

# Deep learning

Lecture 12

## tl;dr Reinforcement learning



Yandex  
Data Factory

LAMBDA



**British Hedgehog  
Preservation Society**

# Supervised learning

Given:

- objects and answers
- algorithm family
- loss function

$$(x, y)$$

$$a_{\theta}(x) \rightarrow y$$

$$L(y, a_{\theta}(x))$$

Find:

$$\theta' \leftarrow \operatorname{argmin}_{\theta} L(y, a_{\theta}(x))$$

# Supervised learning

Given:

- objects and answers      **customer** → **give loan?**       $(x, y)$
- algorithm family      **linear / tree / NN**       $a_{\theta}(x) \rightarrow y$
- loss function      **crossentropy**       $L(y, a_{\theta}(x))$

Find:

$$\theta' \leftarrow \operatorname{argmin}_{\theta} L(y, a_{\theta}(x))$$

# Reinforcement learning

Given:

- Bank with some budget
- Influx of clients
- A month to make it work or you're fired

Find:

$$\theta' = \operatorname{argmax}_{\theta} \textit{PROFIT} !!! (a_{\theta}(\textit{client}))$$

# Reinforcement learning

Given:

- Bank with some budget
- Influx of clients
- A month to make it work or you're fired

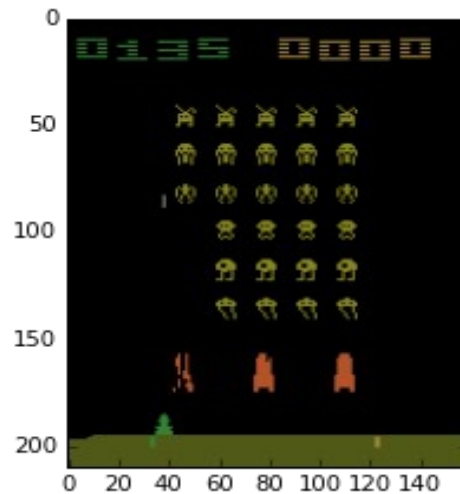
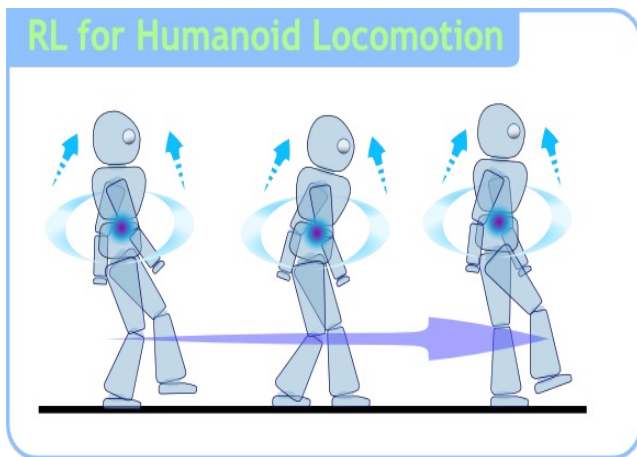
Find:

$$\theta' = \operatorname{argmax}_{\theta} \textit{PROFIT} !!! (a_{\theta}(\textit{client}))$$

**Ideas?**

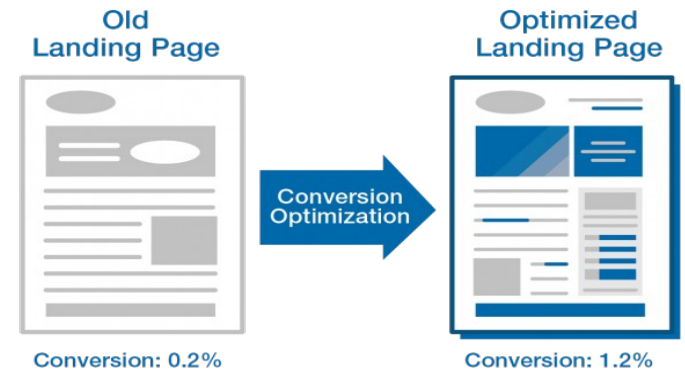
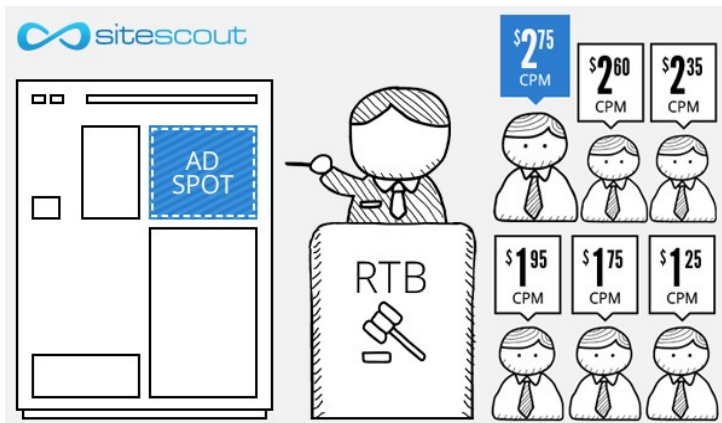
# Similar tasks

- Control robot to maximize speed
- Play game to maximize score
- Drive car to minimize human casualties



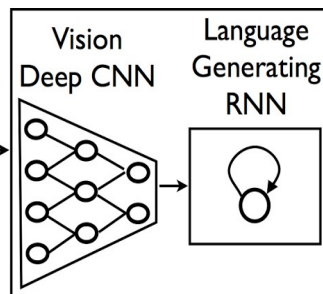
# Similar tasks

- Show banners to maximize clicks (or \$)
- Recommend films to maximize happiness
- Find pages that maximize relevance to queries



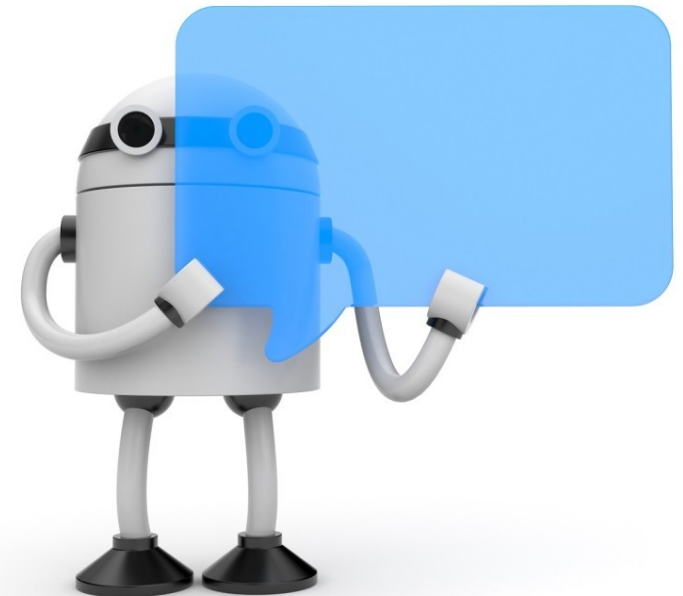
# Similar tasks

- Talk to human to satisfy his goals/constraints
- Translate sentence to maximize BLEU
- Generate captions for image with max CIDEr



**A group of people shopping at an outdoor market.**

**There are many vegetables at the fruit stand.**





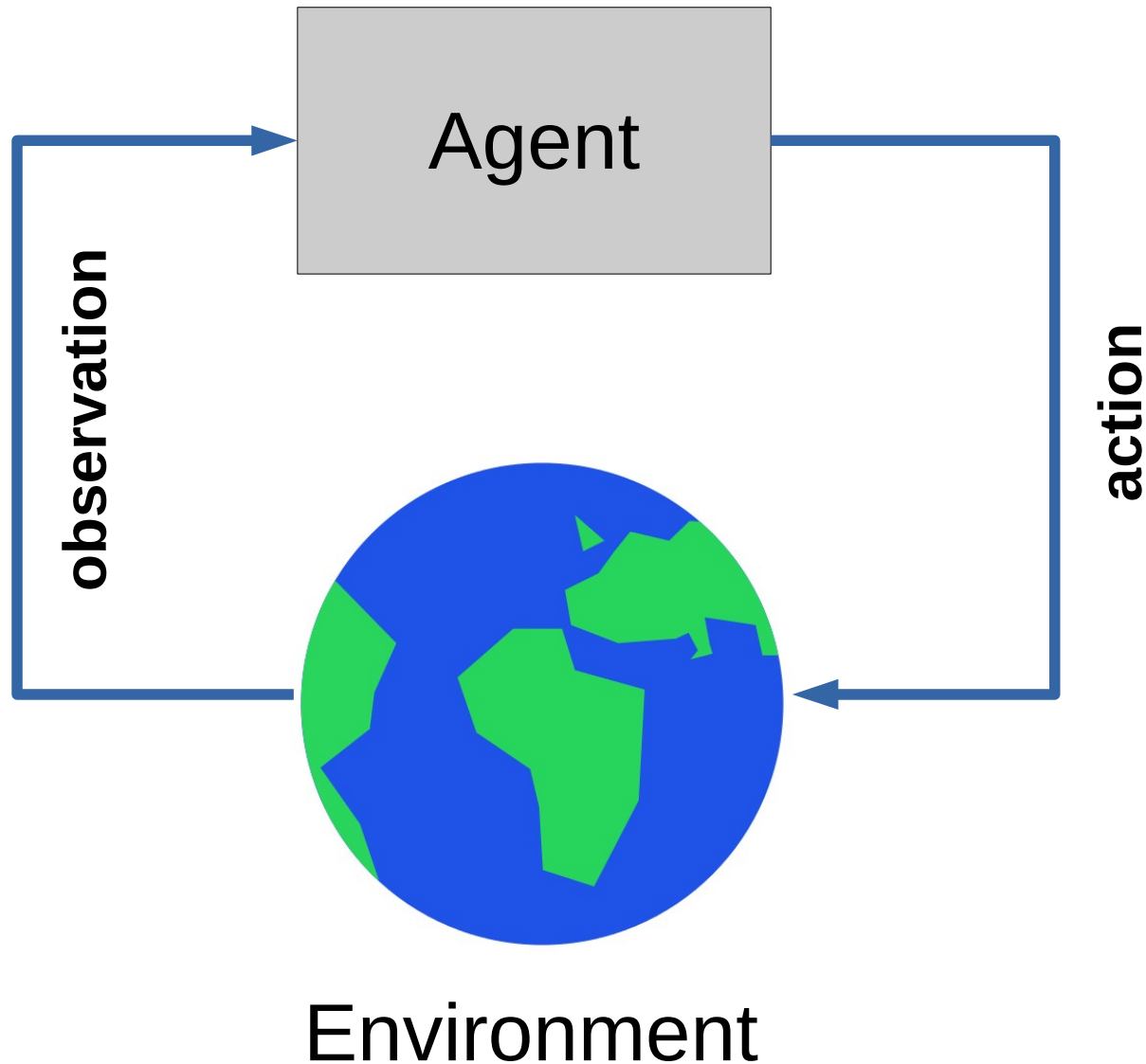
# Similar tasks

- Trade stocks
- Optimize datacenter usage
- ~~Bring you coffee~~

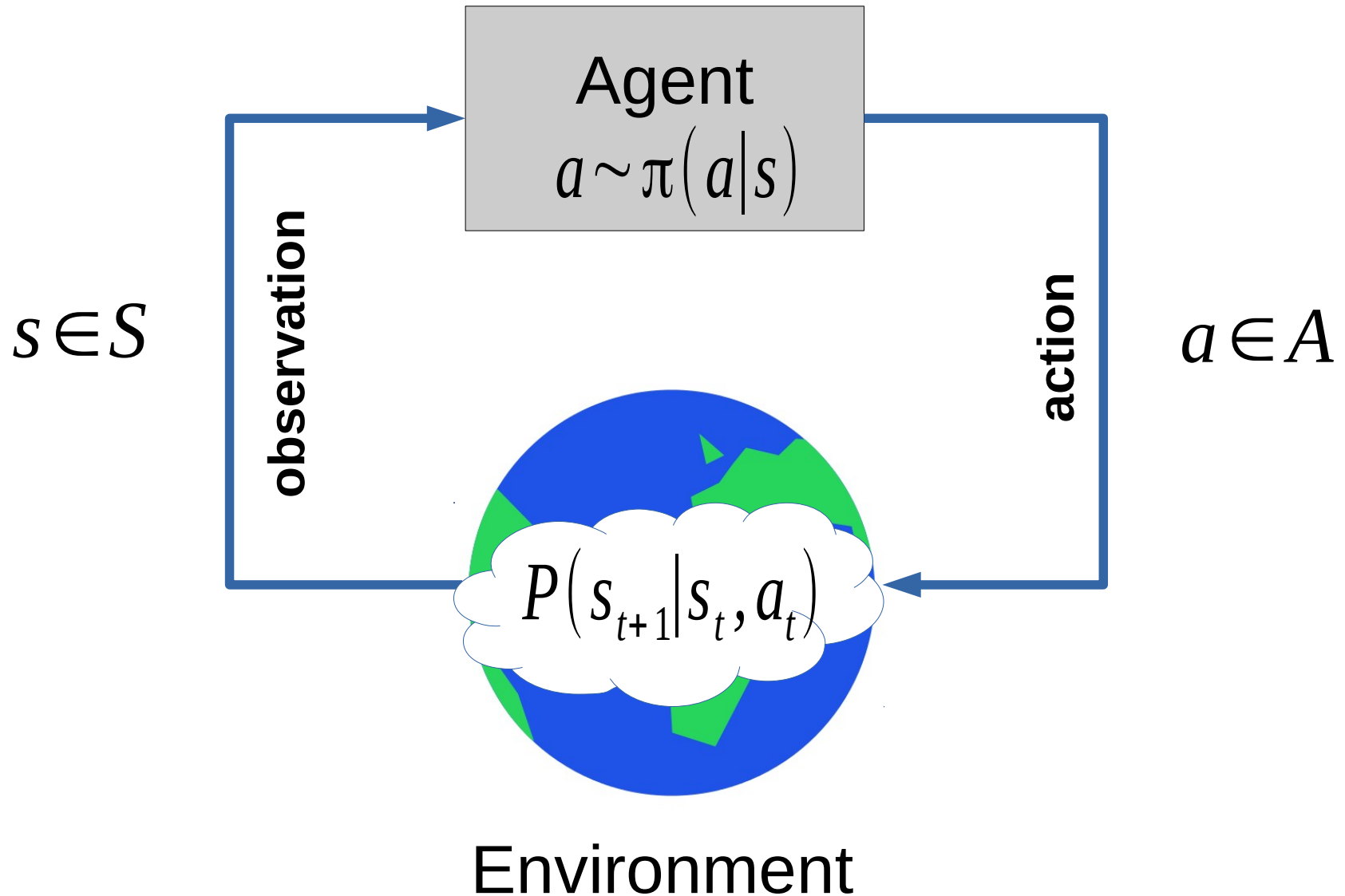


Image: I googled “whatsoever”

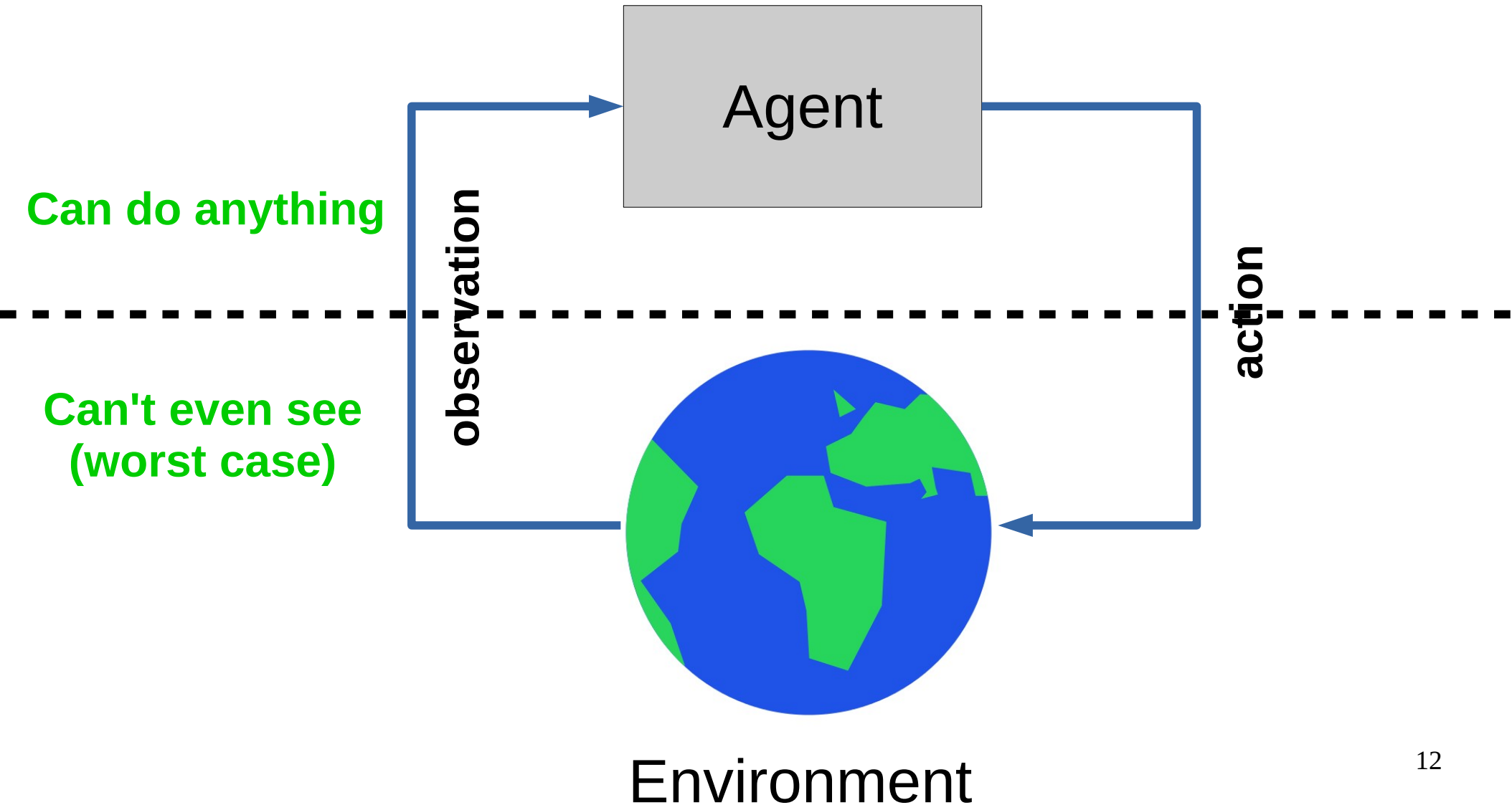
# Reinforcement learning formalism



# Reinforcement learning formalism



# Reinforcement learning formalism



# Objective:

- $R(s_0, a_0, s_1, a_1, s_2, a_2 \dots s_n, a_n)$  – reward for session
  - E.g. CIDEr metric of your captioning or total score
  - Total distance your robot walked in 1 minute
- Maximize reward

$$\pi' = \underset{\pi}{\operatorname{argmax}} \quad E_{s_0, a_0, s_1, a_1, \dots} R(s_0, a_0, s_1, a_1, \dots)$$

# Objective:

- $R(s_0, a_0, s_1, a_1, s_2, a_2 \dots s_n, a_n)$  – reward for session
  - E.g. CIDEr metric of your captioning or total score
  - Total distance your robot walked in 1 minute
- Maximize reward

$$\pi' = \underset{\pi}{\operatorname{argmax}} \quad E_{s_0, a_0, s_1, a_1, \dots} R(s_0, a_0, s_1, a_1, \dots)$$

**Where is pi used in the argmax-ed expression?**

# Objective:

- $R(s_0, a_0, s_1, a_1, s_2, a_2 \dots s_n, a_n)$  – reward for session
  - E.g. CIDEr metric of your captioning or total score
  - Total distance your robot walked in 1 minute

$$\pi' = \underset{\pi}{argmax} \quad E_{\substack{s_0 \sim P(s_0) \\ a_0 \sim \pi(a|s_0) \\ s_1 \sim P(s' \vee s_0, a_0) \\ a_1 \sim \dots}} R(s_0, a_0, s_1, a_1, \dots)$$

# Agent's policy

- Policy ~ whatever is used to choose actions
- Table of probabilities for each  $s$
- Linear model that learns  $p(a|s)$
- Guess what?



# Agent's policy

- Policy ~ whatever is used to choose actions
- Table of probabilities for each  $s$
- Linear model that learns  $p(a|s)$
- Neural network that learns  $p(a|s)$

# Objective

- For simplicity, consider single-step process

Agent sees one state, takes one action and it's over

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} R(s, a) = \int_s p(s) \int_a \pi_{\theta}(a|s) R(s, a) da ds$$

# Objective

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} R(s, a) = \int_s p(s) \int_a \pi_{\theta}(a|s) R(s, a) da ds$$

Diagram illustrating the components of the objective function  $J$ :

- Agent's policy**: Points to  $\pi_{\theta}(a|s)$ .
- state visitation frequency**: Points to  $p(s)$ .
- True action value**: Points to  $R(s, a)$ .

**Trivia:** how do we compute that?

# Objective

$$J = \underset{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}}{E} R(s, a) = \int_s p(s) \int_a \pi_{\theta}(a|s) R(s, a) da ds$$

$$J \approx \frac{1}{N} \sum_{i=0}^N R(s_i, a_i)$$

True action value



sample N sessions



# Objective

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} R(s, a) = \int_s p(s) \int_a \pi_{\theta}(a|s) R(s, a) da ds$$

$$J \approx \frac{1}{N} \sum_{i=0}^N R(s_i, a_i)$$

True action value



sample N sessions



**Can we optimize policy now?**

# Objective

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} R(s, a) = \int_s p(s) \int_a \pi_{\theta}(a|s) R(s, a) da ds$$

parameters “sit” here



$$J \approx \frac{1}{N} \sum_{i=0}^N R(s_i, a_i)$$

**We don't know how to compute  $dJ/d\theta$**

# Optimization

## Finite differences

- Change policy a little, evaluate

$$\nabla J \approx \frac{J_{\theta+\epsilon} - J_{\theta}}{\epsilon}$$

# Optimization

## Finite differences

- Change policy a little, evaluate

$$\nabla J \approx \frac{J_{\theta+\epsilon} - J_{\theta}}{\epsilon}$$

**Trivia:** any problems with this?



# Optimization

## Finite differences

- Change policy a little, evaluate

$$\nabla J \approx \frac{J_{\theta+\epsilon} - J_{\theta}}{\epsilon}$$

**VERY noisy**

**especially if both J are sampled**

# Objective

$$J = E_{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}} R(s, a) = \int_s p(s) \int_a \pi_\theta(a|s) R(s, a) da ds$$

Wish list:

- Analytical gradient
- Easy/stable approximations

# Logderivative trick

Simple math

$$\nabla \log \pi(z) = ? ? ?$$

(try chain rule)

# Logderivative trick

Simple math

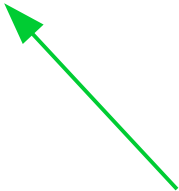
$$\nabla \log \pi(z) = \frac{1}{\pi(z)} \cdot \nabla \pi(z)$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$

# Policy gradient

Analytical inference

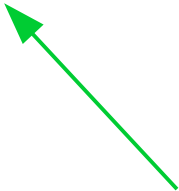
$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) R(s, a) da ds$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$


# Policy gradient

## Analytical inference

$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) R(s, a) da ds$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$


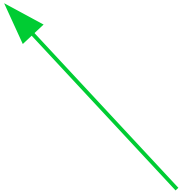
$$\nabla J = \int_s p(s) \int_a \pi_\theta(a|s) \nabla \log \pi_\theta(a|s) R(s, a) da ds$$

**Trivia:** anything curious about that formula?

# Policy gradient

## Analytical inference

$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) R(s, a) da ds$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$


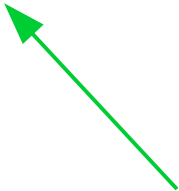
$$\nabla J = \int_s p(s) \int_a \pi_\theta(a|s) \nabla \log \pi_\theta(a|s) R(s, a) da ds$$

that's expectation :)

# Policy gradient

Analytical inference

$$\nabla J = \int_s p(s) \int_a \nabla \pi_\theta(a|s) R(s, a) da ds$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$


$$\nabla J = E_{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}} \nabla \log \pi_\theta(a|s) \cdot R(s, a)$$



# Policy gradient (REINFORCE)

- Policy gradient

$$\nabla J = E_{\substack{s \sim p(s) \\ a \sim \pi_{\theta}(s|a)}} \nabla \log \pi_{\theta}(a|s) \cdot R(s, a)$$

- Approximate with sampling

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in z_i} \nabla \log \pi_{\theta}(a|s) \cdot R(s, a)$$

# REINFORCE algorithm

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions  $\mathbf{z}$  under current  $\pi_\theta(a|s)$
  - Evaluate policy gradient

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_\theta(a|s) \cdot R(s, a)$$

- Ascend  $\theta_{i+1} \leftarrow \theta_i + \alpha \cdot \nabla J$

# REINFORCE algorithm

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions  $\mathbf{z}$  under current  $\pi_\theta(a|s)$   
actions under current policy  
= on-policy
  - Evaluate policy gradient

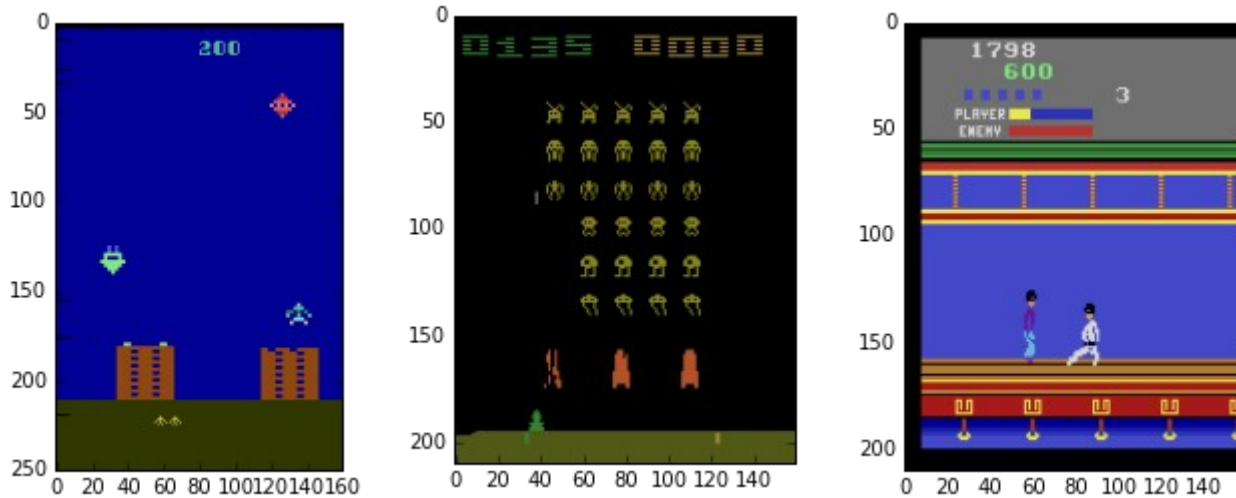
$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_\theta(a|s) \cdot R(s, a)$$

- Ascend  $\theta_{i+1} \leftarrow \theta_i + \alpha \cdot \nabla J$

# Problems so far

- We assume that process is finite
- Okay, you just did 1000 steps and got  $R=10$   
What of 1000 actions caused that reward?
- Can we define immediate rewards for
  - E.g. chess?
  - Atari game?

# Reality check: videogames



- Trivia: how to measure reward before game ends?

# Discounted reward MDP



Objective:  
Total action value

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots + \gamma^n \cdot r_{t+n}$$

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \quad \gamma \in (0,1) \text{ const}$$

$\gamma \sim$  patience

Cake tomorrow is  $\gamma$  as good as now

Reinforcement learning:

- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow \max$$

# Discounted reward MDP



Objective:  
Total action value

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots + \gamma^n \cdot r_{t+n}$$

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \quad \gamma \in (0,1) \text{ const}$$

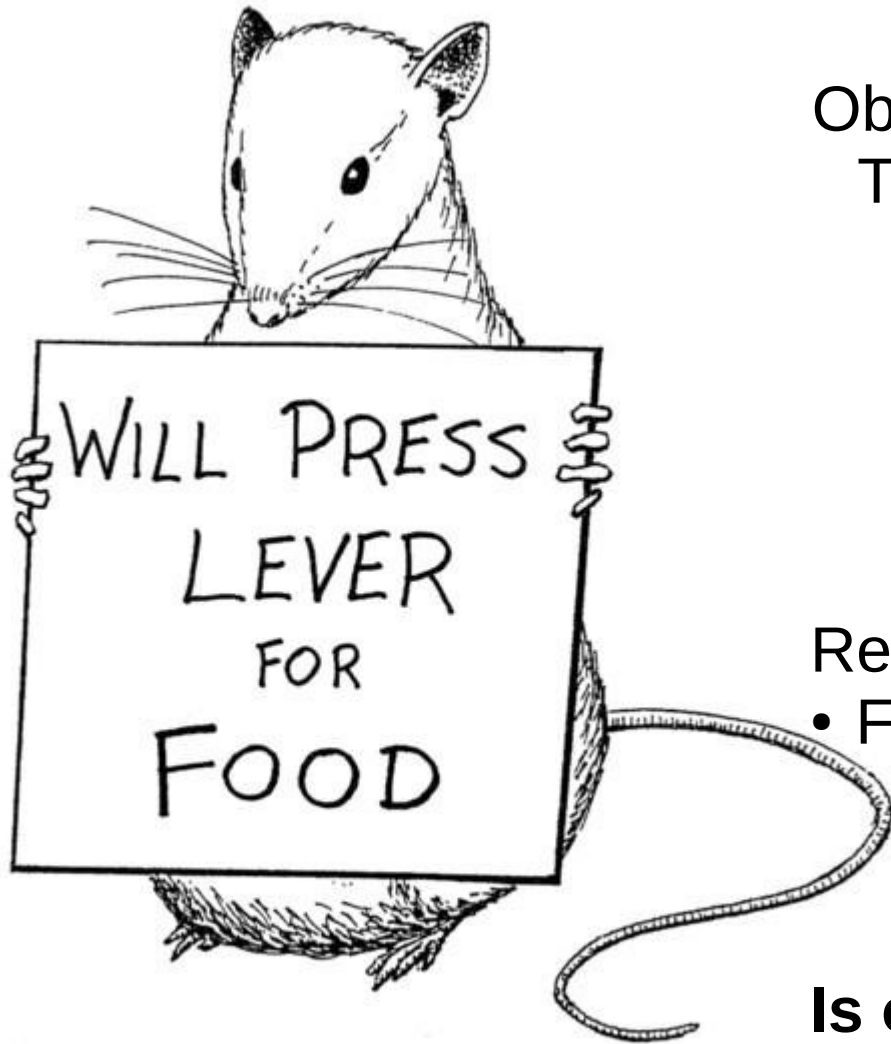
**Trivia: which  $\gamma$  corresponds to “only current reward matters”?**

Reinforcement learning:

- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow \max$$

# Discounted reward MDP



Objective:  
Total reward

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots + \gamma^n \cdot r_{t+n}$$

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \quad \gamma \in (0,1) \text{ const}$$

Reinforcement learning:

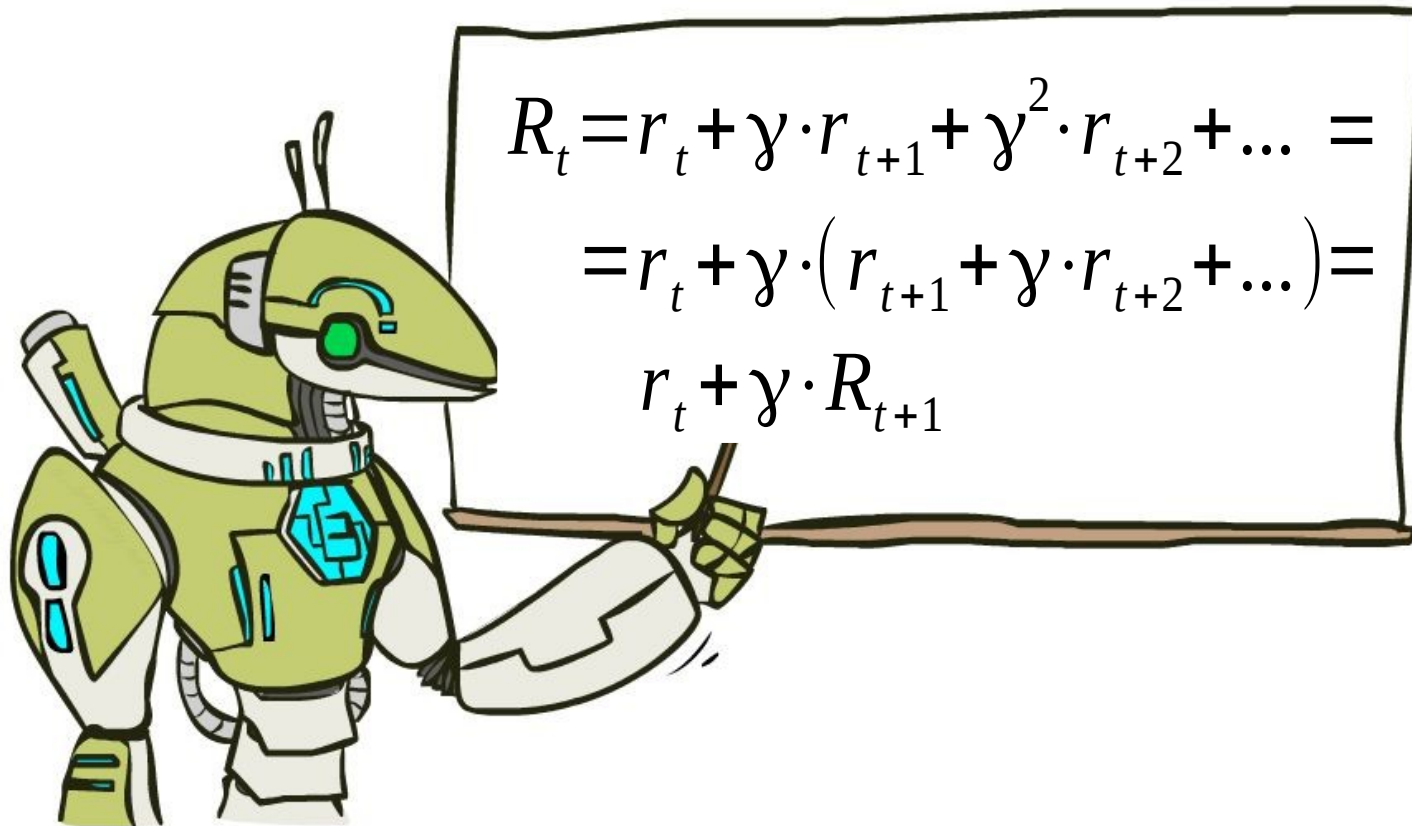
- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow \max$$

**Is optimal policy same as it would be in  $R(s_0 a_0 \dots s_n)$  (if we add-up all  $r_t$ )?**<sup>40</sup>



# Optimal policy



We rewrite R with sheer power of math!

# Value iteration (Temporal Difference)

Idea:

- For each state, obtain  $V(\text{state})$

$$V(s) = E_{a \sim \pi(a|s)} R(s, a)$$

**Definition**  $V(s)$  – expected total reward  $R$  that can be obtained starting from state  $s$  under **current** policy

Note: some algorithms define  $V$  differently,  
e.g. using not current but optimal policy

# Value iteration (Temporal Difference)

Idea:

- For each state, obtain  $V(\text{state})$

$$V(s) = E_{a \sim \pi(a|s)} R(s, a) = E_{a \sim \pi(a|s)} [r(s, a) + \gamma \cdot V(s'(s, a))]$$

**Definition**  $V(s)$  – expected total reward  $R$  that can be obtained starting from state  $s$  under **current** policy

Note: some algorithms define  $V$  differently,  
e.g. using not current but optimal policy

# REINFORCE baseline

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions  $\mathbf{z}$  under current  $\pi_\theta(a|s)$
  - Evaluate policy gradient

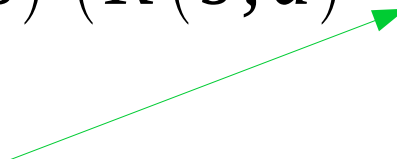
$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_\theta(a|s) \cdot R(s, a)$$

$$R(s, a) = V(s) + A(s, a)$$

Actions influence  $A(s, a)$  only, so  $V(s)$  is irrelevant

# REINFORCE baseline

- Initialize NN weights  $\theta_0 \leftarrow \text{random}$
- Loop:
  - Sample N sessions  $\mathbf{z}$  under current  $\pi_\theta(a|s)$
  - Evaluate policy gradient

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in \mathbf{z}_i} \nabla \log \pi_\theta(a|s) \cdot (R(s, a) - V(s))$$


Anything that doesn't depend on action

# Actor-critic

- Learn both  $V(s)$  and  $\pi_{\theta}(a|s)$
- Hope for best of both worlds :)



# Advantage actor-critic

Idea: learn both  $\pi_{\theta}(a|s)$  and  $V_{\theta}(s)$

Use  $V_{\theta}(s)$  to learn  $\pi_{\theta}(a|s)$  faster!

**Non-trivia:** how can we estimate  $\mathbf{A(s,a)}$   
from  $(s,a,r,s')$  and  $V(s)$  function?

# Advantage actor-critic

Idea: learn both  $\pi_{\theta}(a|s)$  and  $V_{\theta}(s)$

Use  $V_{\theta}(s)$  to learn  $\pi_{\theta}(a|s)$  faster!

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

$$R(s, a) = r + \gamma \cdot V(s')$$

$$A(s, a) = r + \gamma \cdot V(s') - V(s)$$



# Advantage actor-critic

Idea: learn both  $\pi_{\theta}(a|s)$  and  $V_{\theta}(s)$

Use  $V_{\theta}(s)$  to learn  $\pi_{\theta}(a|s)$  faster!

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

$$R(s, a) = r + \gamma \cdot V(s')$$

$$A(s, a) = r + \gamma \cdot V(s') - V(s)$$

# Advantage actor-critic

Idea: learn both  $\pi_{\theta}(a|s)$  and  $V_{\theta}(s)$

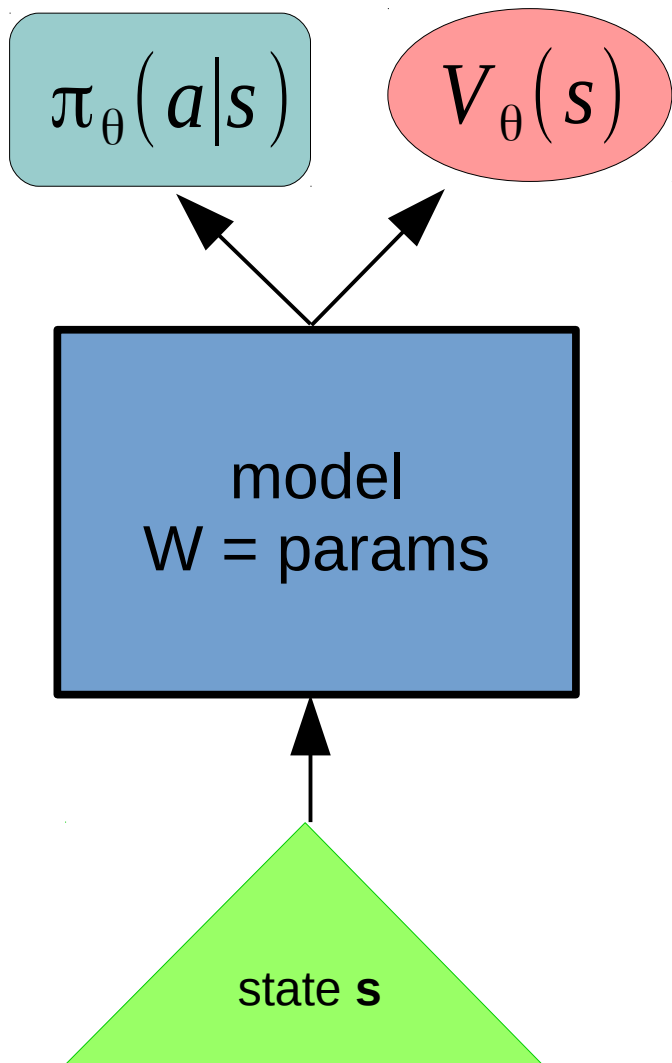
Use  $V_{\theta}(s)$  to learn  $\pi_{\theta}(a|s)$  faster!

$$A(s, a) = r + \gamma \cdot V(s') - V(s)$$

$$\nabla J_{actor} \approx \frac{1}{N} \sum_{i=0}^N \sum_{s, a \in z_i} \nabla \log \pi_{\theta}(a|s) \cdot \underbrace{A(s, a)}_{\text{consider const}}$$

**Trivia:** how do we train  $V$  then?

# Advantage actor-critic



Improve policy:

$$\nabla J_{actor} \approx \frac{1}{N} \sum_{i=0}^N \sum_{s,a \in z_i} \nabla \log \pi_{\theta}(a|s) \cdot A(s, a)$$

Improve value:

$$L_{critic} \approx \frac{1}{N} \sum_{i=0}^N \sum_{s,a \in z_i} (V_{\theta}(s) - [r + \gamma \cdot V(s')])^2$$

# Continuous action spaces

What if there's continuously many actions?

- Robot control: control motor voltage
- Trading: assign money to equity

How does the algorithm change?

# Continuous action spaces

What if there's continuously many actions?

- Robot control: control motor voltage
- Trading: assign money to equity

How does the algorithm change?

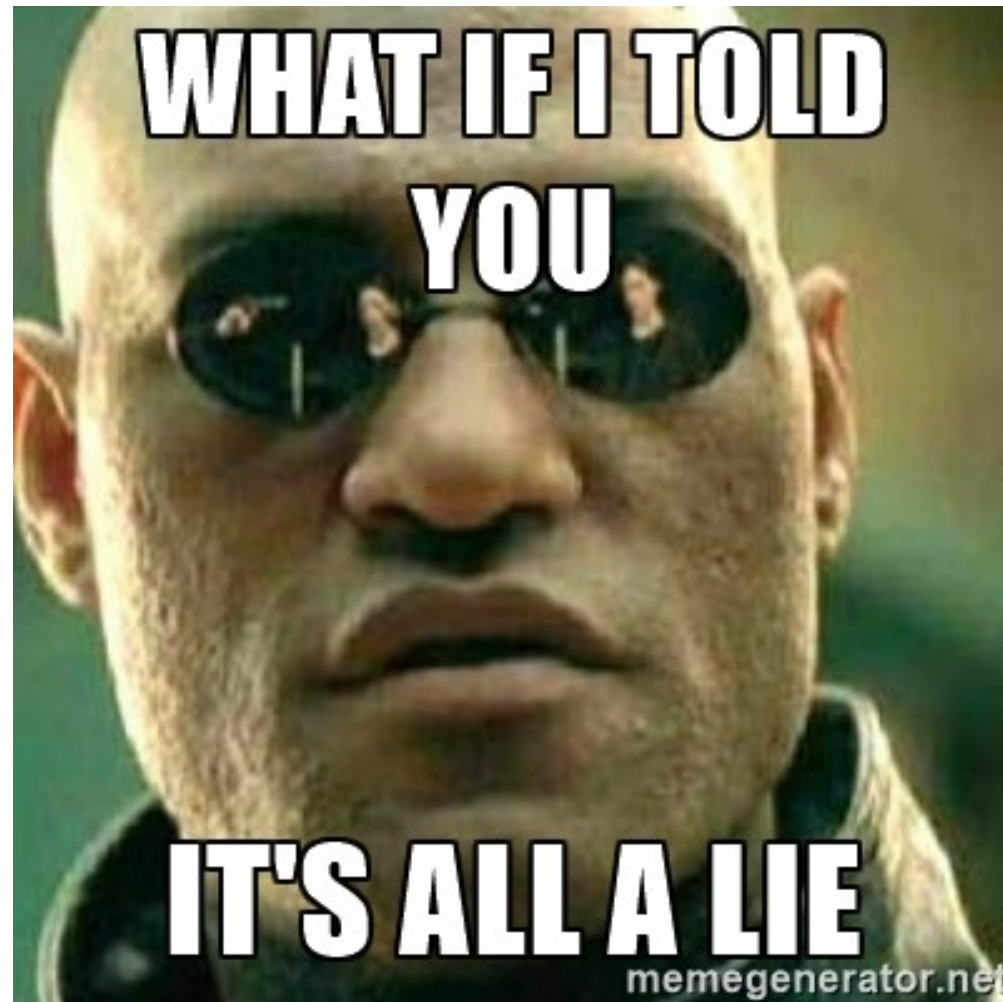
it doesn't :)  
Just plug in a different formula for  
 $\pi(a|s)$ , e.g. normal distribution

# Duct tape zone

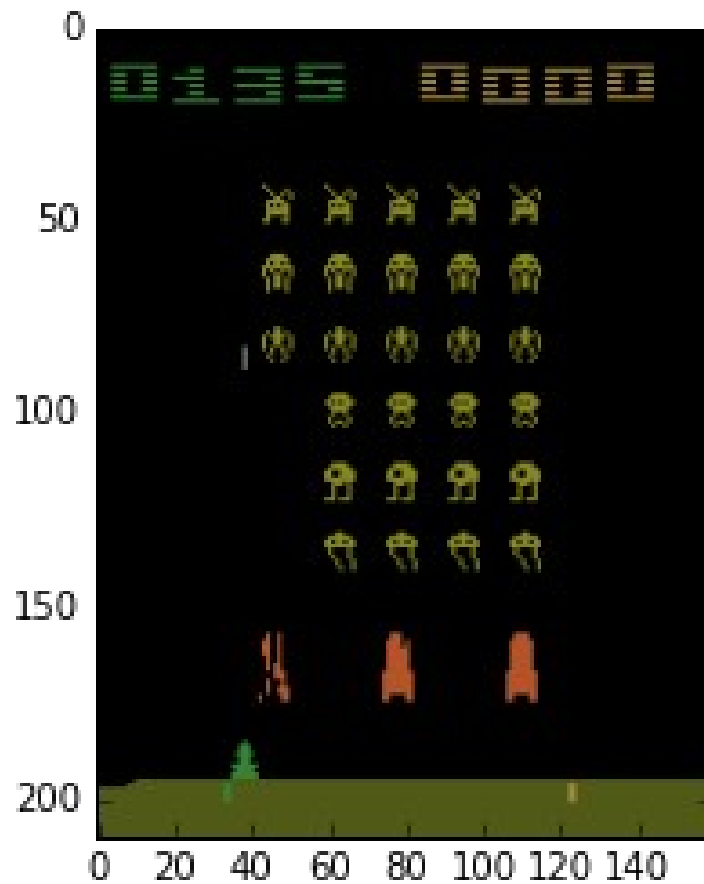
- $V(s)$  errors less important than in Q-learning
  - actor still learns even if critic is random, just slower
- Regularize with entropy
  - to prevent premature convergence
- Learn on parallel sessions
  - Or super-small experience replay
- Use logsoftmax for numerical stability



Let's code!







How bad it is if agent spends  
next 1000 ticks under the left rock?  
(while training)

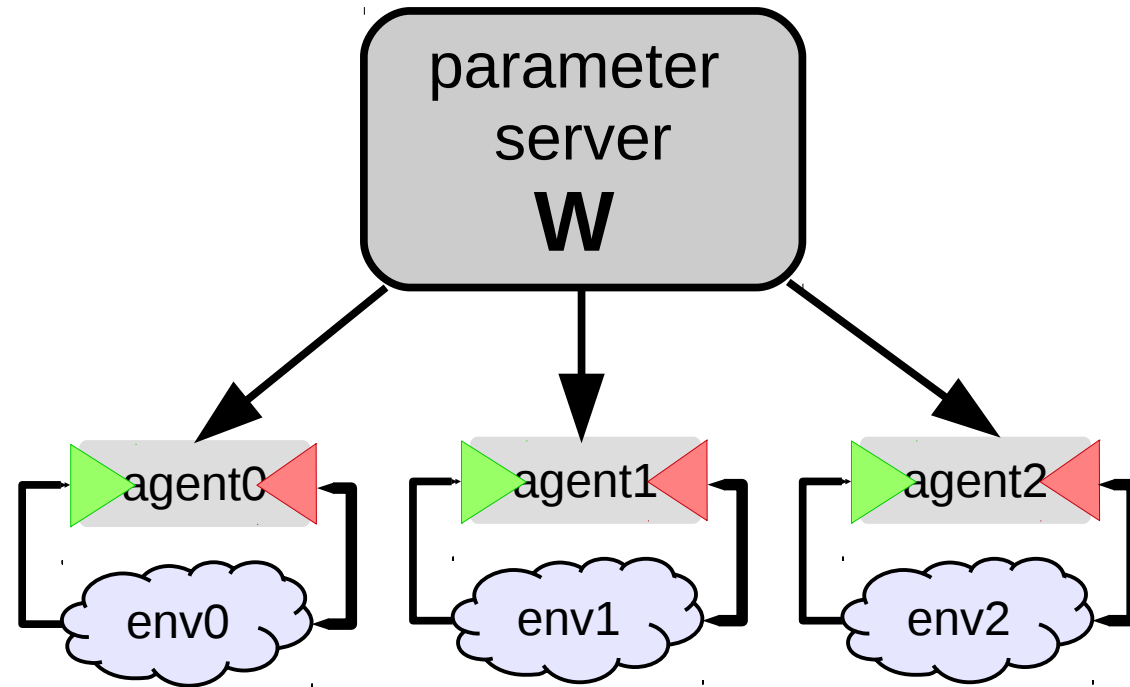
# Problem

- Training samples are **not** “i.i.d”,
- Model forgets parts of environment it hasn't visited for some time
- Drops on learning curve
- **Any ideas?**



# Multiple agent trick

**Idea:** Throw in several agents with shared  $\mathbf{W}$ .

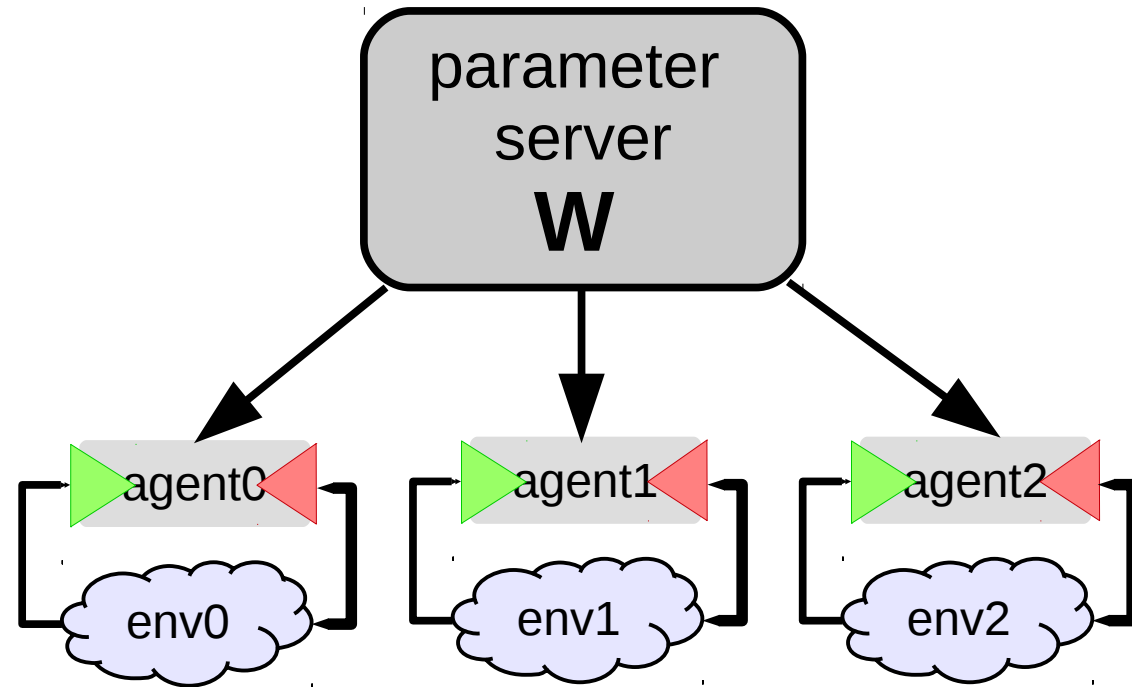


# Multiple agent trick

**Idea:** Throw in several agents with shared  $W$ .

- Chances are, they will be exploring different parts of the environment,
- More stable training,
- Requires a lot of interaction

**Trivia:** your agent is a real robot car. Any problems?



# Final problem



**Left or right?**

# Problem:

Most practical cases are partially observable:

Agent observation does not hold all information about process state  
(e.g. human field of view).

**Any ideas?**

# Problem:

Most practical cases are partially observable:

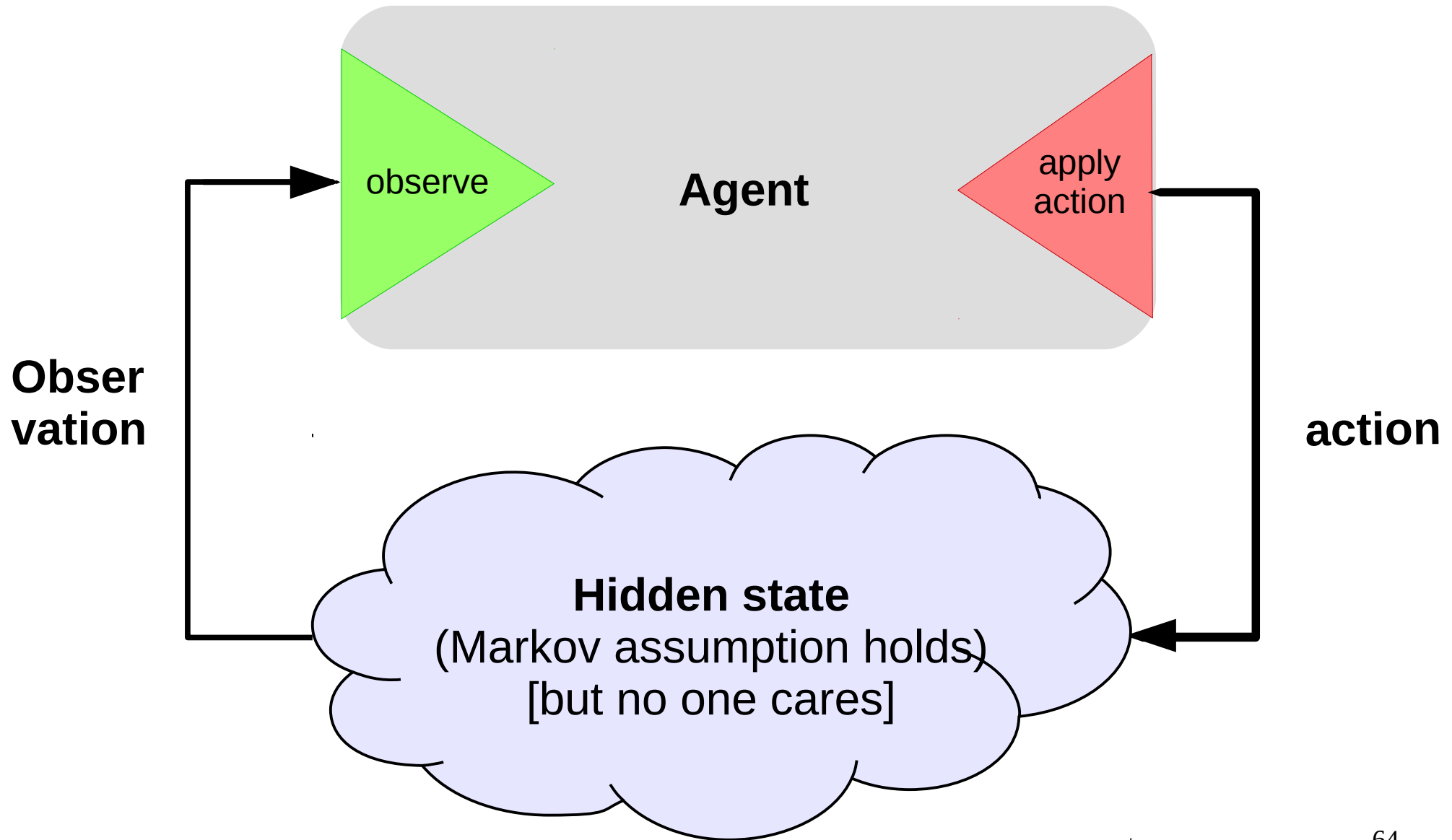
Agent observation does not hold all information about process state  
(e.g. human field of view).

- However, we can try to infer hidden states from sequences of observations.

$$s_t \simeq m_t : P(m_t | o_t, m_{t-1})$$

- Intuitively that's agent memory state.

# Partially observable MDP





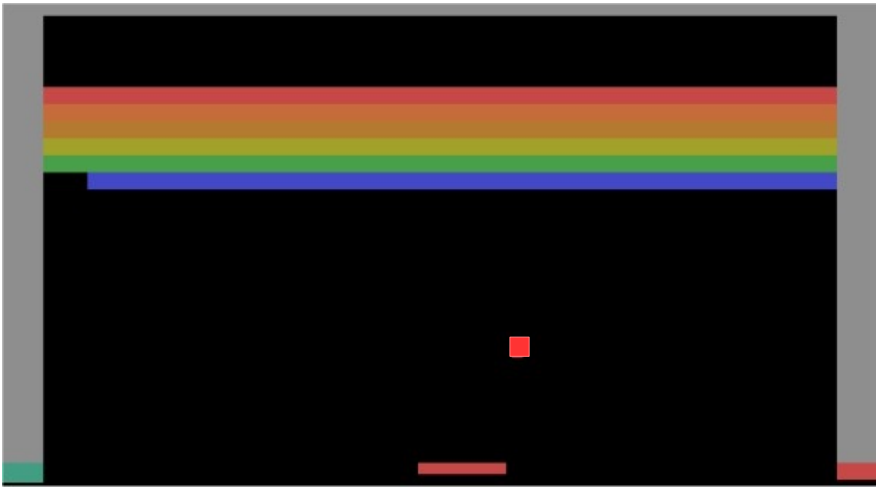
# N-gram heuristic

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

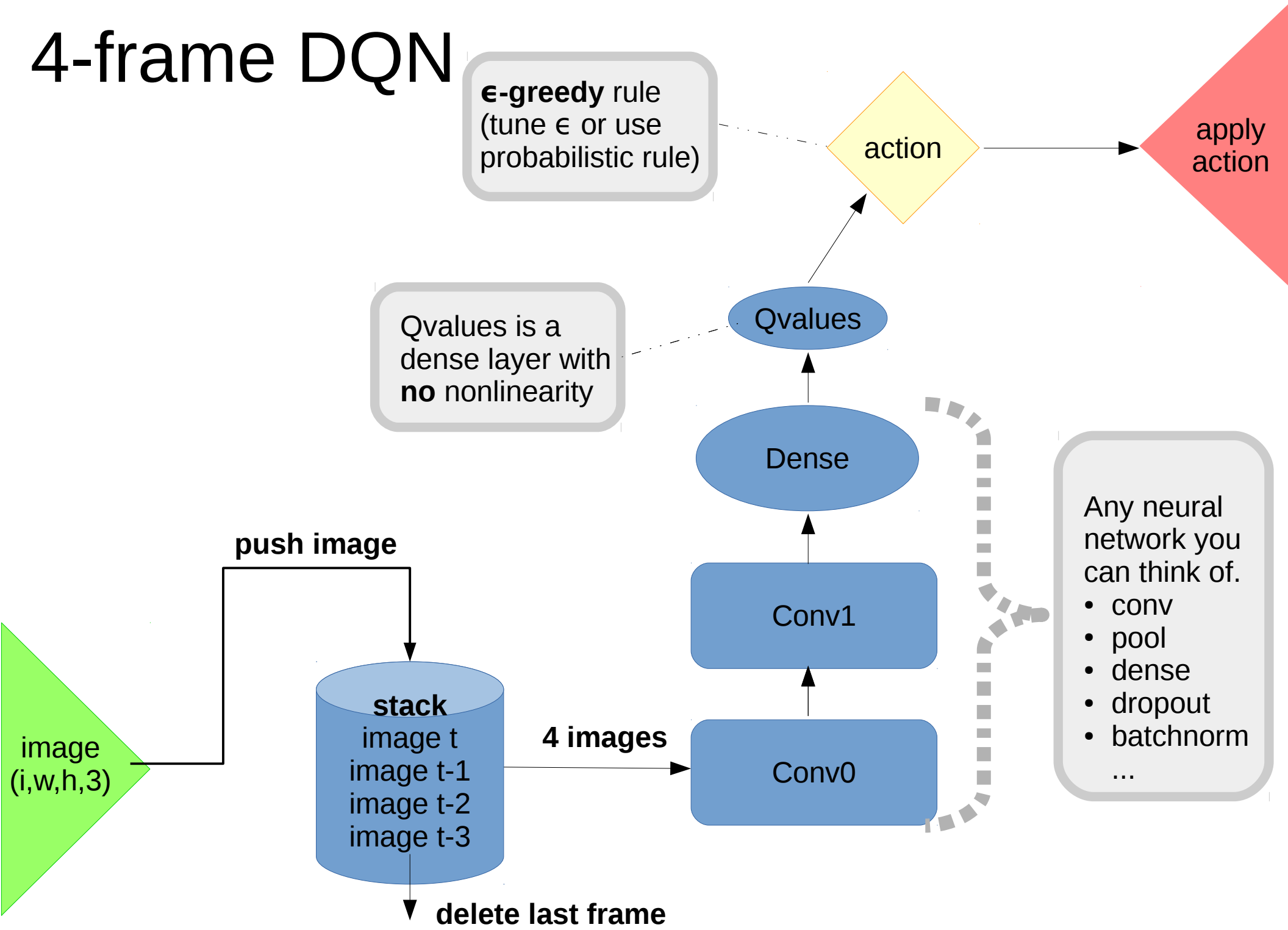


• One frame



• Several frames

# 4-frame DQN



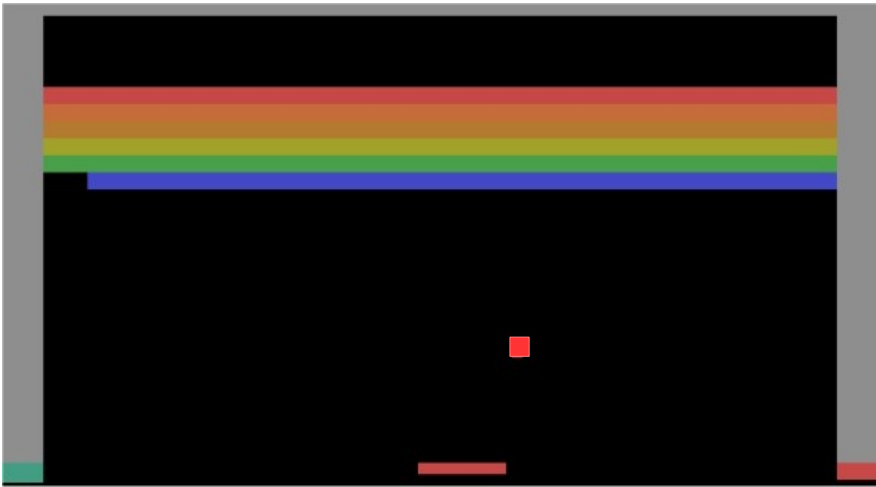
# N-gram heuristic

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

# Alternatives

## **Ngrams:**

- Nth-order markov assumption
- Works for velocity/timers
- Fails for anything longer than N frames
- Impractical for large N

## **Alternative approach:**

- Infer hidden variables given observation sequence
- Kalman Filters, Recurrent Neural Networks
- More on that in a few lectures