Reinforcement learning

Episode 2

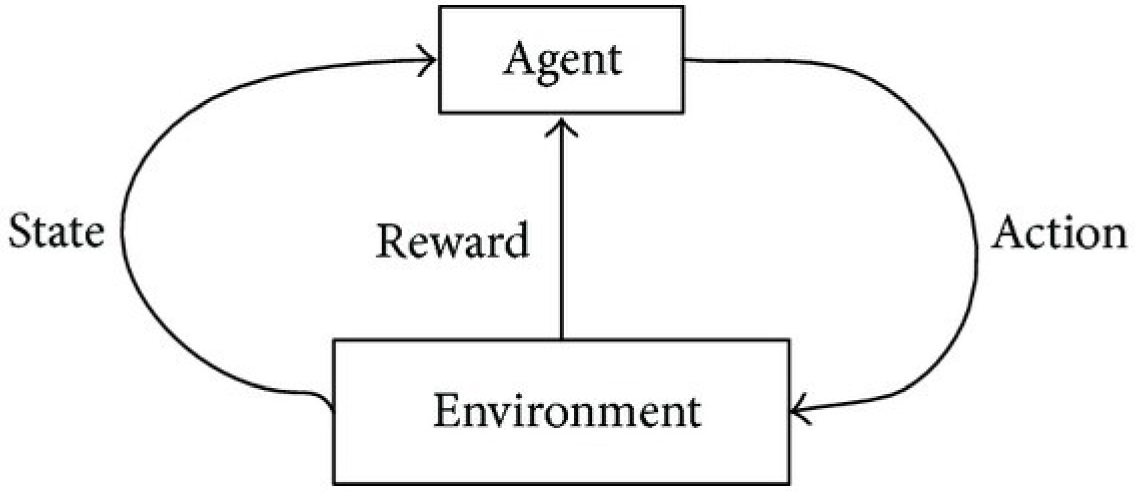
Approximate & deep RL

Yandex
Data Factory

LAMBDA

British Hedgehog
Preservation Society

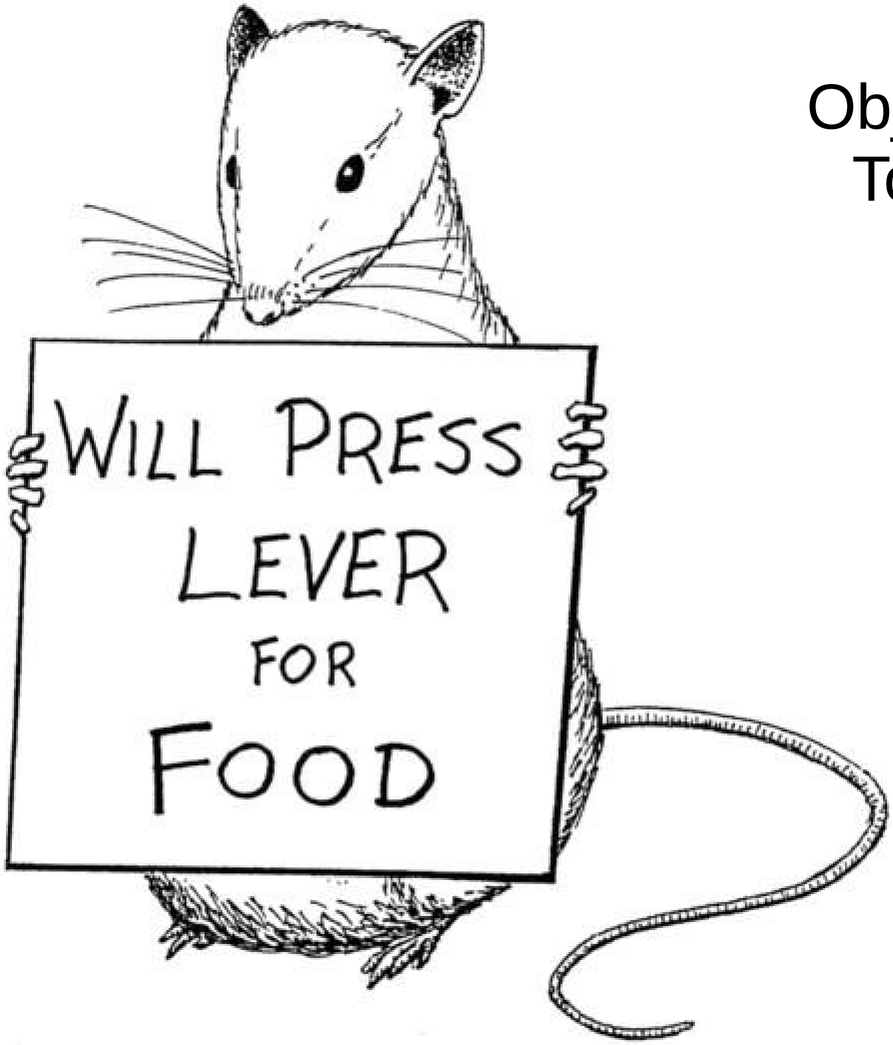
Recap: MDP



Classic MDP(Markov Decision Process)
Agent interacts with environment

- Environment states: $s \in S$
- Agent actions: $a \in A$
- State transition: $P(s_{t+1} | s_t, a_t)$
- Reward: $r_t = r(s_t, a_t)$

Recap: total reward



Objective:

Total reward

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

$$\gamma \in (0,1)\ const$$

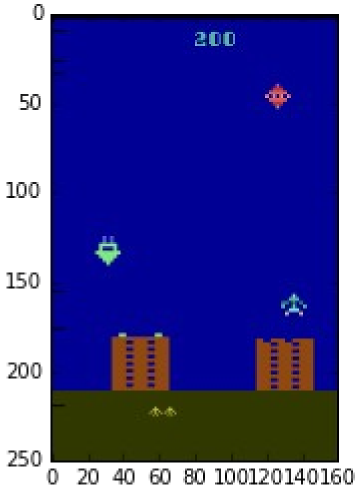
γ ~ patience

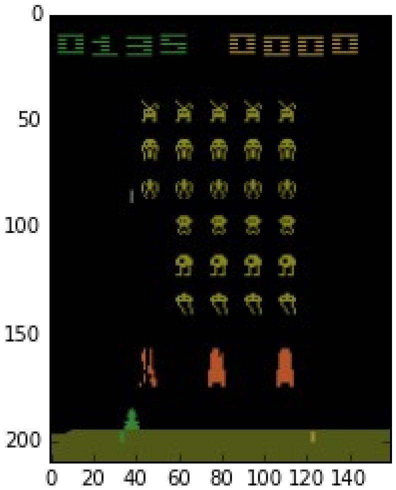
Reinforcement learning:

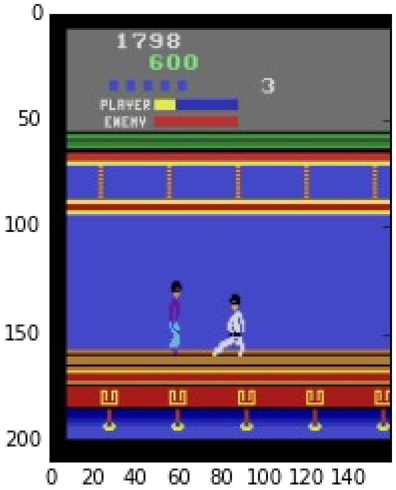
- Find policy that maximizes the expected reward

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

~~Reality~~ check: videogames

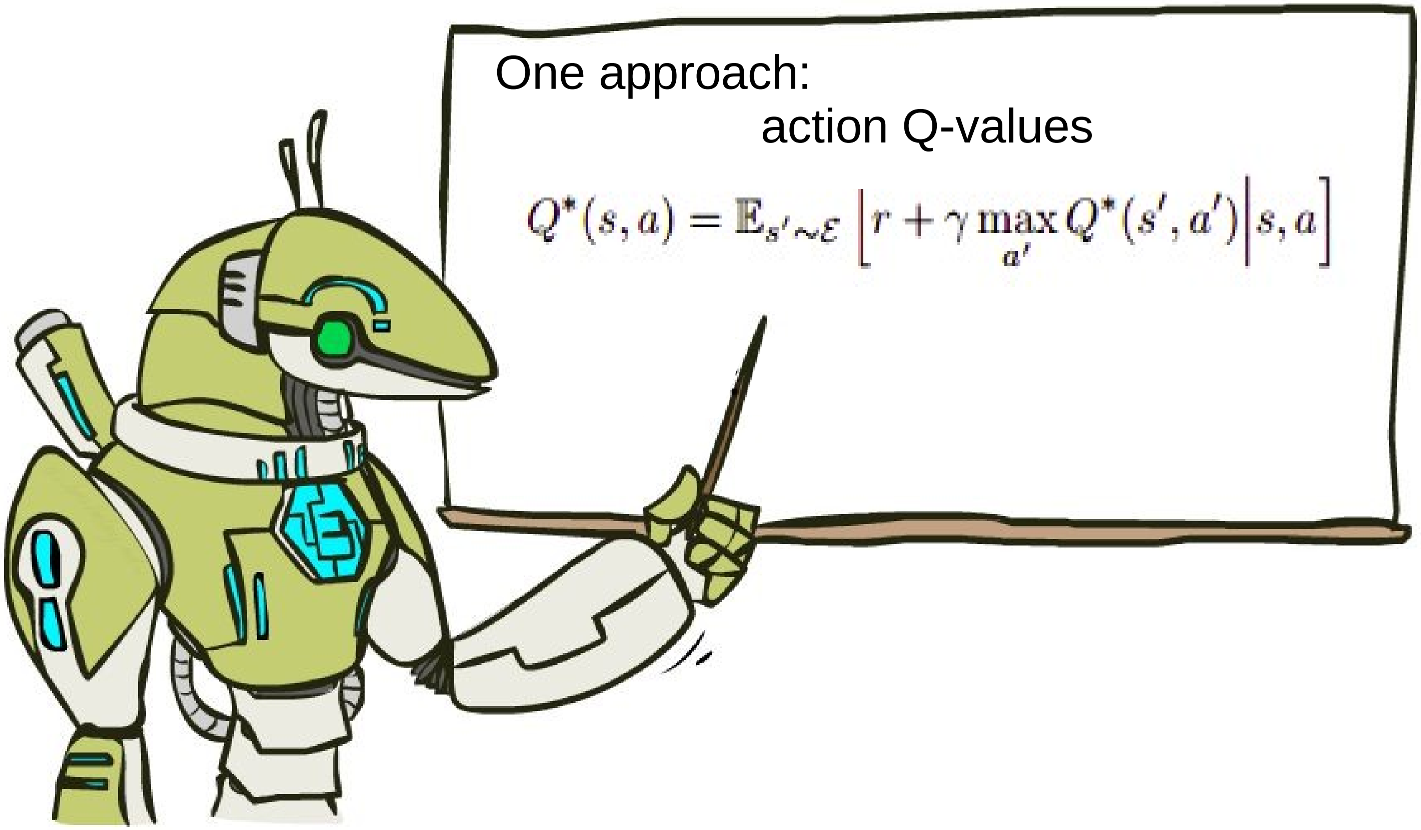






- **Trivia:** What are the states and actions?

Recap: Q-learning



One approach:

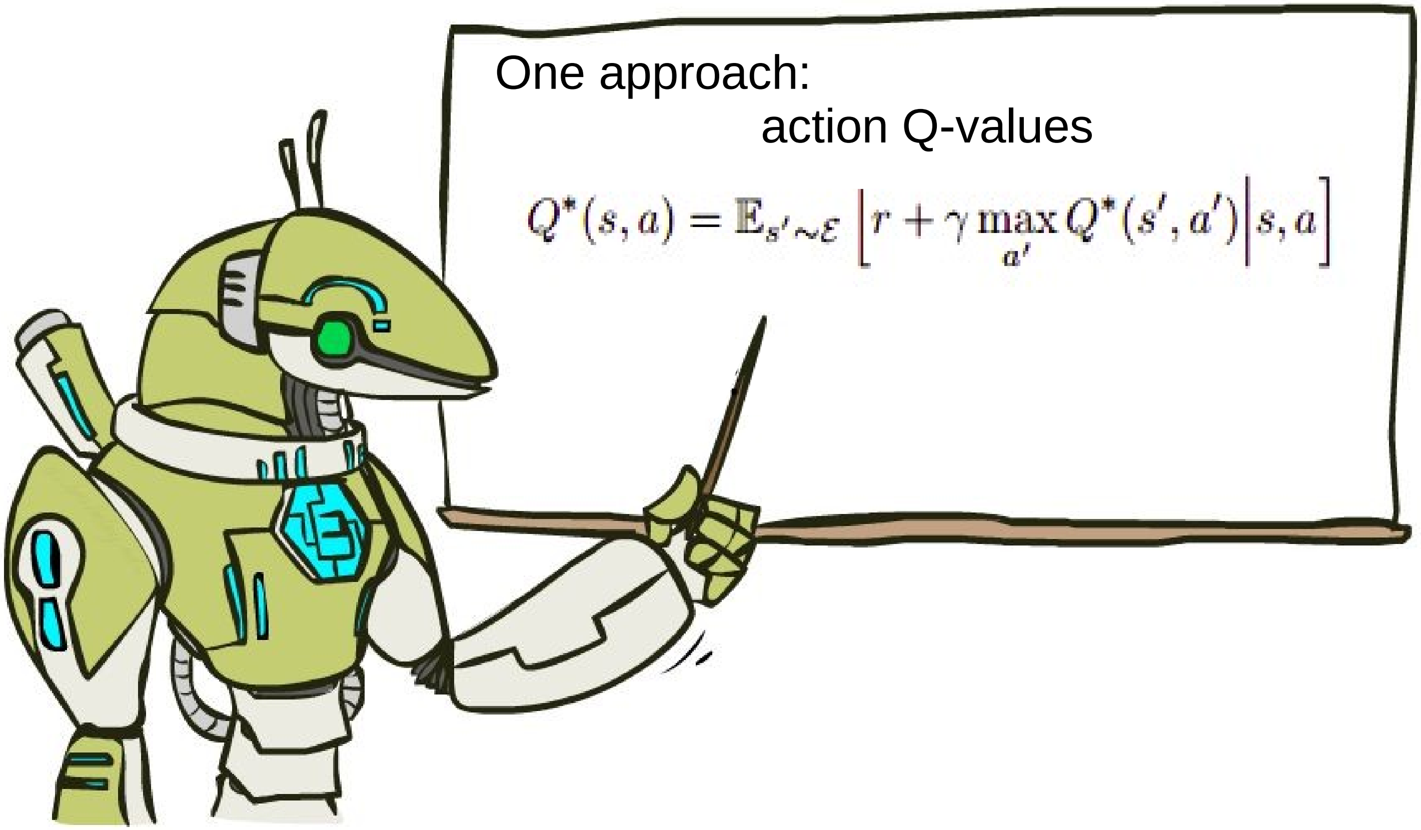
action Q-values

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a \right]$$

Definition: Q(s,a) is an expected total reward **R** that can be obtained starting from state **s** by taking action **a** and following optimal policy since next state.

$$\pi(s) : argmax_a Q(s,a)$$

Recap: Q-learning



One approach:

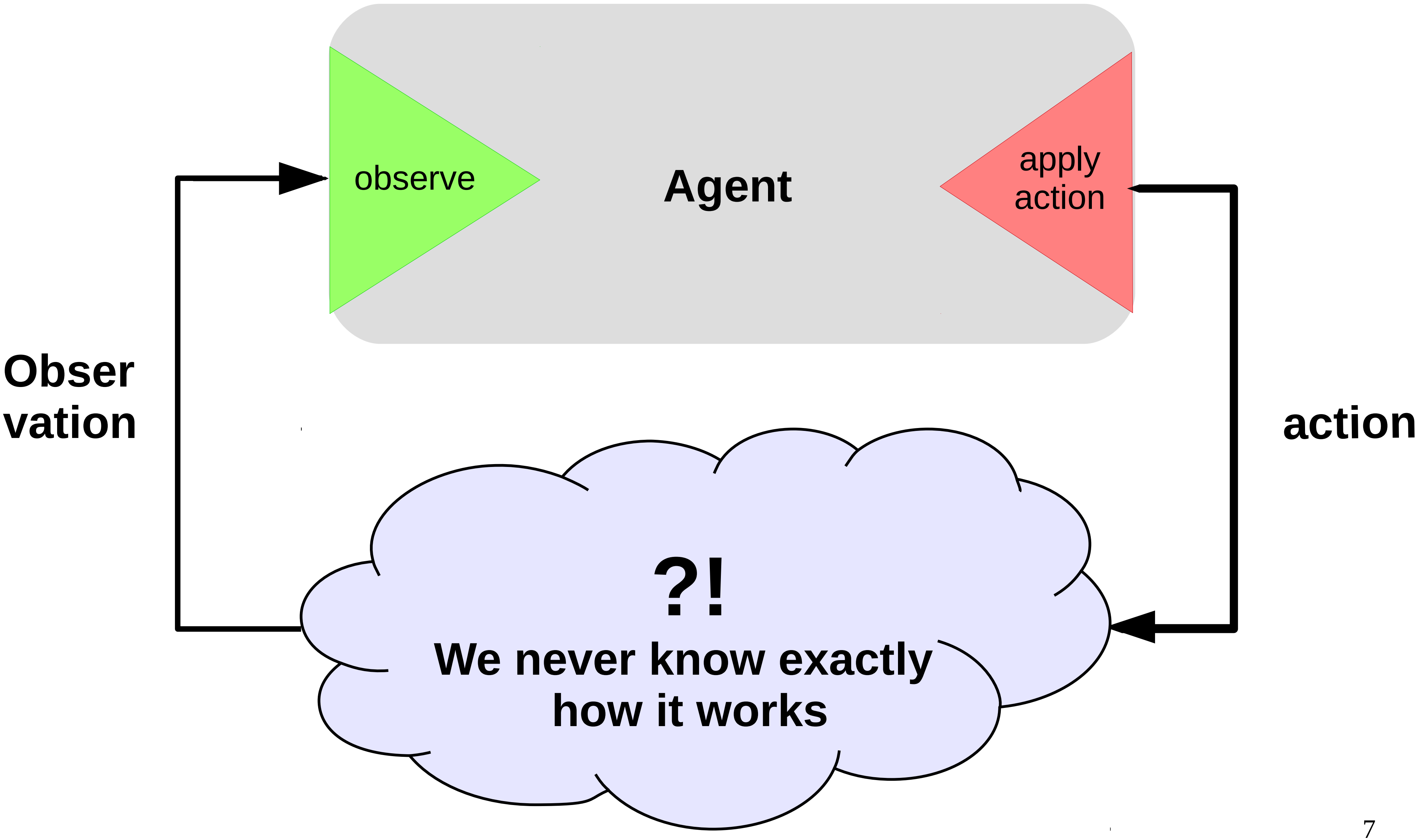
action Q-values

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a\right]$$

- Initialize with zeros/random
- Iteratively minimize

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

Real world


observe
Agent
apply
action
Obser
vation
action
?!
We never know exactly
how it works

Problem:

State space is usually large,

sometimes continuous.

And so is action space;

However, states do have a structure, similar states have similar action outcomes.

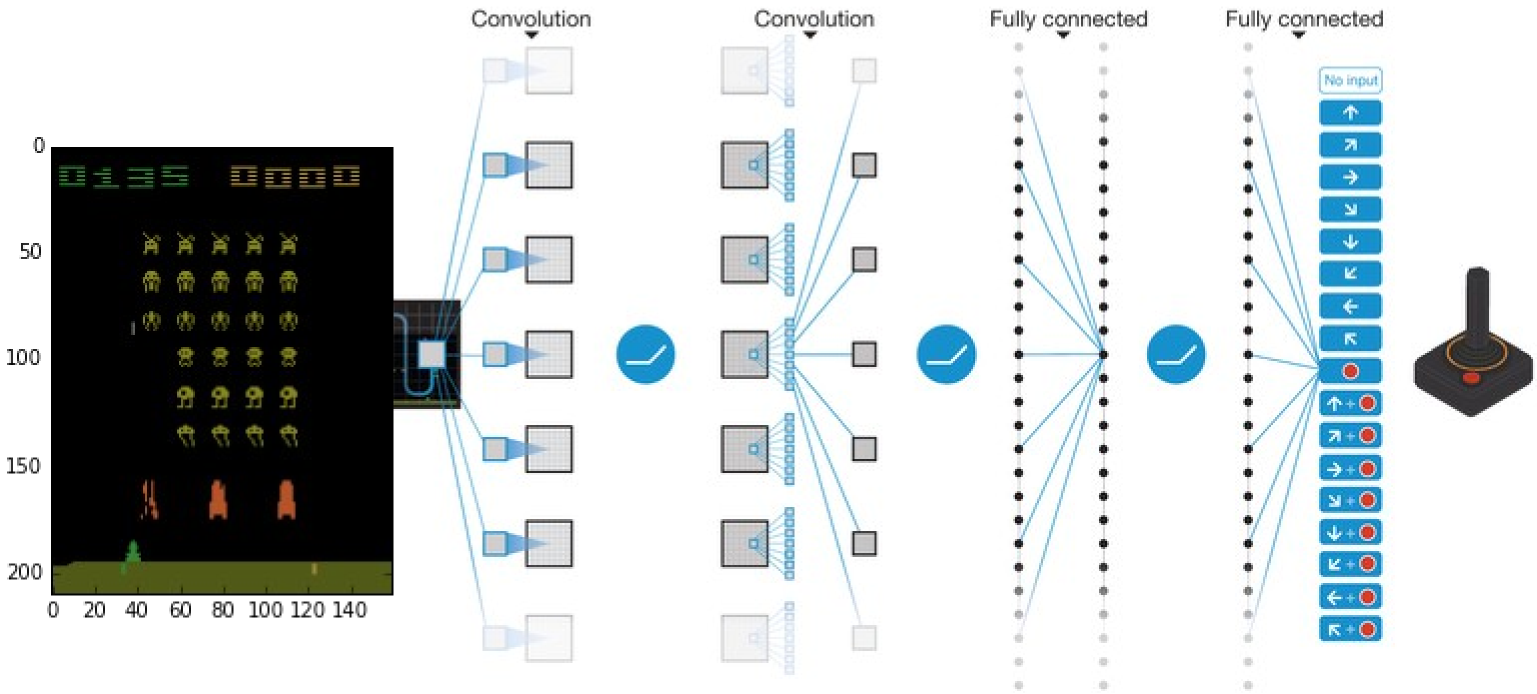
From tables to approximations

- Before:
  - For all states, for all actions, remember Q(s,a)
- Now:
  - Approximate Q(s,a) with some function
  - e.g. linear model over state features

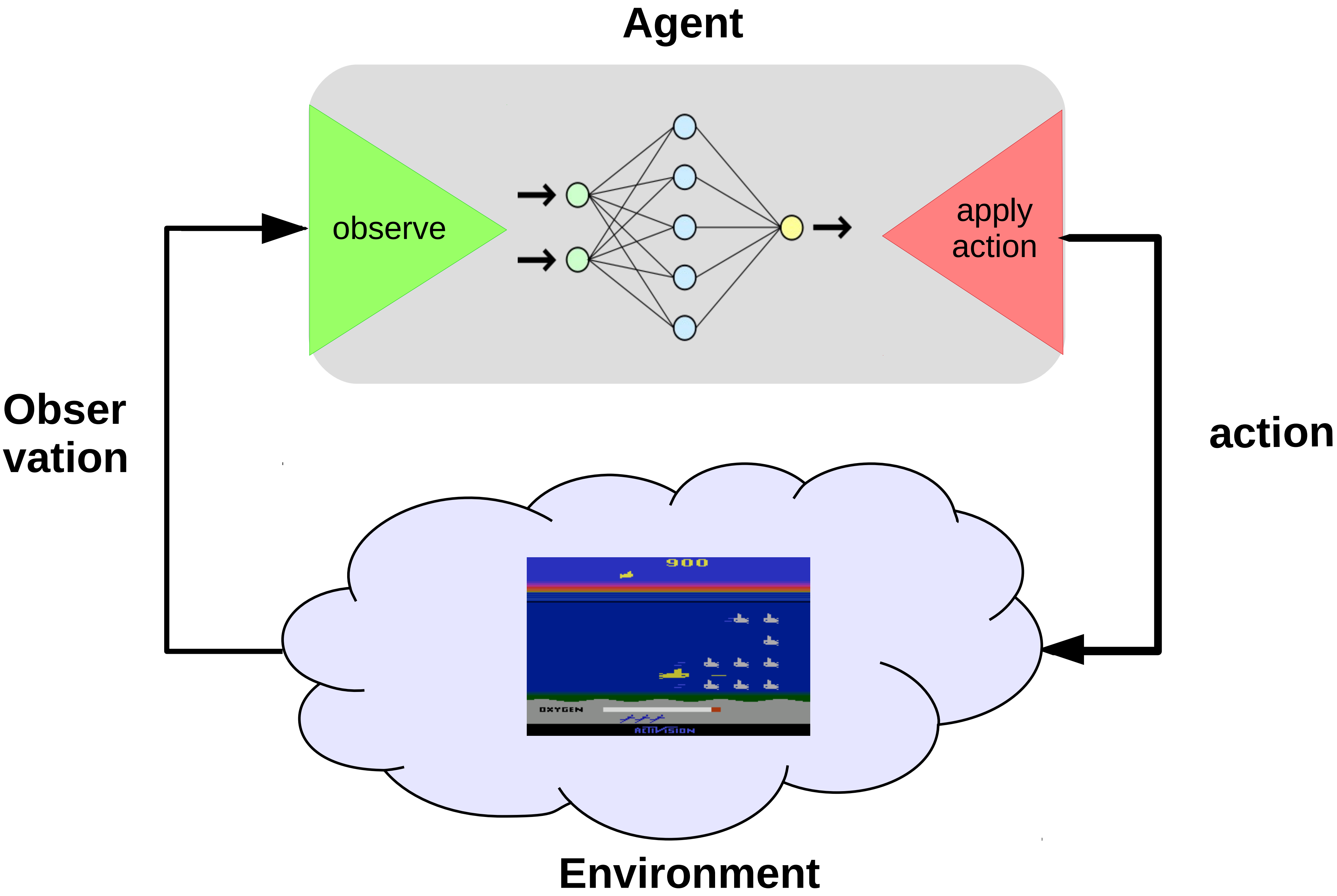
$$argmin_{w,b}\left(Q\left(s_t,a_t\right)-\left[r_t+\gamma\cdot max_{a'}Q\left(s_{t+1},a'\right)\right]\right)^2$$

Trivia: should we use linear regression or logistic regression?

Deep learning approach: DQN



MDP again



Agent

observe

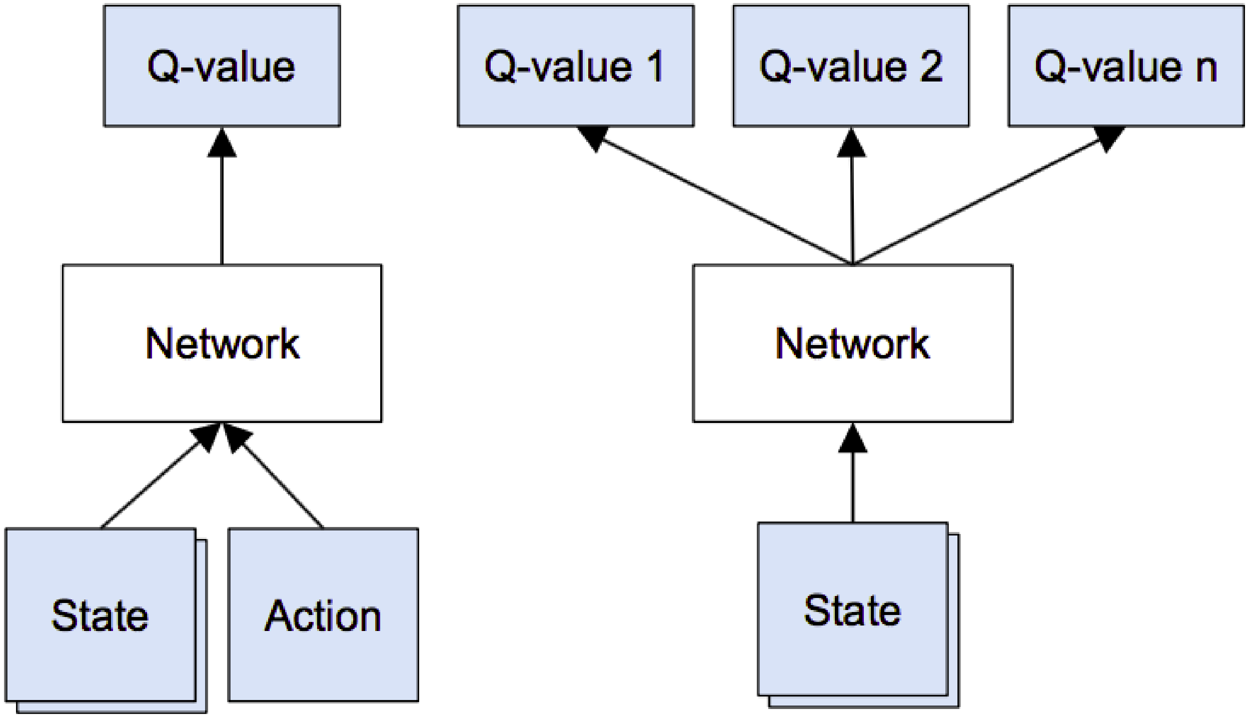
apply
action

Obser
vation

action

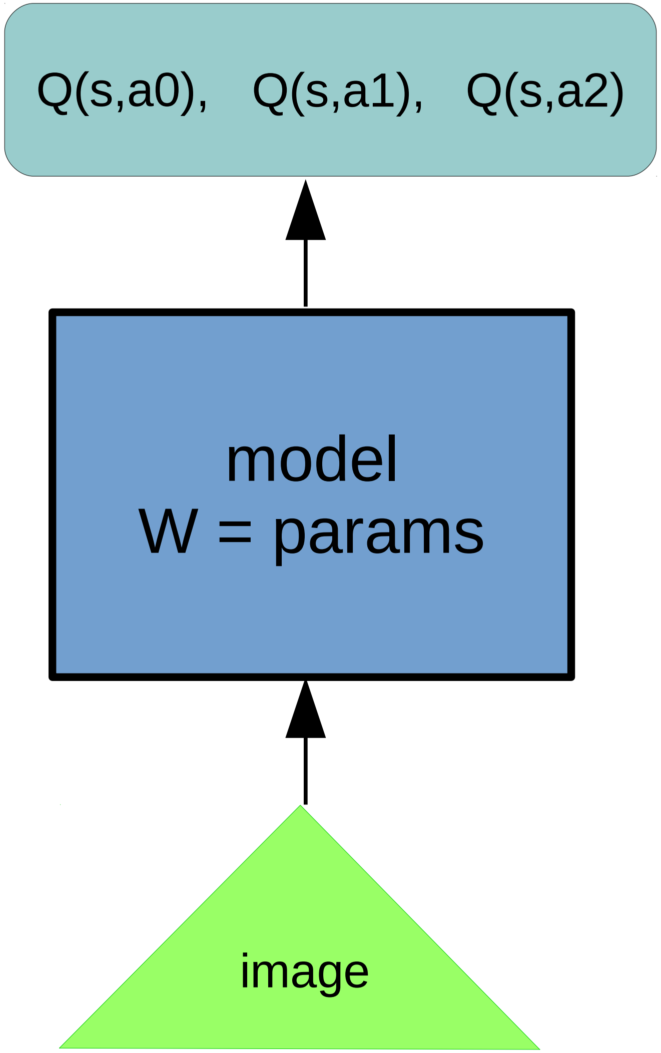
Environment

Deep learning approach: DQN



$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i) \right)^2 \right]$$

Approximate Q-learning



Q(s,a0), Q(s,a1), Q(s,a2)

model
W = params

image

Q-values:

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

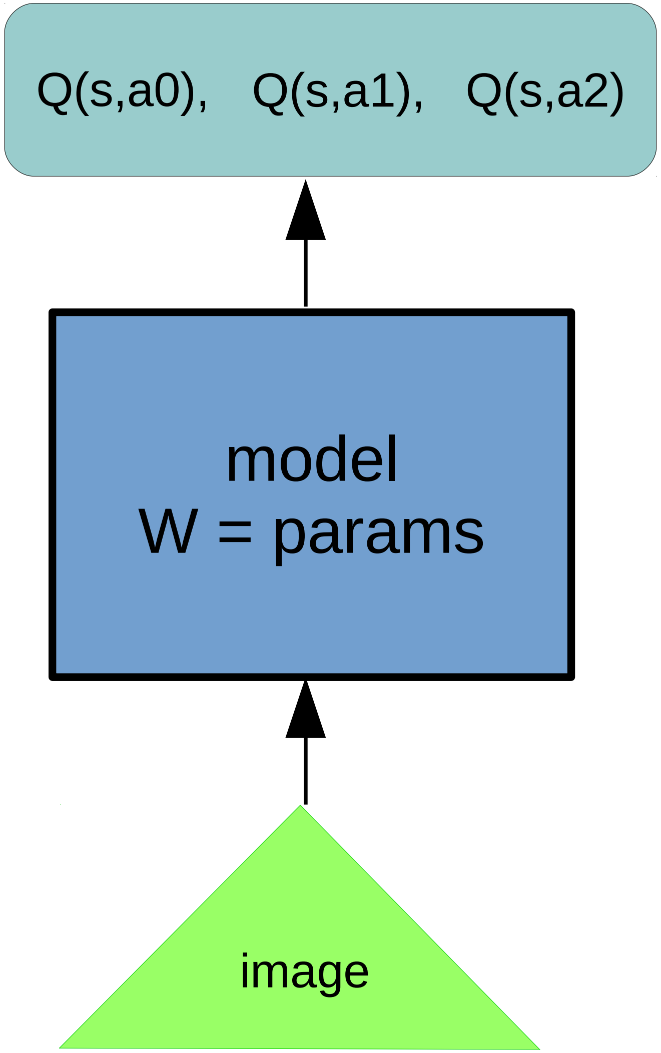
Objective:

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

Gradient step:

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

Approximate Q-learning



Q(s,a0), Q(s,a1), Q(s,a2)

model

W = params

image

Q-values:

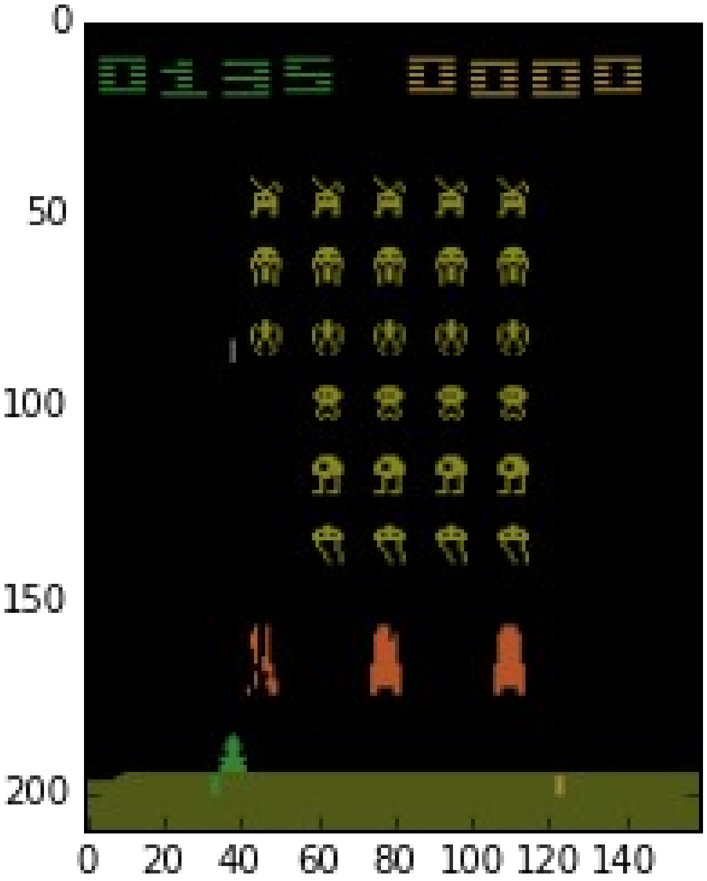
$$\hat{Q}(s_t,a_t)=r+\gamma\cdot argmax_{a'}\hat{Q}(s_{t+1},a')$$

Objective:

$$L=\left(Q(s_t,a_t)-\left[r+\gamma\cdot \underbrace{argmax_{a'}Q(s_{t+1},a')\right]}_{\textbf{consider const}}\right)^2$$

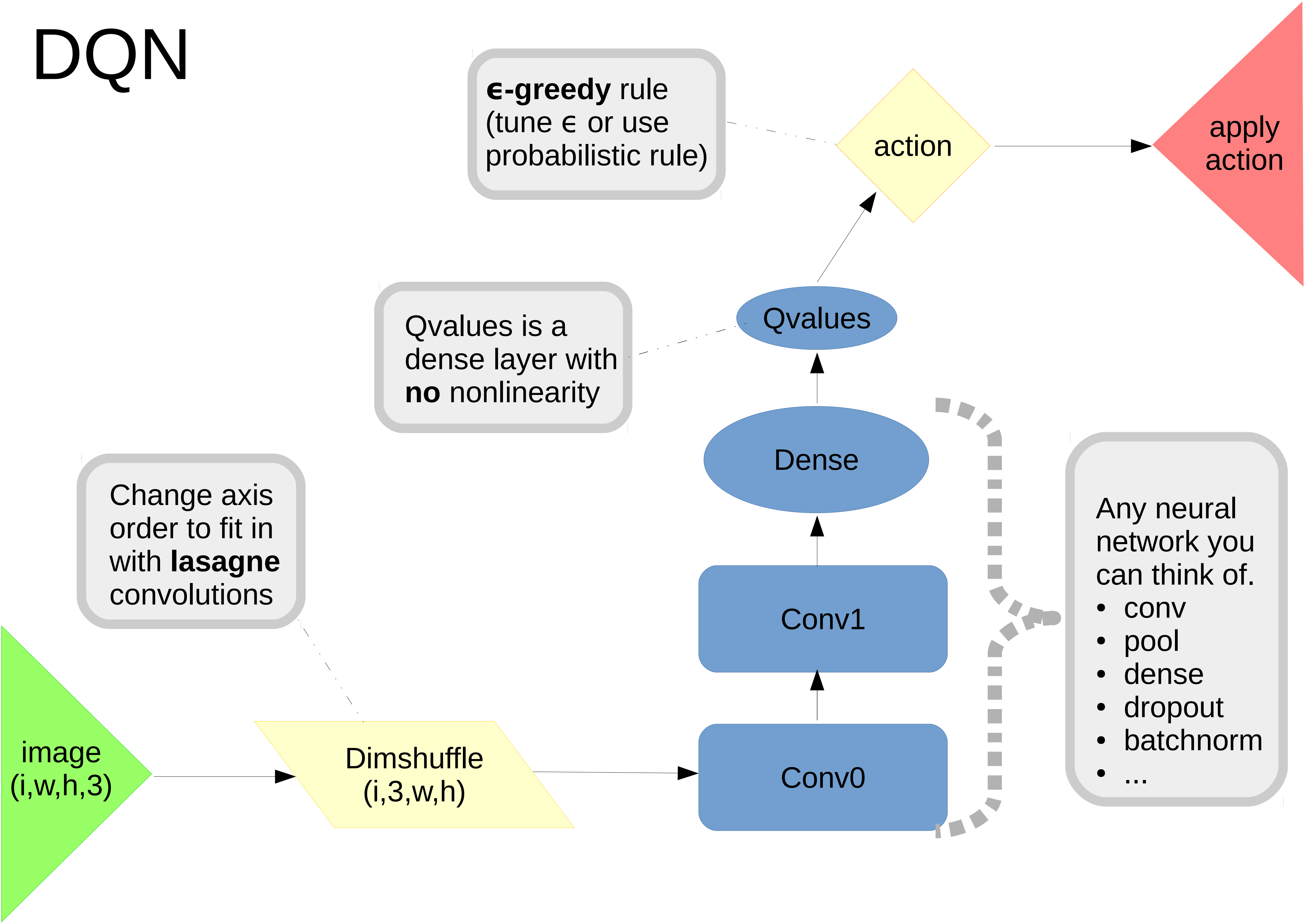
Gradient step:

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



Let's devise some network

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.

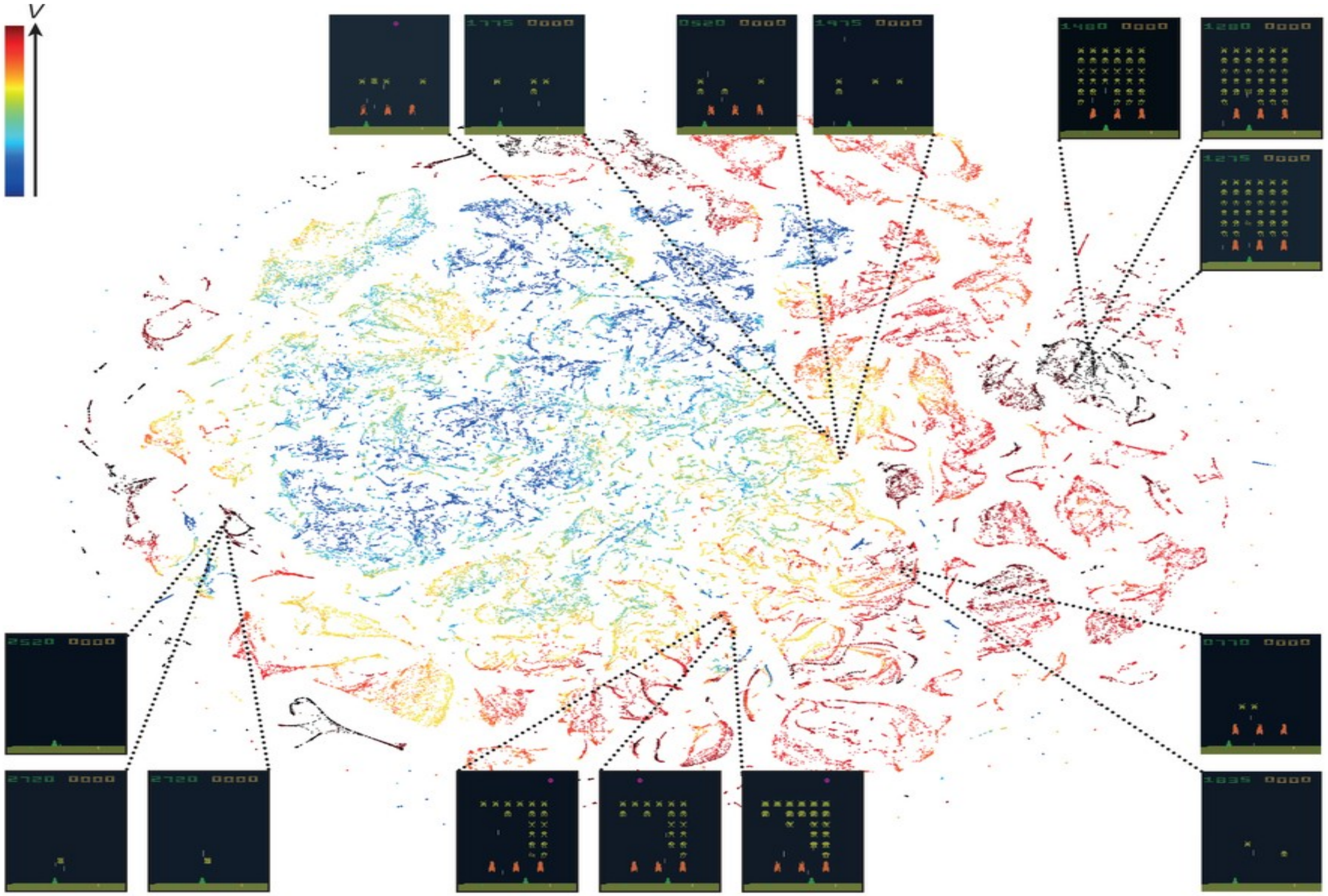
- conv
- pool
- dense
- dropout
- batchnorm
- ...

Change axis
order to fit in
with **lasagne**
convolutions

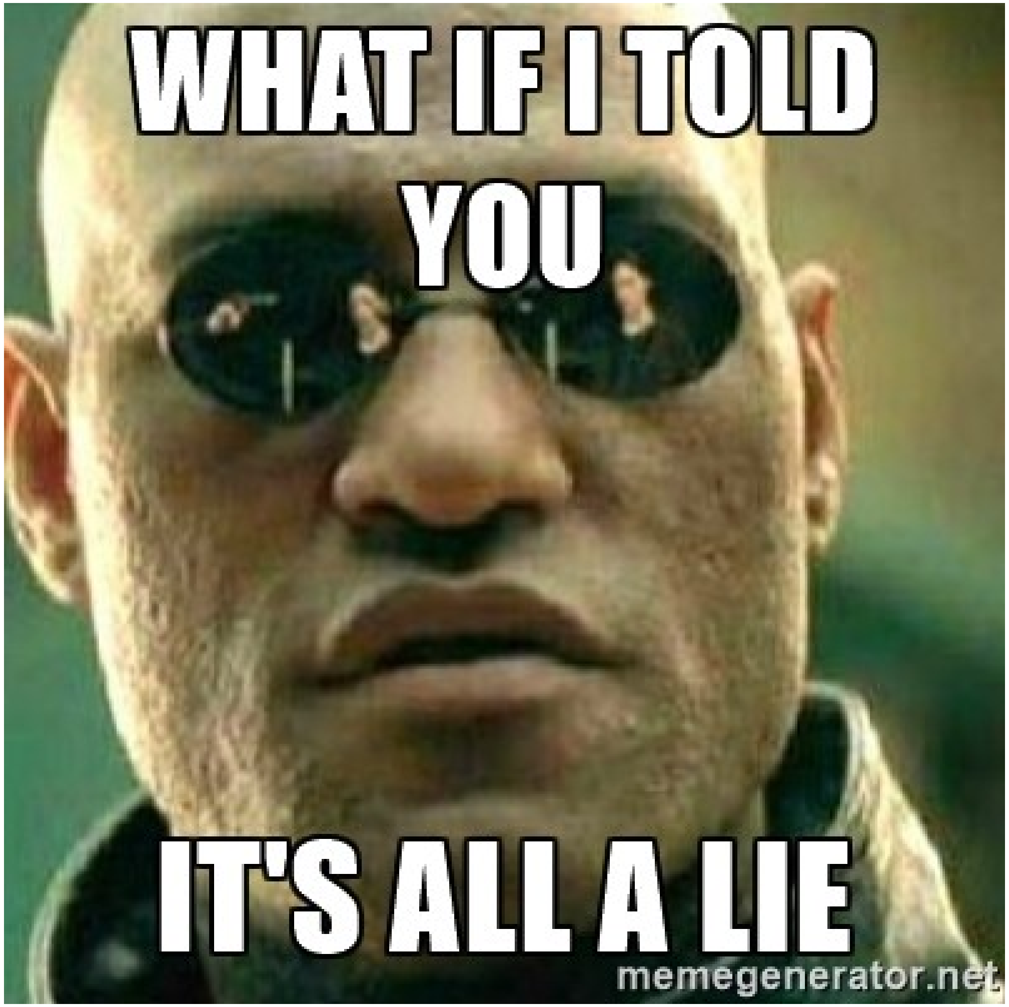
image
(i,w,h,3)

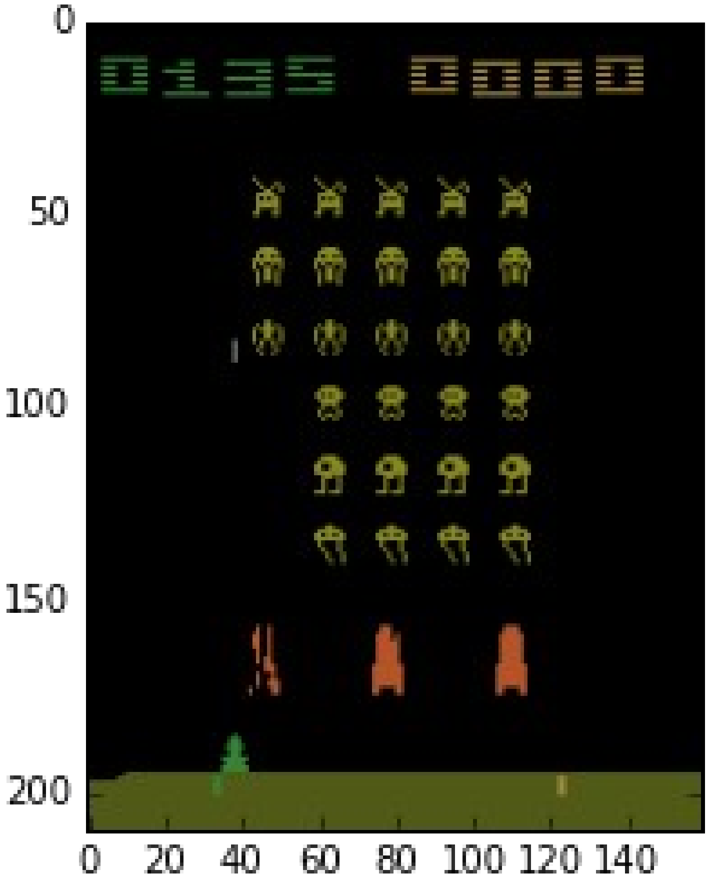
Dimshuffle
(i,3,w,h)

Because TSNE



- Embedding of pre-last layer activations
- Color = state value = max_a Q(s,a)


WHAT IF I TOLD YOU
IT'S ALL A LIE
memegenerator.net

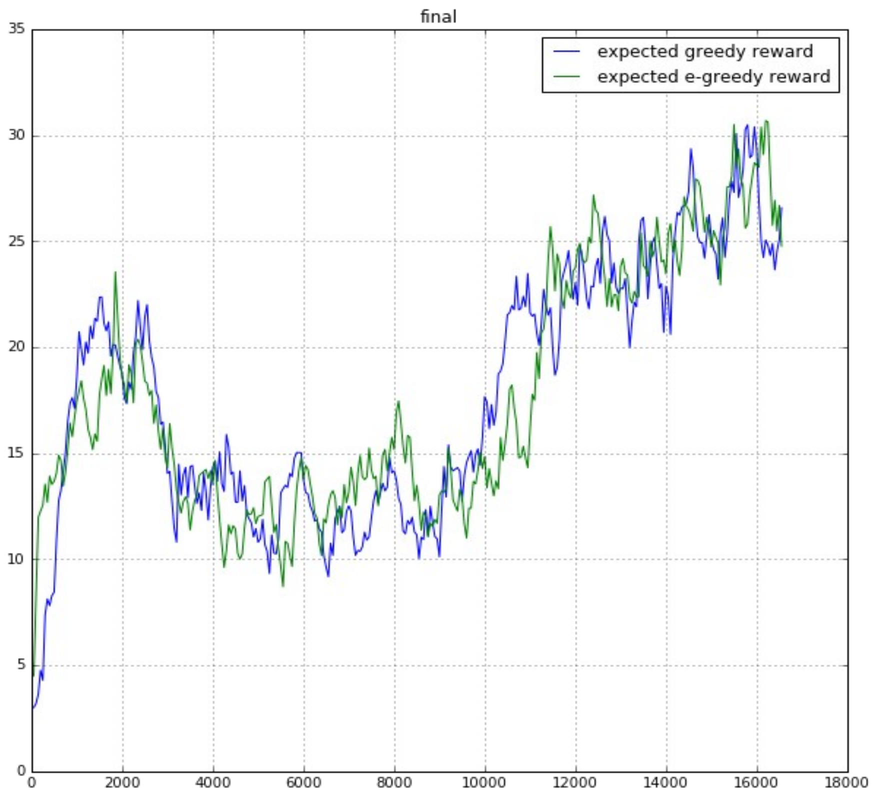


How bad is it if agent spent

last 1000 ticks under the left rock?

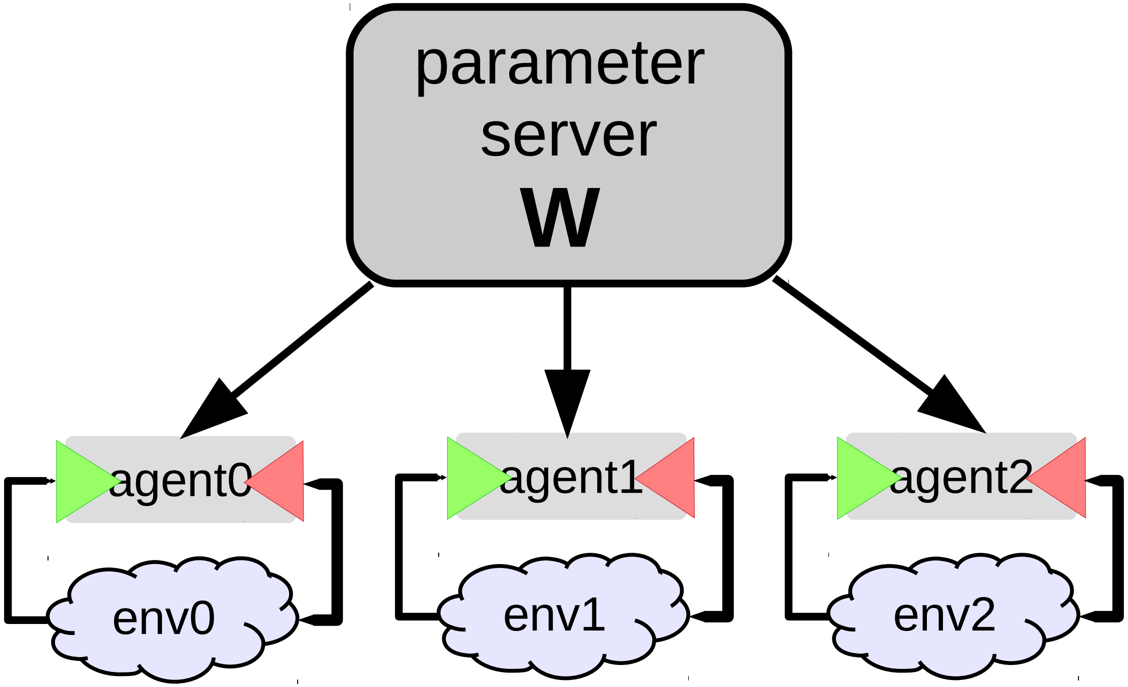
Problem

- 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?**



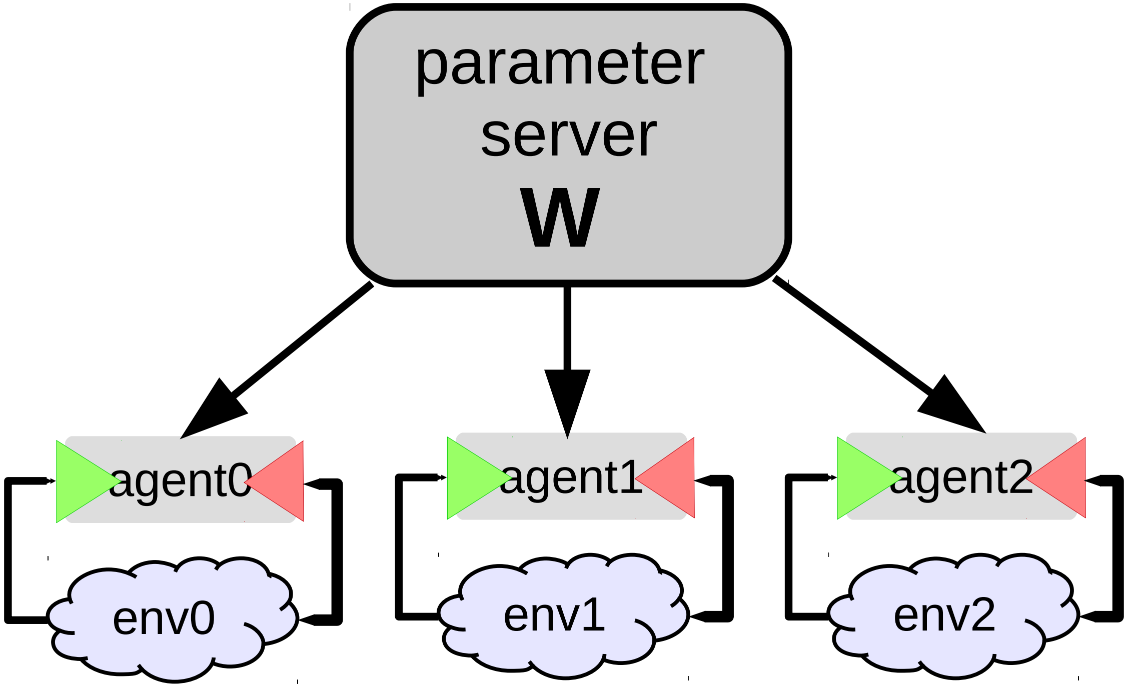
Multiple agent trick

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



Multiple agent trick

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

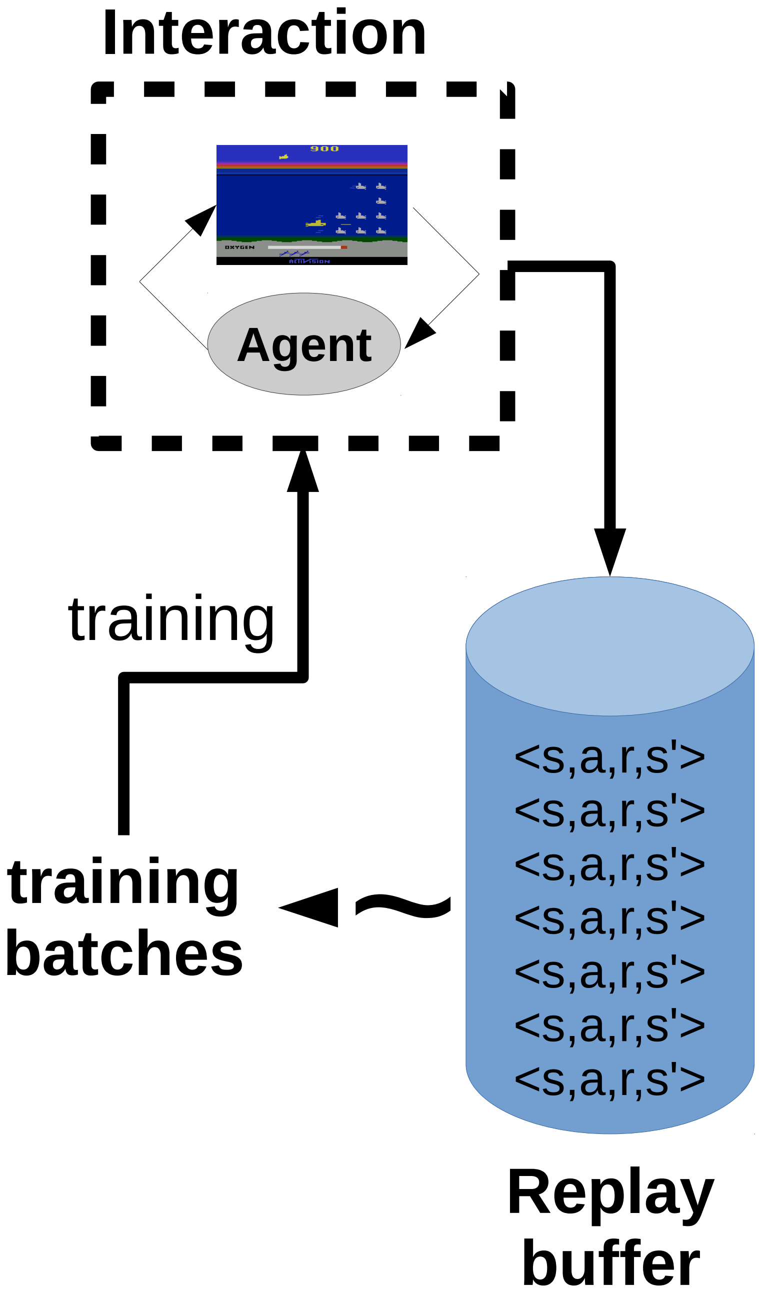


- Chances are, they will be exploring different parts of the environment,
- More stable training,
- Requires a lot of interaction,
- Alternative to experience replay.

Experience replay

Idea: store several past interactions
<s,a,r,s'>
Train on random subsamples

Any +/- ?



Interaction

Agent

training

training batches

~

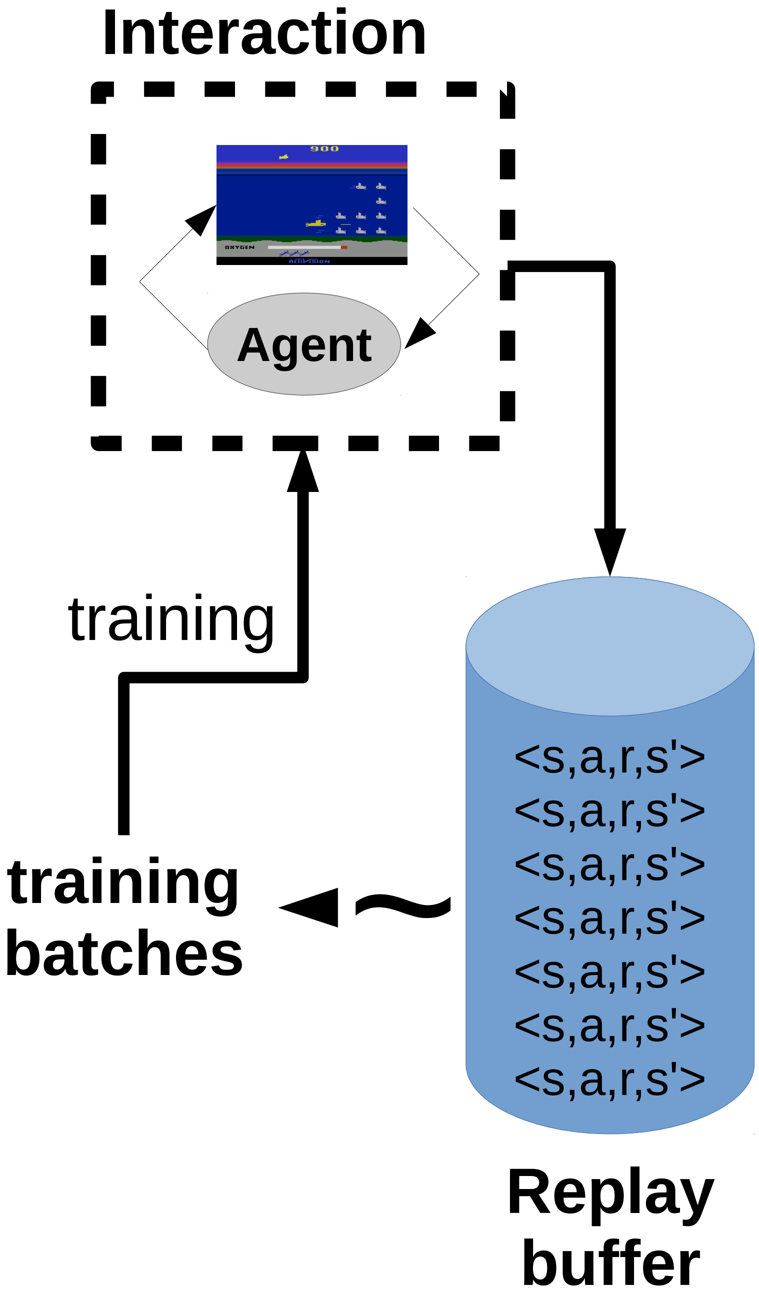
<s,a,r,s'>
<s,a,r,s'>
<s,a,r,s'>
<s,a,r,s'>
<s,a,r,s'>
<s,a,r,s'>
<s,a,r,s'>

Replay buffer

Experience replay

Idea: store several past interactions
<s,a,r,s'>
Train on random subsamples

- Atari DQN: >10^5 interactions
- Closer to i.i.d
pool contains several sessions
- Older interactions were obtained under weaker policy
- Further development: prioritizing samples based on their importance. https://arxiv.org/abs/1511.05952



Interaction

Agent

training

training batches

~

<s,a,r,s'>
<s,a,r,s'>
<s,a,r,s'>
<s,a,r,s'>
<s,a,r,s'>
<s,a,r,s'>
<s,a,r,s'>

Replay buffer

Autocorrelation

- Reference is based on predictions

$$r+\gamma\cdot argmax_{a'}Q\left(s_{t+1},a'\right)$$

- Any error in Q approximation is propagated to neighbors
- If some Q(s,a) is mistakenly over-exaggerated, neighboring qvalues will also be increased in a cascade
- Worst case: divergence
- **Any ideas?**

Target networks

Idea: use older network snapshot
to compute reference

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

- Update Q old periodically
  - Slows down training

Target networks

Idea: use older network snapshot
to compute reference

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

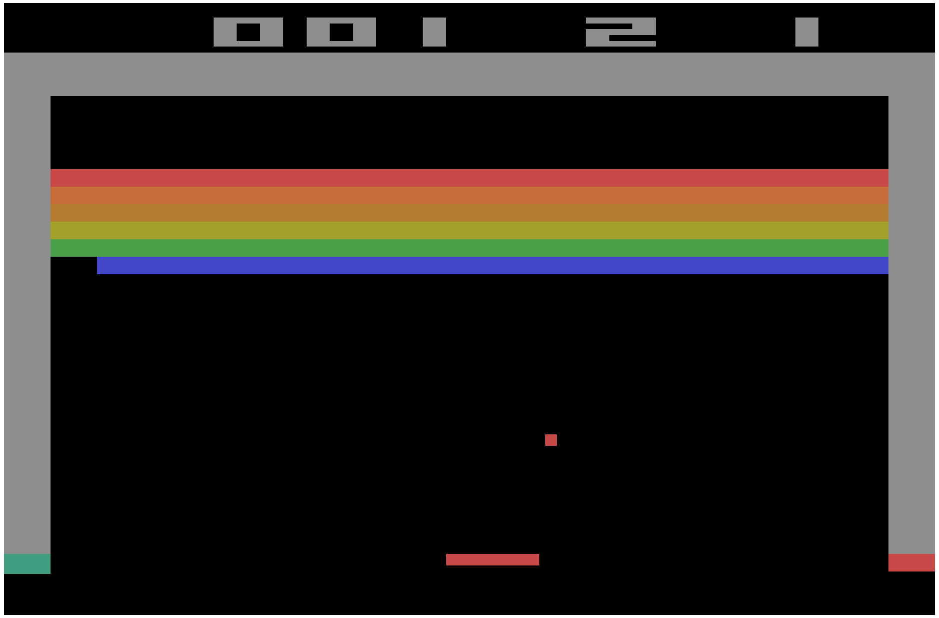
- Update Q old periodically
  - Slows down training

- Smooth version:
  - use moving average

$$\theta^{old} := (1 - \alpha) \cdot \theta^{old} + \alpha \cdot \theta^{new}$$

  - Θ = weights

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:

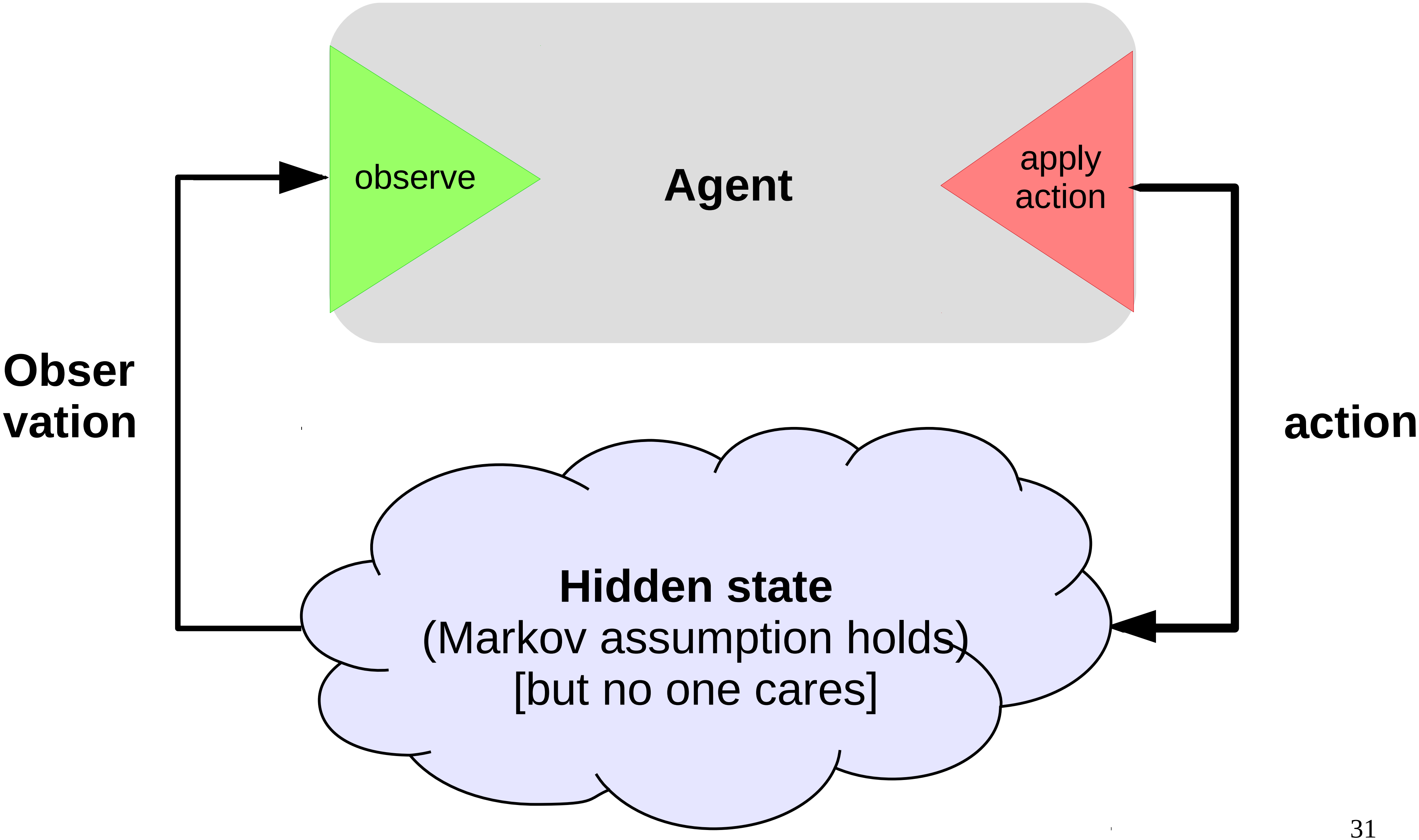
Agent observation does not hold all information about process state (e.g. human field of view).

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

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

- Intuitively that's agent memory state.

Partially observable MDP



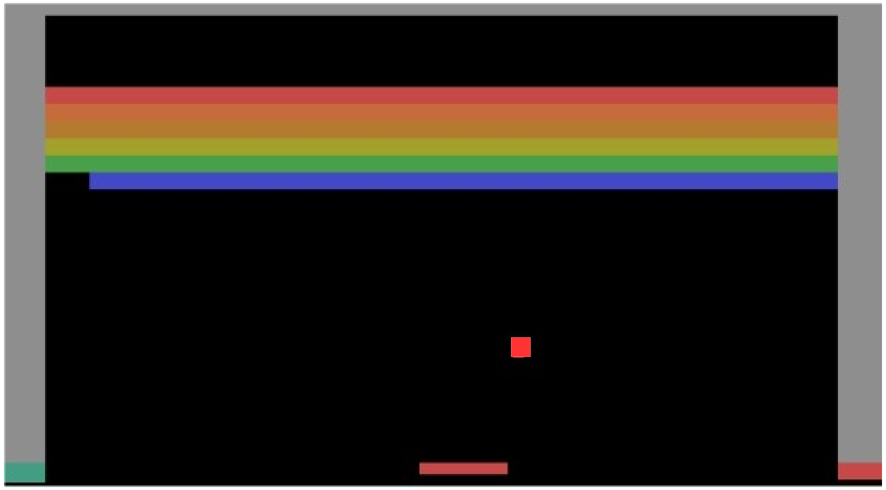
N-gram heuristic

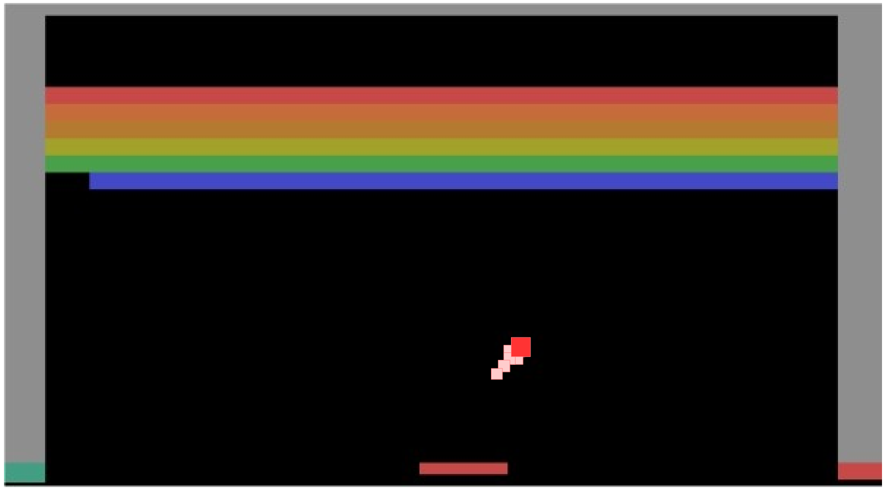
Idea:

$$s_t \neq o(s_t)$$

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

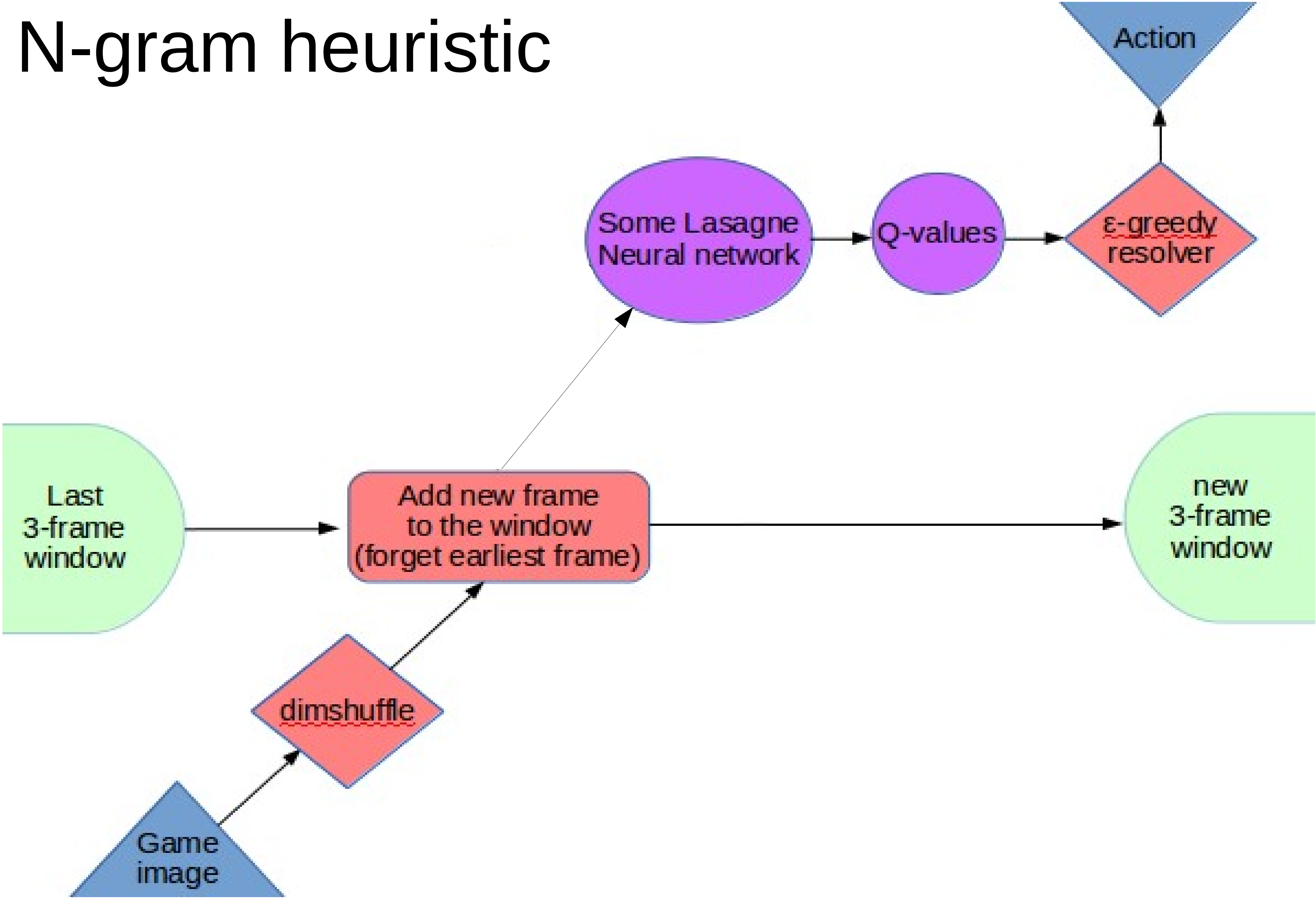
e.g. ball movement in breakout



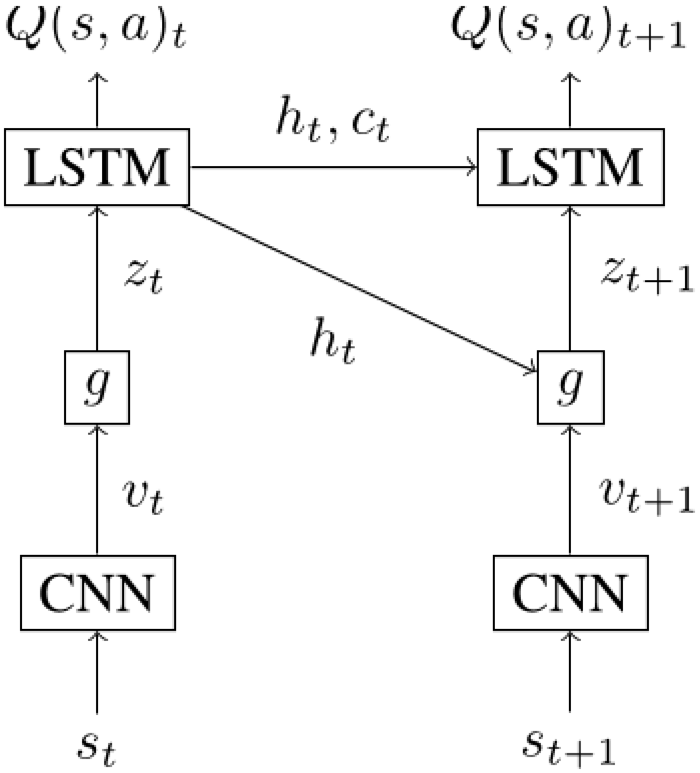


- One frame
- Several frames

N-gram heuristic

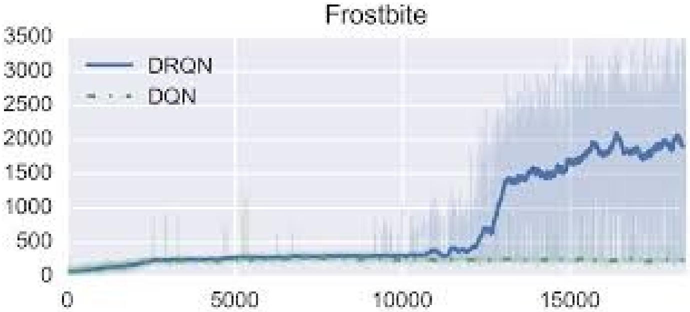


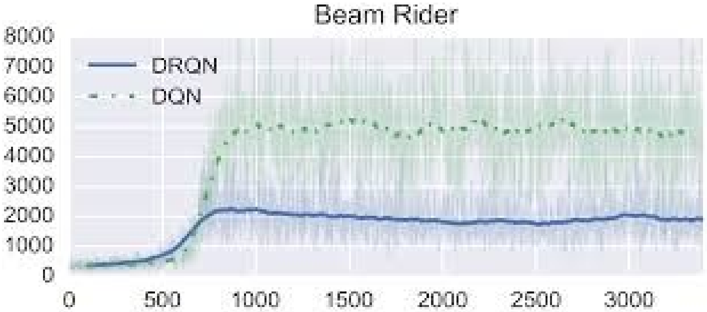
Deep Recurrent RL

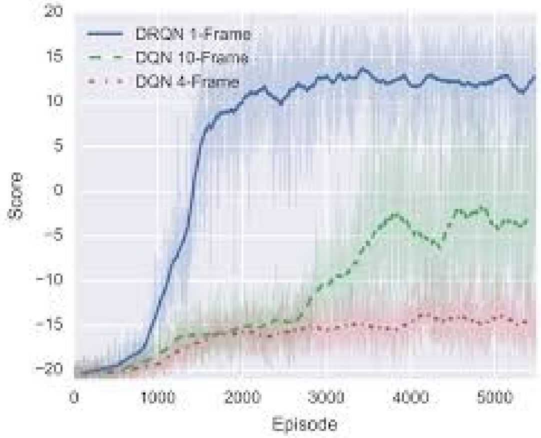


Recurrent agent memory

- Agent has his own hidden state.
- Trained via BPTT with a fixed depth
- Problem: next input depends on chosen action
- Even more autocorrelations :)

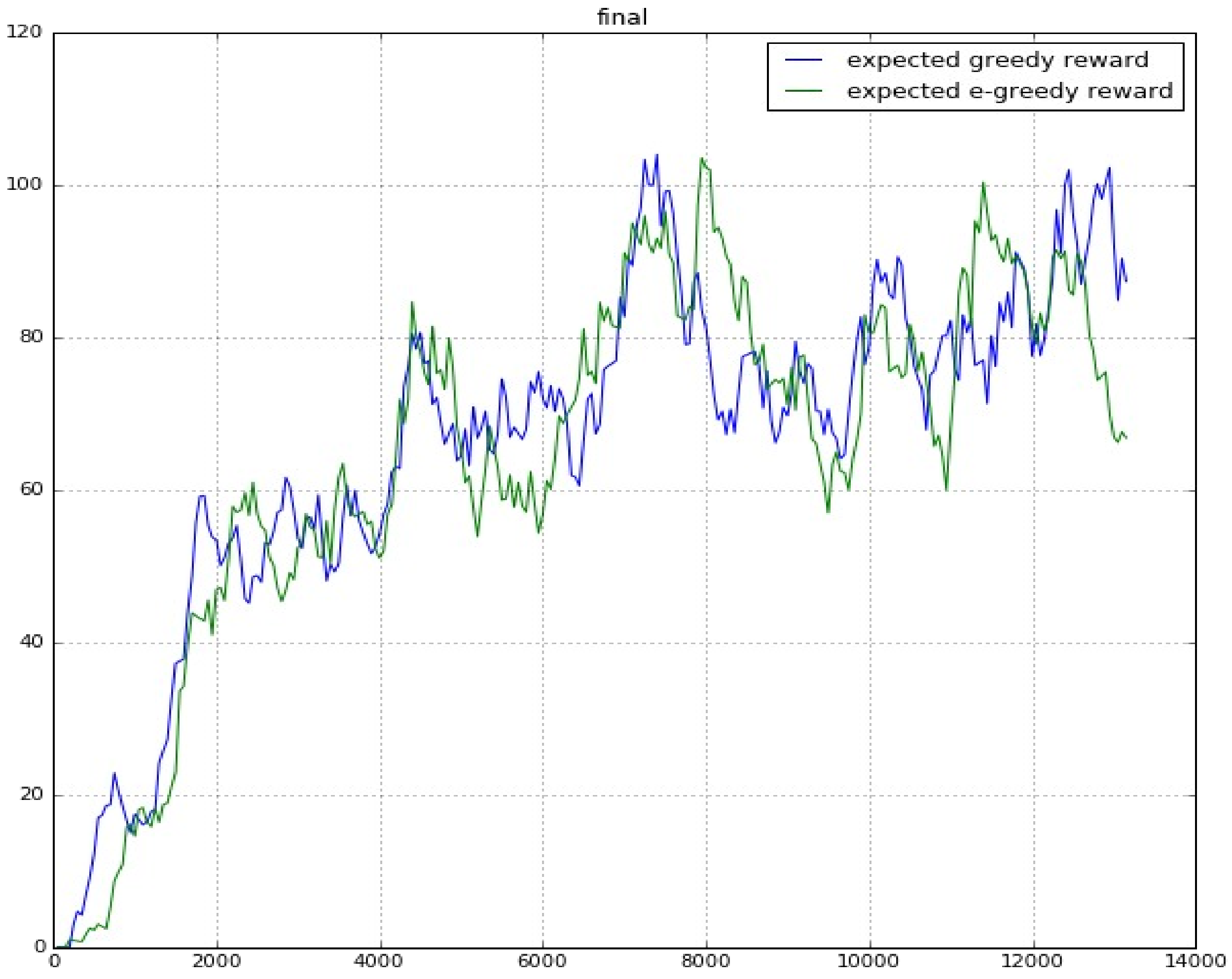






Deep Recurrent RL

Learning curves for KungFuMaster



http://mybinder.org/repo/yandexdataschool/dqn_binder

Most important slide

RL isn't magical

- It won't learn everything in the world given any data
- Requires interaction
- Sparse and/or delayed rewards are a major problem
- Less playing Atari, more real world problems
  No, doom is not a real world problem, dummy!
- Getting rid of heuristics towards mathematical soundness
- Machine Intelligence revolution date TBA