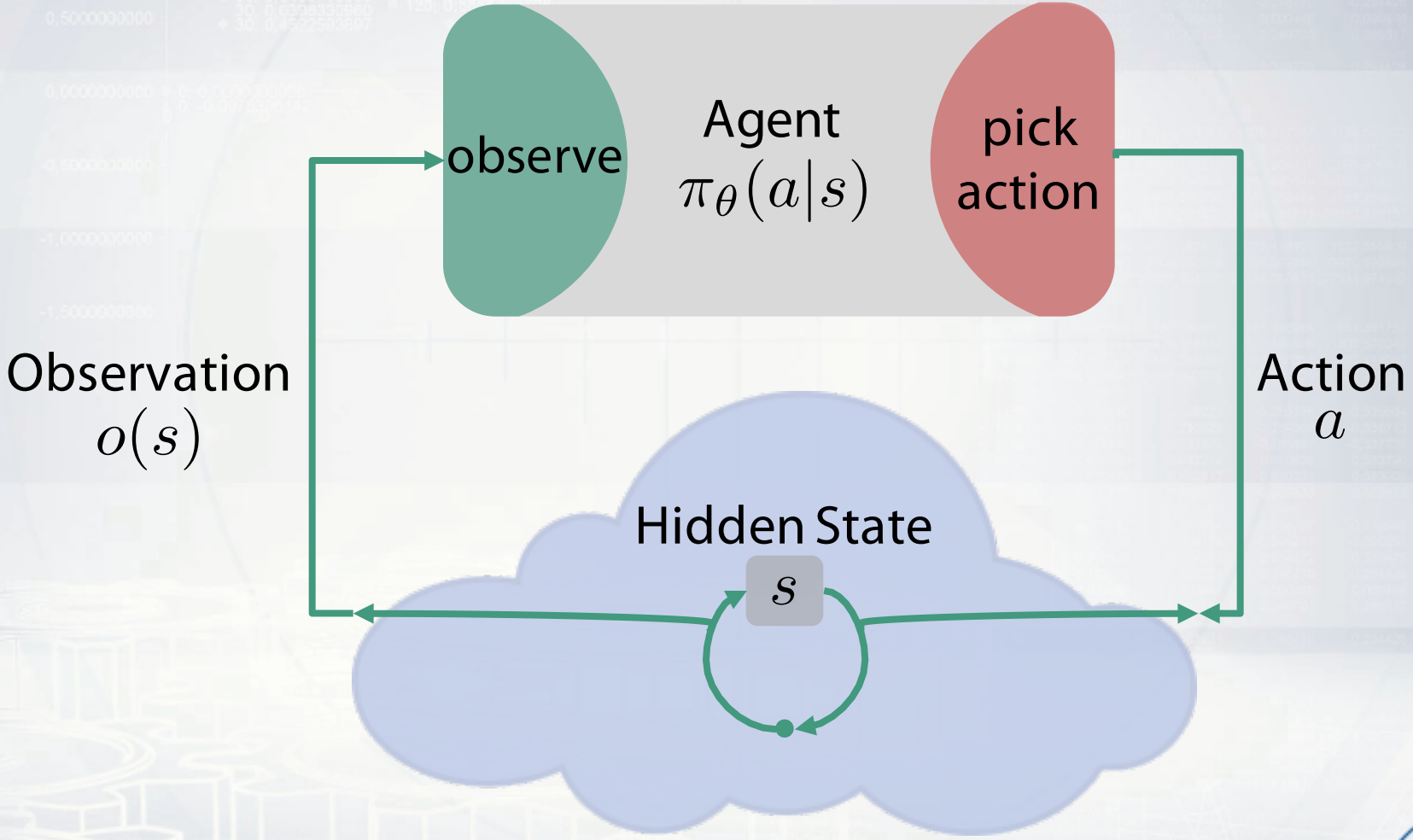
Examples:

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

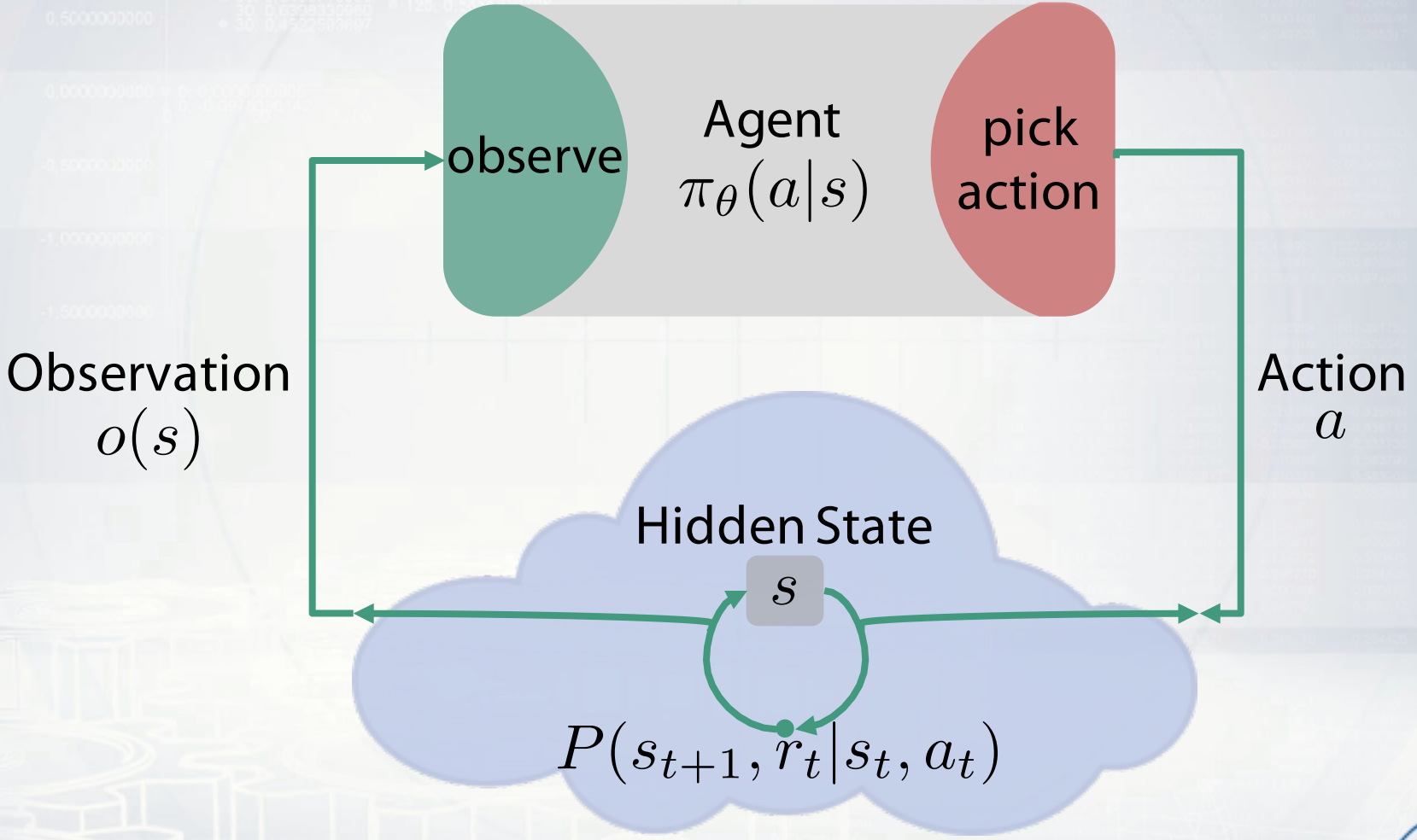
Recap: MDP


observe
Agent
$\pi_\theta(a|s)$
pick action
State
S
Action
a
Environment
$P(s_{t+1}, r_t|s_t, a_t)$

Partially Observable MDP



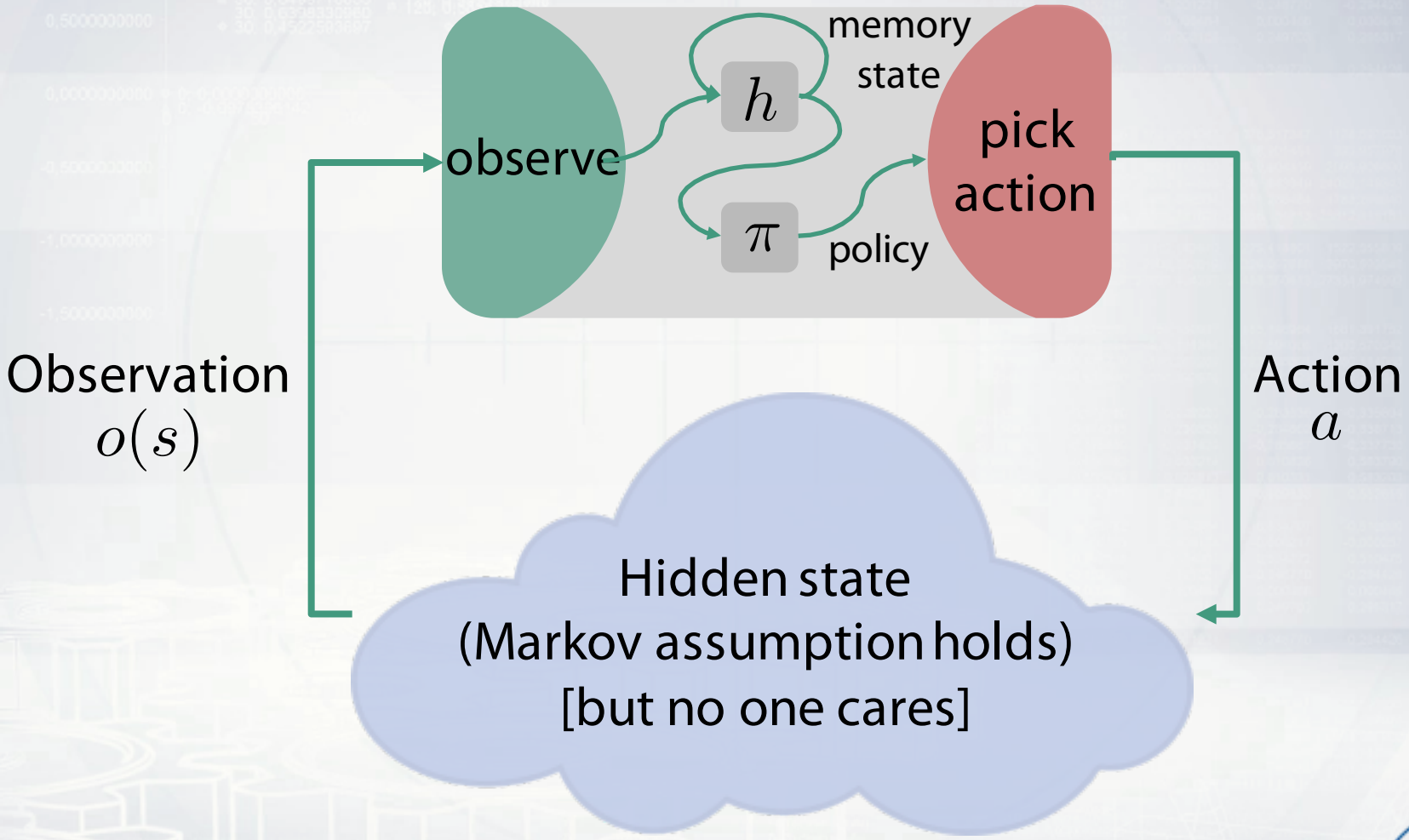
Partially Observable MDP



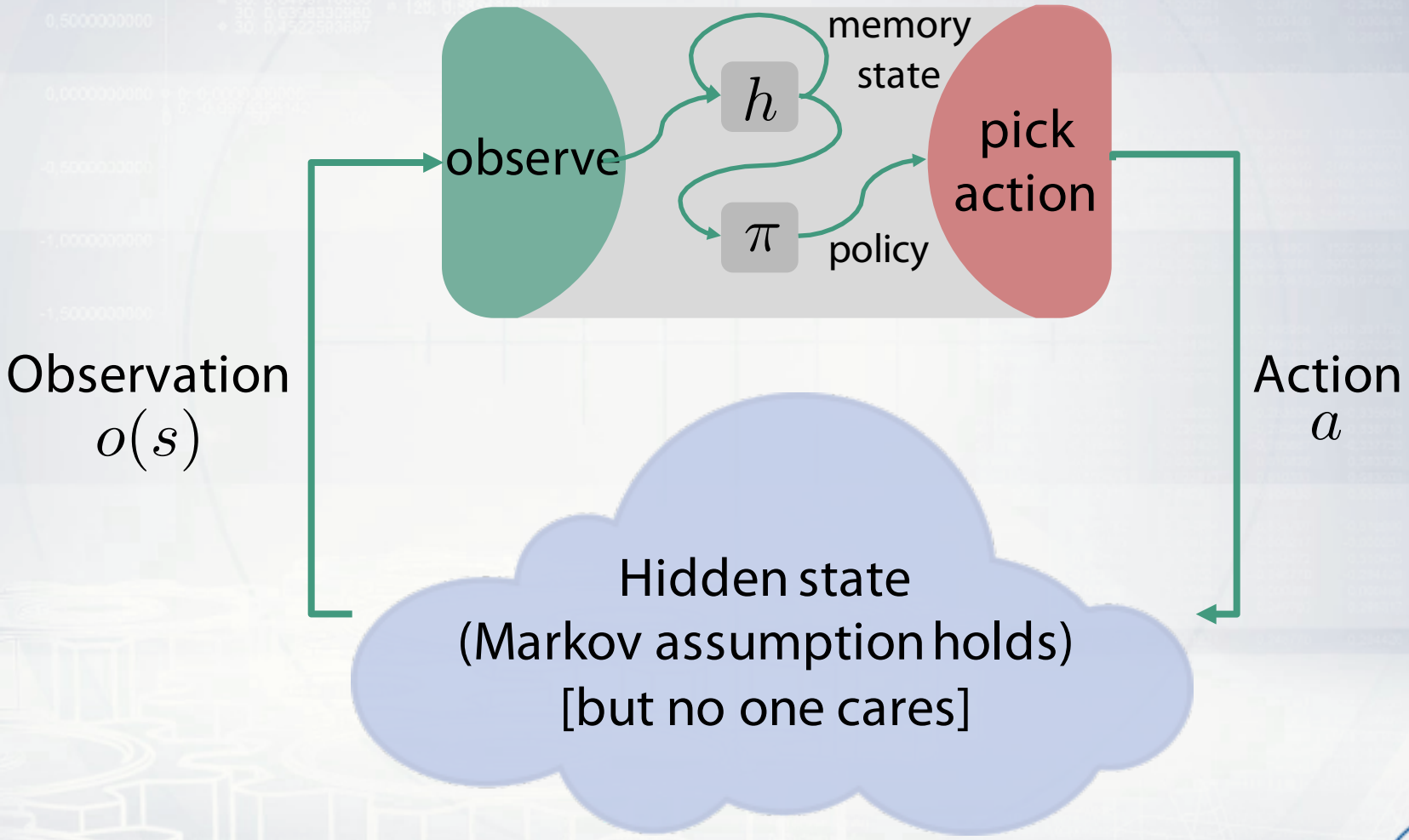
Partially Observable MDP

How do we train an agent in such process?

Partially Observable MDP



Partially Observable MDP



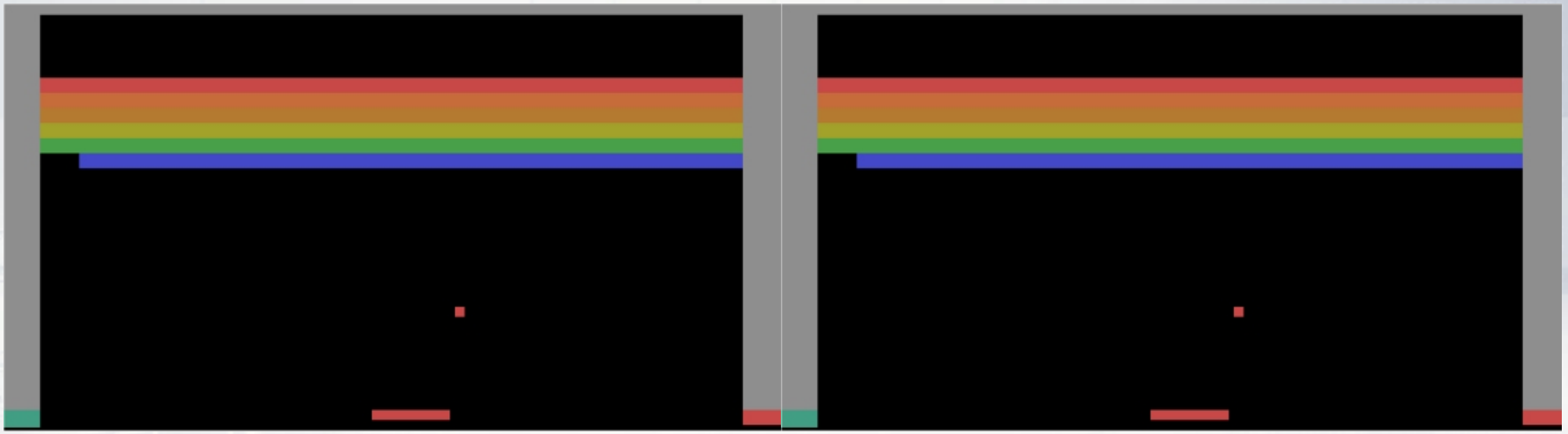
N-gram heuristic

Idea:

$$s_t \neq o(s_t)$$

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

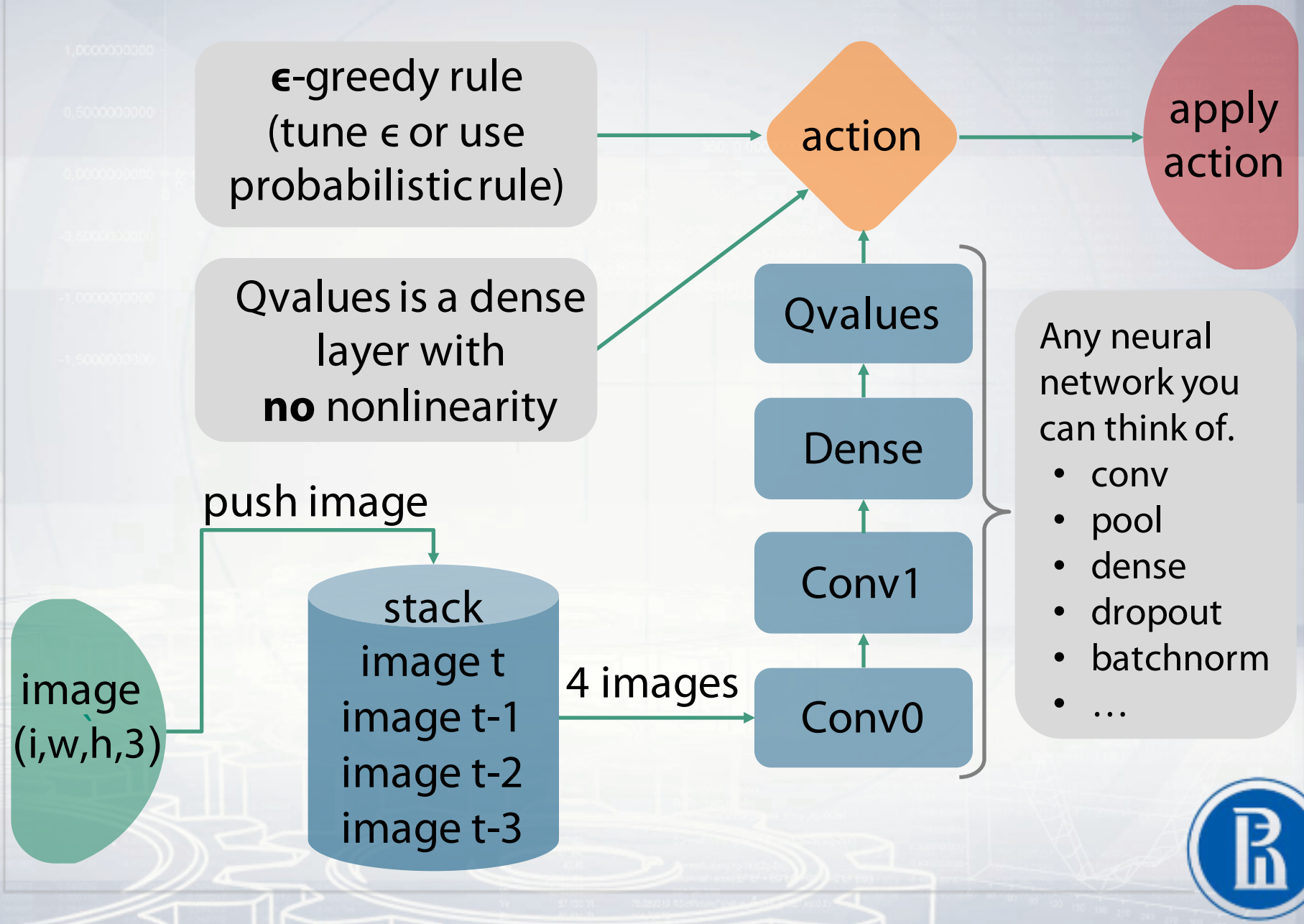
e.g. ball movement in breakout



One frame

Several frames

4-frame DQN

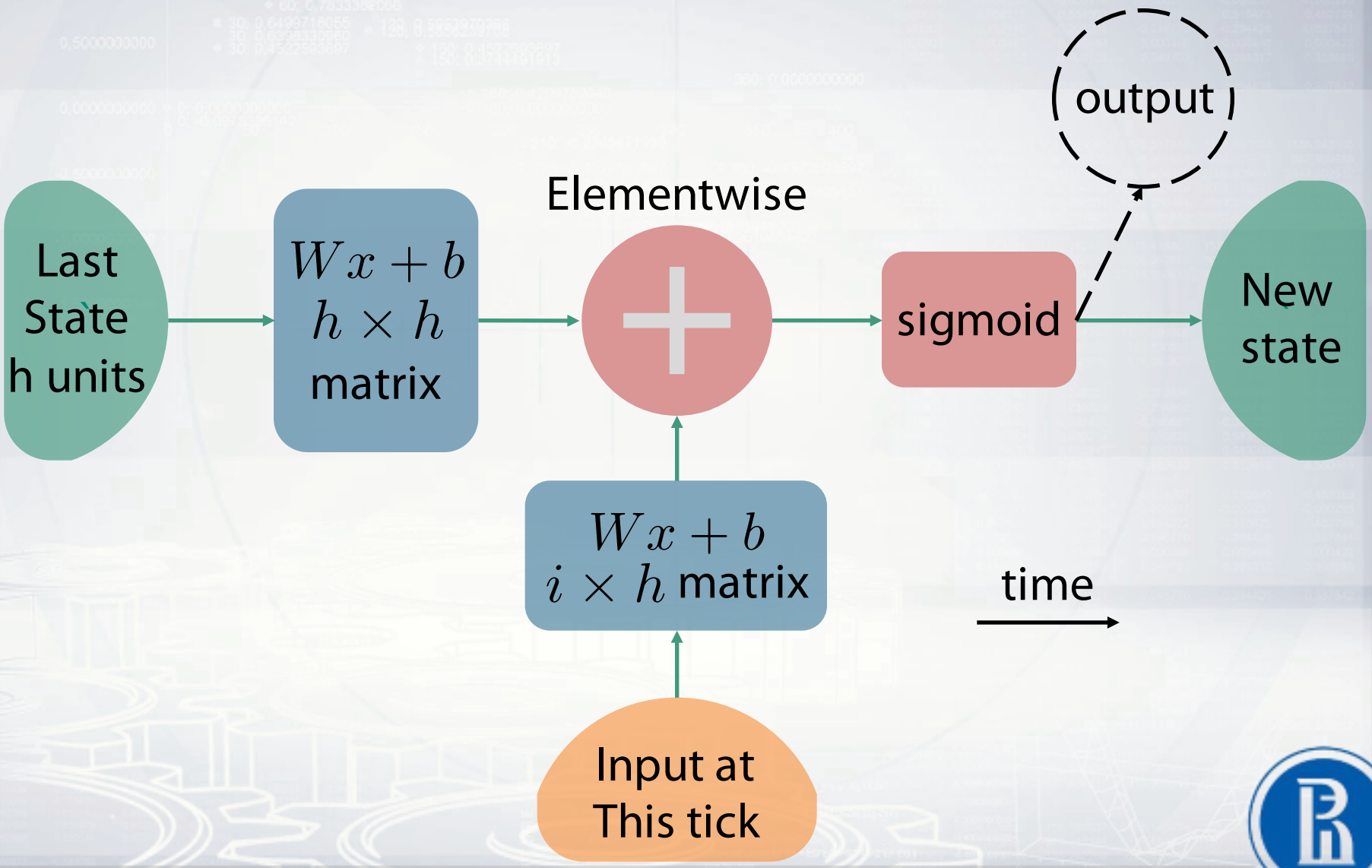

ϵ-greedy rule
(tune ϵ or use
probabilistic rule)
action
apply
action
Qvalues is a dense
layer with
no nonlinearity
Qvalues
Dense
Conv1
Conv0
Any neural
network you
can think of.
• conv
• pool
• dense
• dropout
• batchnorm
• …
push image
image
(i,w,h,3)
stack
image t
image t-1
image t-2
image t-3
4 images

Can we do better?

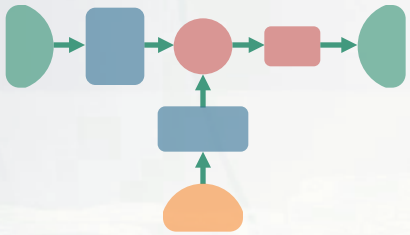
- There's a hidden process
- You only see some part of it
- You want hidden states

Reminds you of something?

Recurrent neural networks

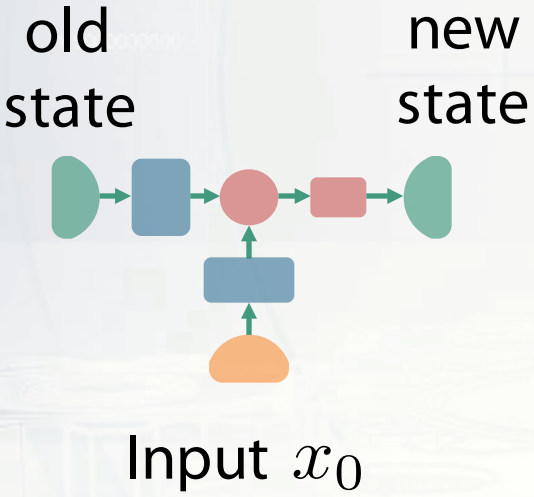


Recurrent neural networks

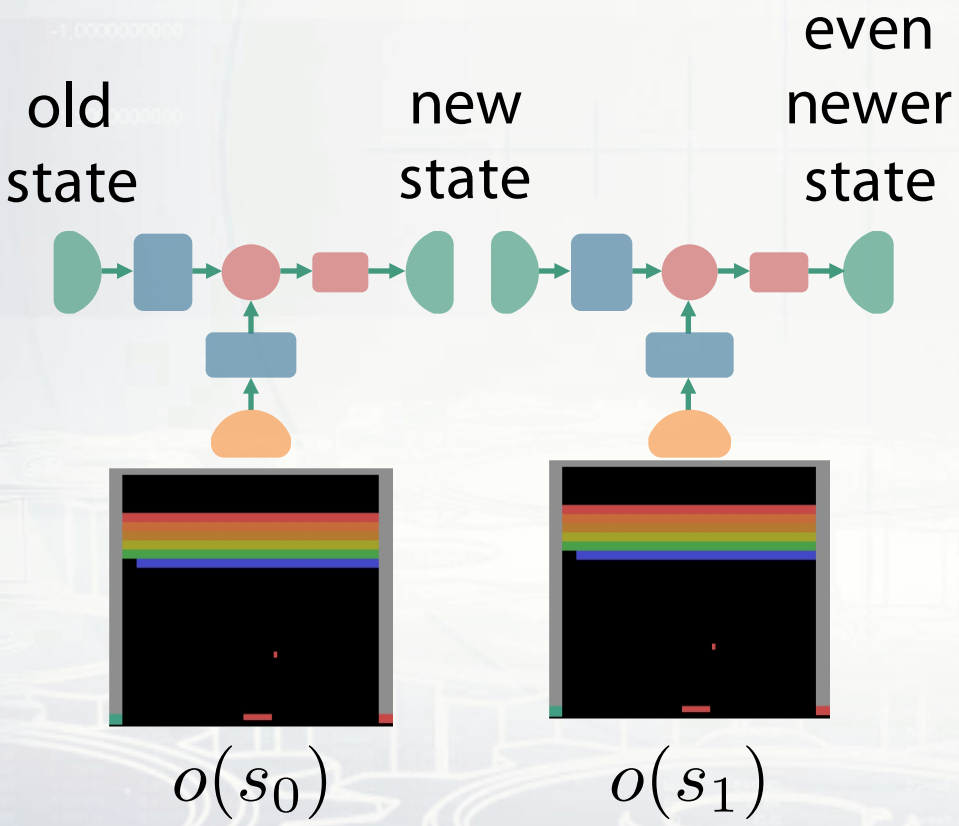


Zoom-out
of previous slide

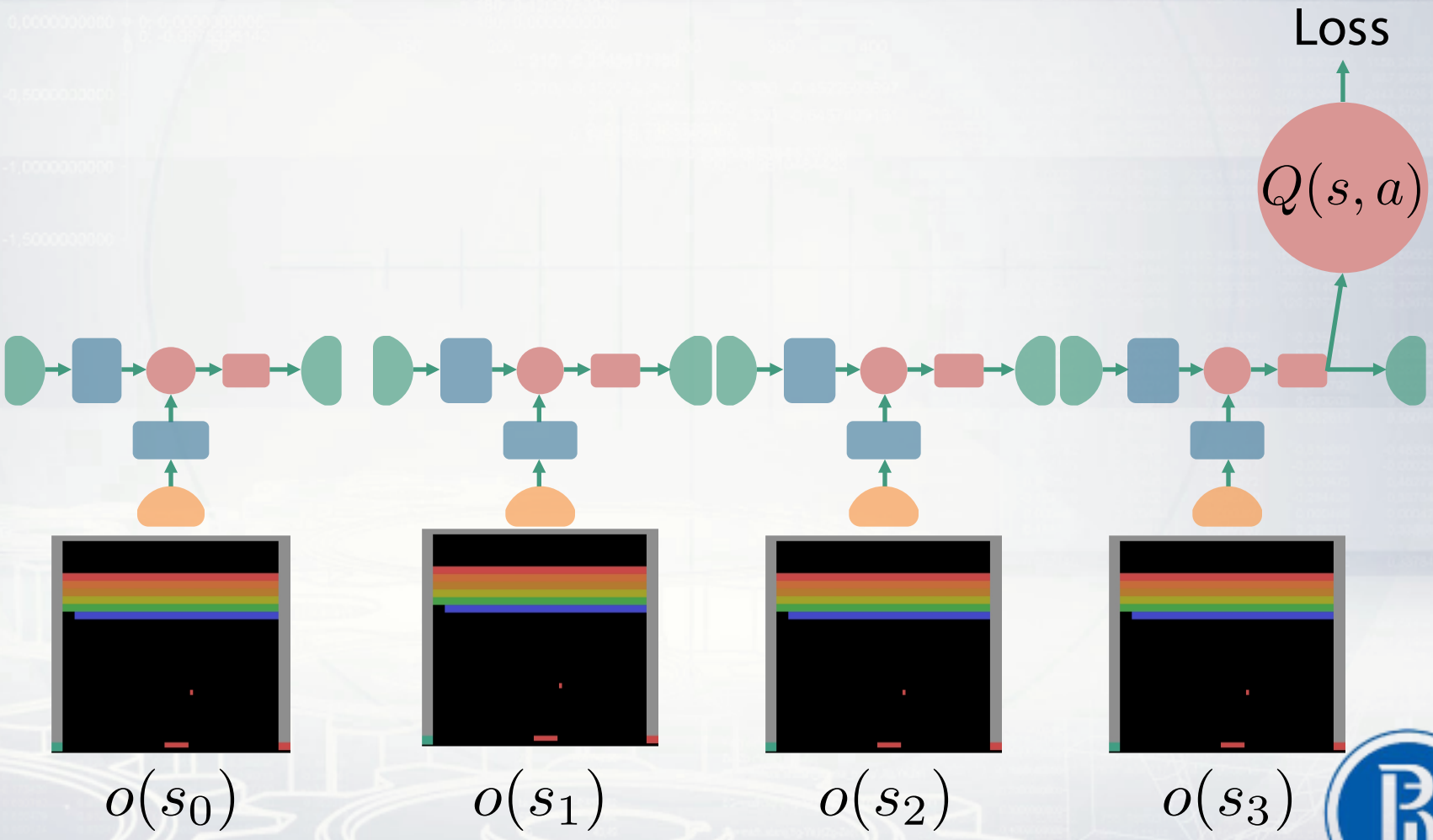
Recurrent neural networks



Recurrent neural networks



Recurrent neural networks

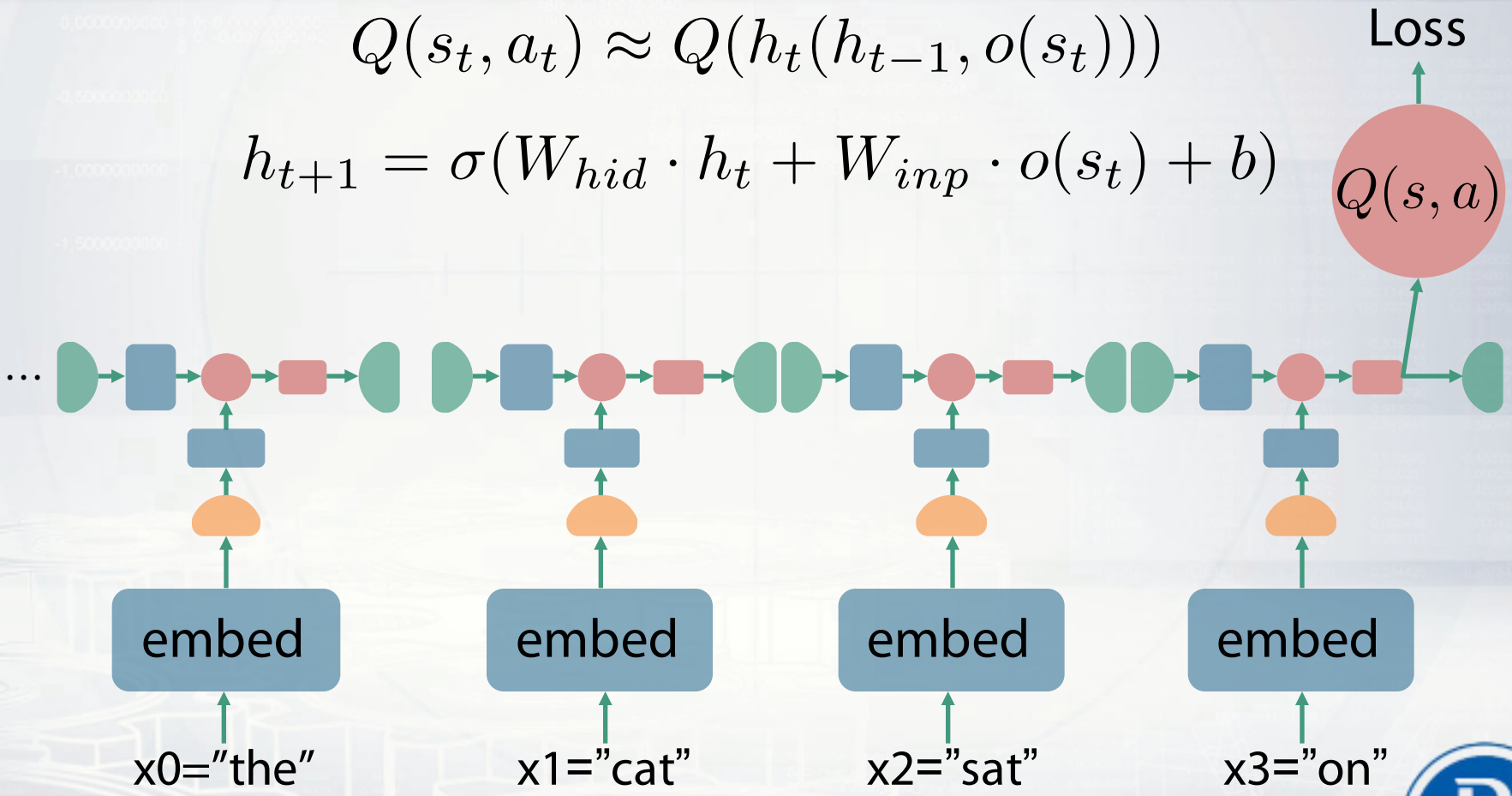


Recurrent neural networks

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

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

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



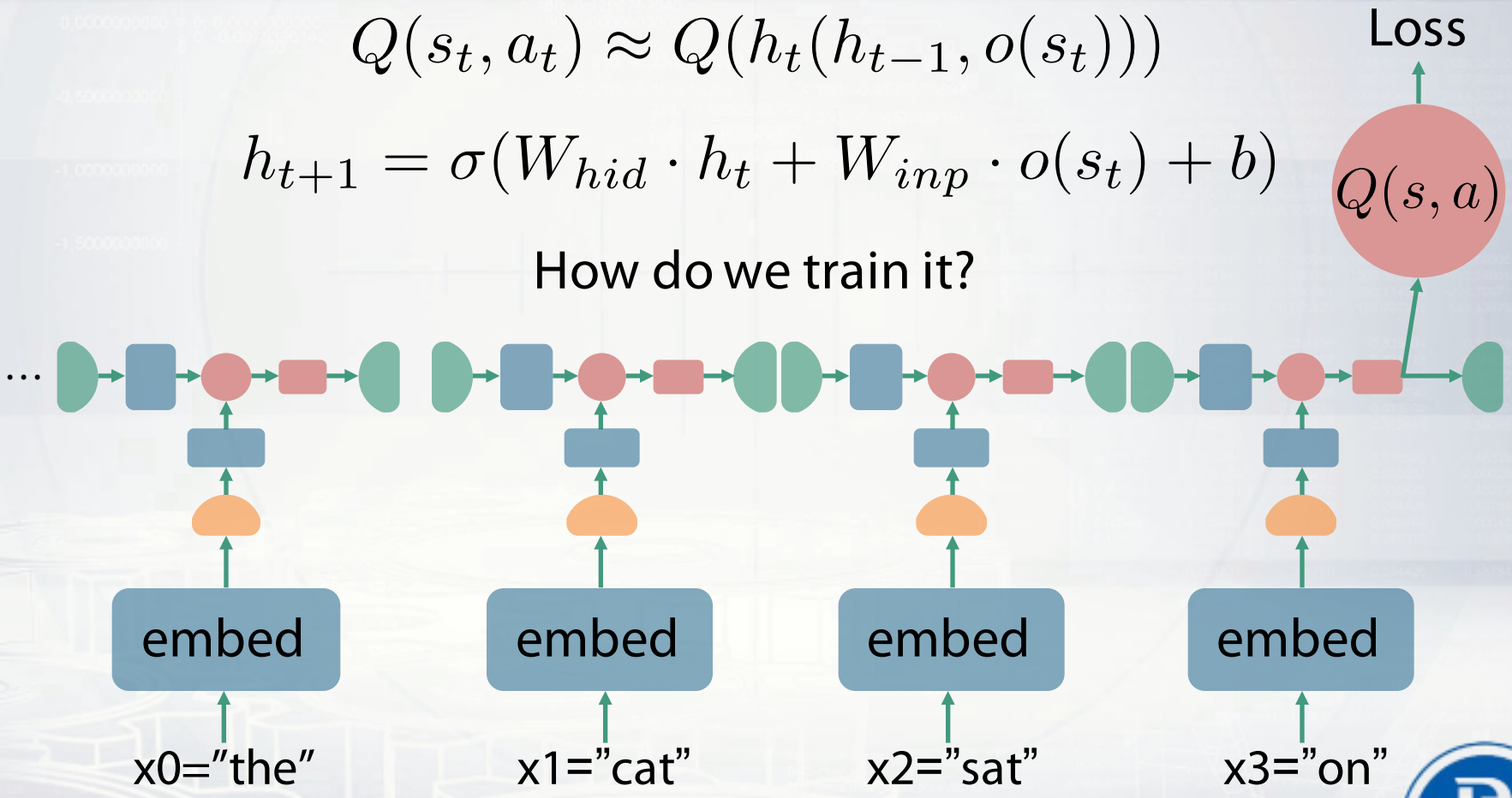
Recurrent neural networks

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

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

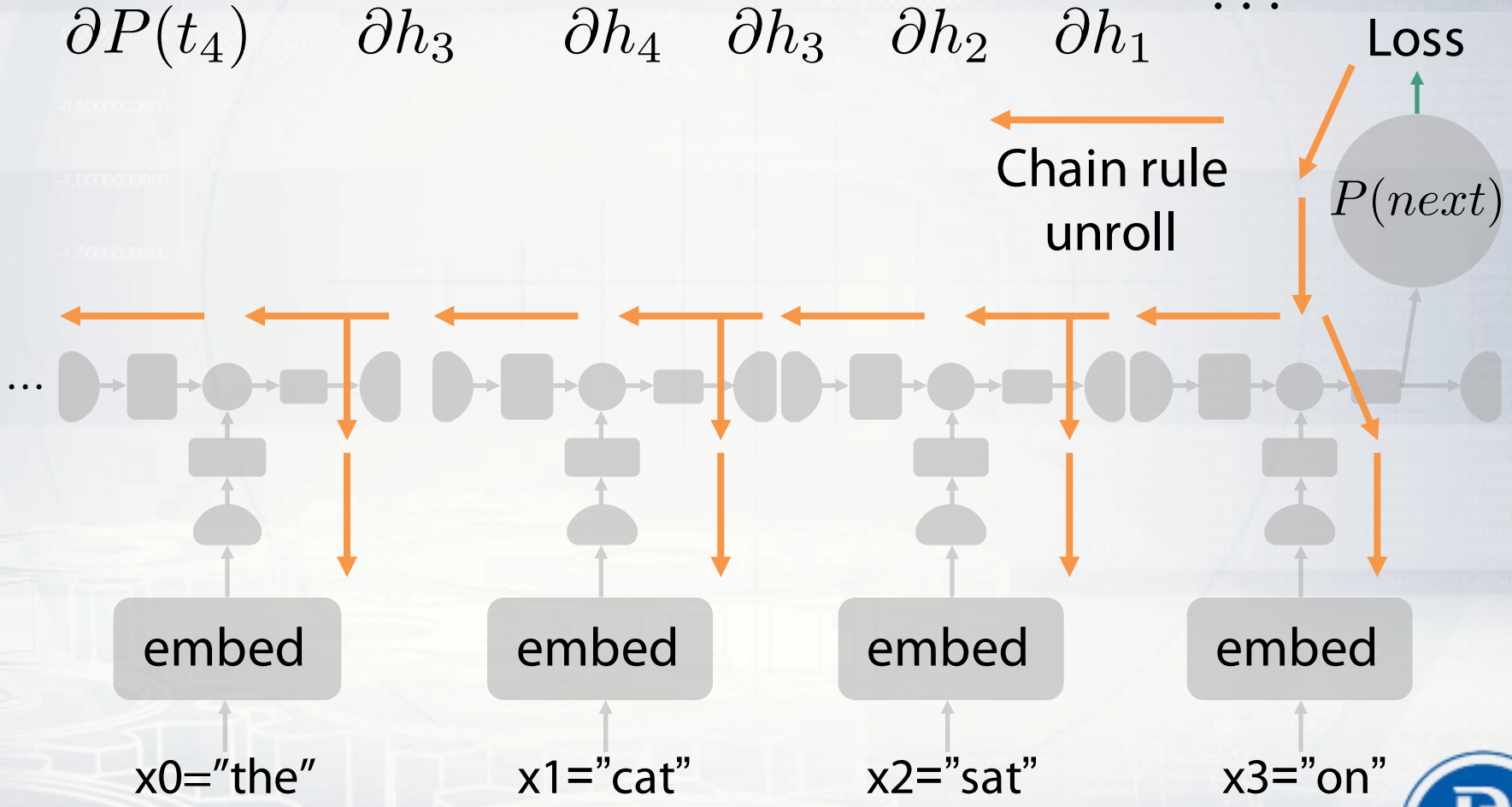
$$h_{t+1} = \sigma(W_{hid} \cdot h_t + W_{inp} \cdot o(s_t) + b)$$

How do we train it?



Backpropagation through time

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



DRQN

Idea:

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

<image here>

Results

Todo add table