Reinforcement learning

Episode 4

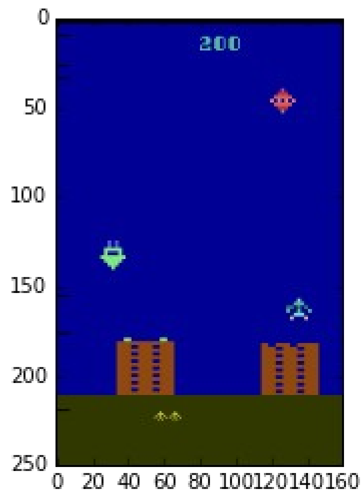
Approximate reinforcement learning

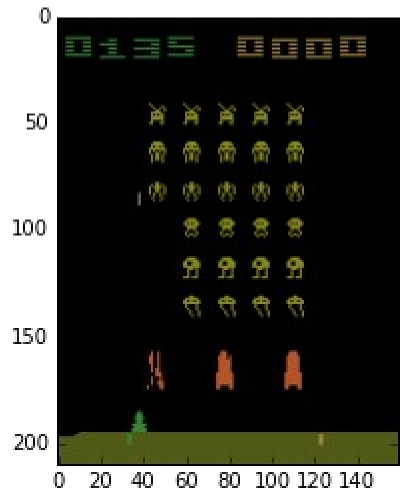
Yandex
Data Factory

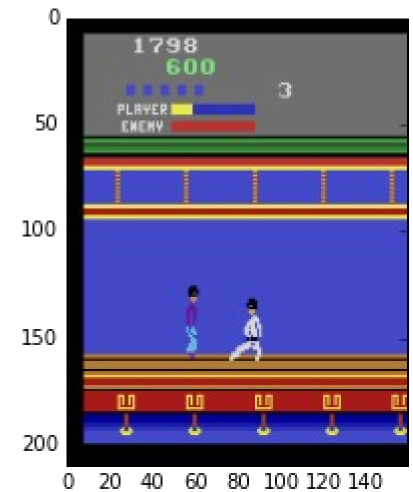
LAMBDA

British Hedgehog
Preservation Society

~~Reality~~ check: videogames

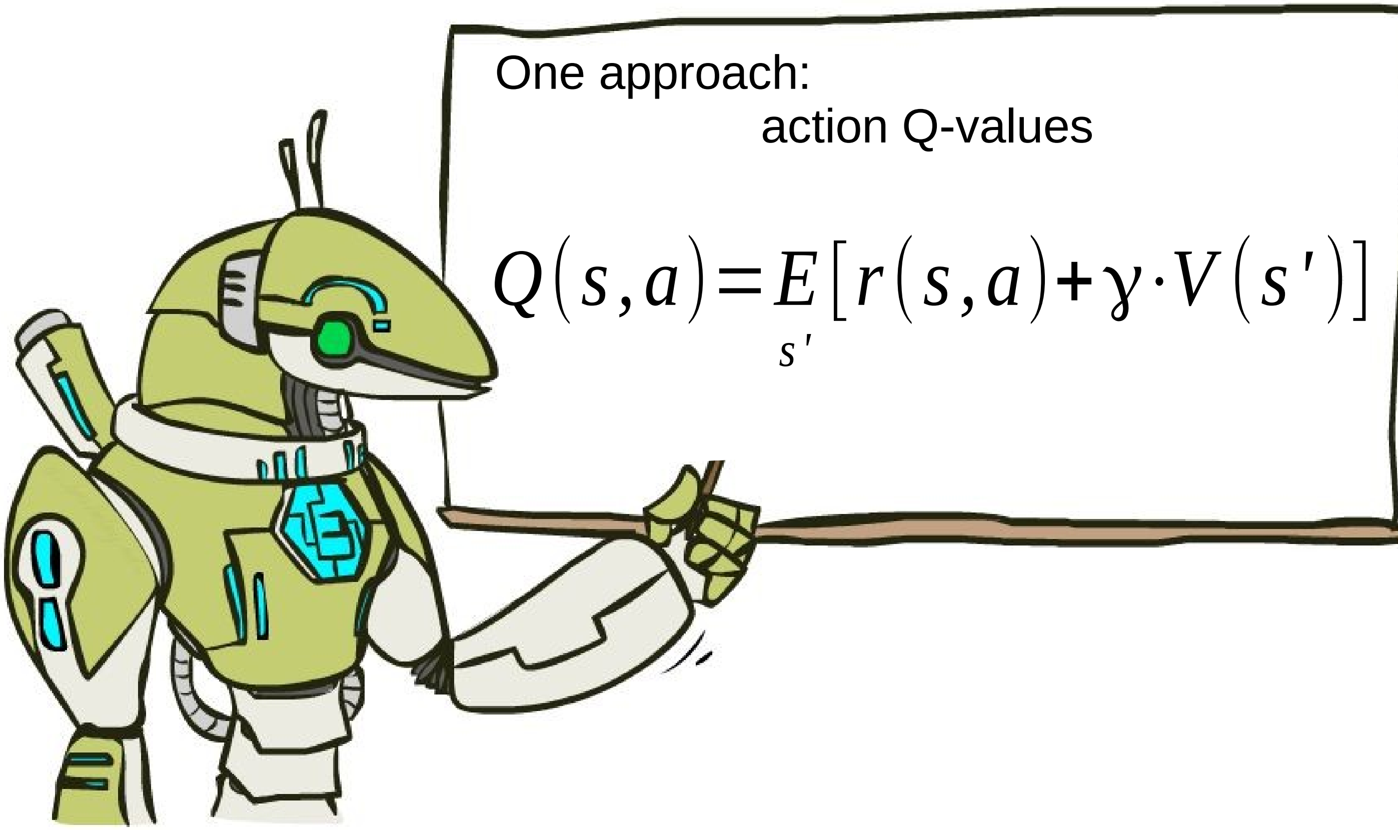






- **Trivia:** What are the states and actions?

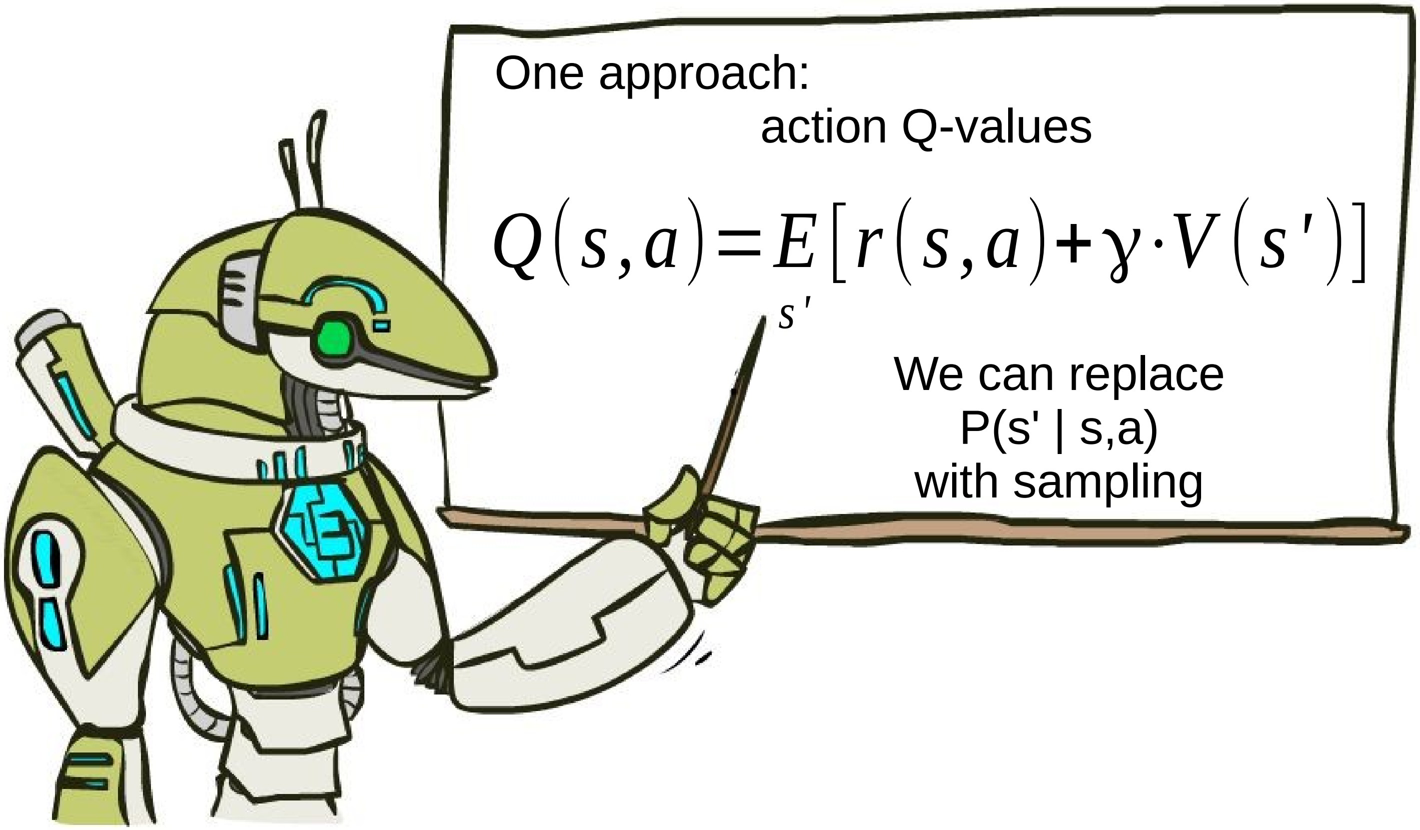
Recap: Q-learning



Action value Q(s,a) is the expected total reward **R** agent gets from state **s** by taking action **a** and following policy **π** from next state.

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

Recap: Q-learning



$$Q(s_t, a_t) \leftarrow \alpha \cdot \left(r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a')\right) + (1-\alpha) Q(s_t, a_t)$$

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

Q-learning as MSE minimization

Given <**s**,**a**,**r**,**s'**> minimize

$$L = [Q(s_t, a_t) - Q^{true}(s_t, a_t)]^2$$

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

How to optimize?

Q-learning as MSE minimization

Given <**s**,**a**,**r**,**s'**> minimize

$$L=[Q(s_t,a_t)-Q^{true}(s_t,a_t)]^2$$

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

For tabular Q(s,a)

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

Q-learning as MSE minimization

Given <**s**,**a**,**r**,**s'**> minimize

$$L=[Q(s_t,a_t)-Q^{true}(s_t,a_t)]^2$$

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

For tabular Q(s,a)

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

Something's sooo wrong!

Q-learning as MSE minimization

Given <**s**,**a**,**r**,**s'**> minimize

$$L = [Q(s_t, a_t) - \underbrace{Q^{true}(s_t, a_t)}_{\text{const}}]^2$$

$$L \approx [Q(s_t, a_t) - \underbrace{(r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a'))}_{\text{const}}]^2$$

For tabular Q(s,a)

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

Q-learning as MSE minimization

For tabular Q(s,a)

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

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

Gradient descent step:

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

Q-learning as MSE minimization

For tabular Q(s,a)

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

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

Gradient descent step:

$$Q(s, a) := Q(s, a)(1 - 2\alpha) + 2\alpha(r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a'))$$

Q-learning as MSE minimization

For tabular Q(s,a)

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

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

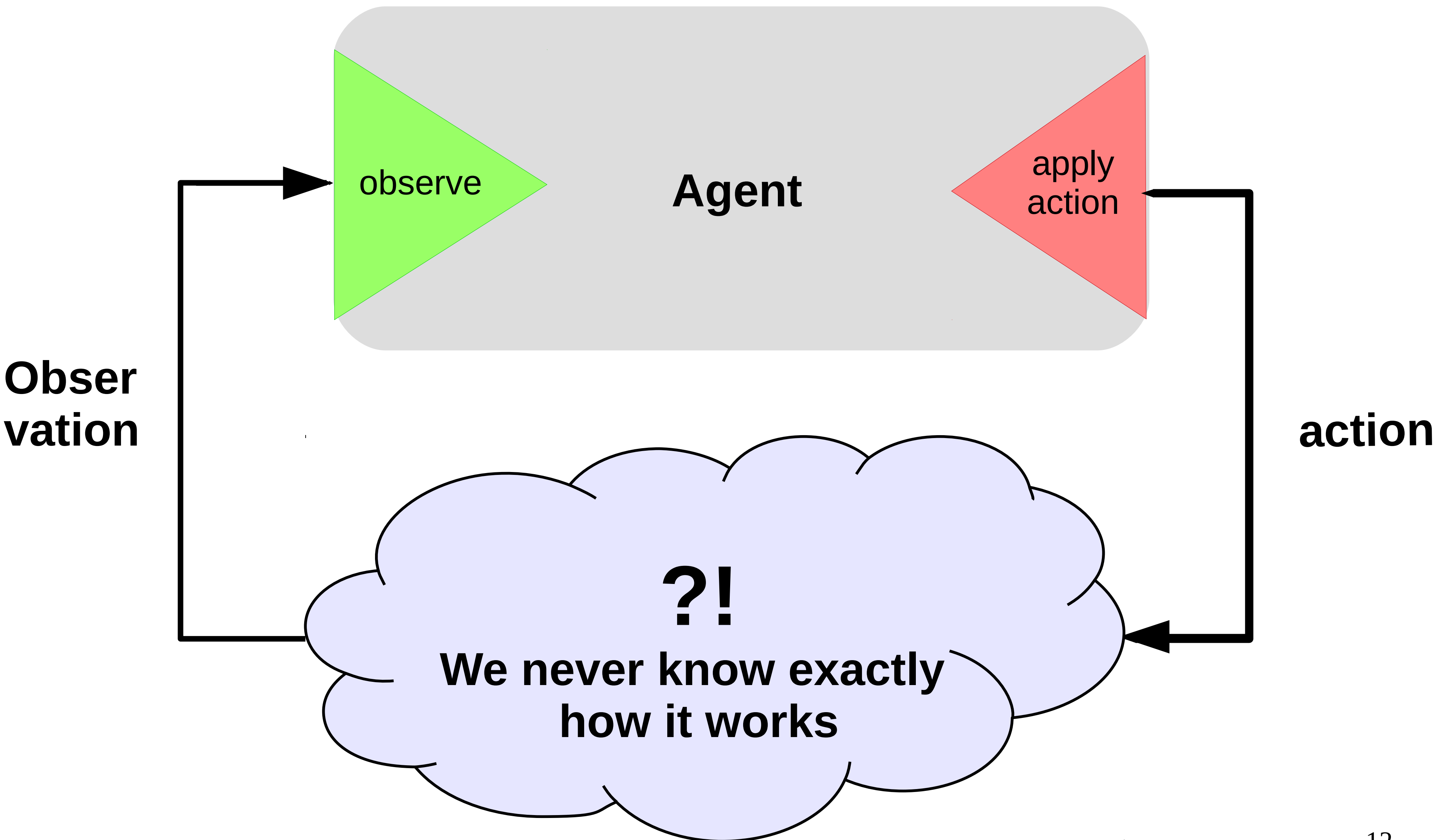
Gradient descent step:

$$Q(s,a) := Q(s,a)(1 - 2\alpha) + 2\alpha(r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a'))$$

= moving average formula

(define alpha' = 2*alpha)

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.

Problem:

State space is usually large,

sometimes continuous.

And so is action space;

Two solutions:

- Binarize state space
- Approximate agent with a function

Which one would you prefer for atari?

Problem:

State space is usually large,

sometimes continuous.

And so is action space;

Two solutions:

- Binarize state space ← **Too many bins or handcrafted features**
- Approximate agent with a function ← **Let's pick this one**

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

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

Trivia: should we use **classification** or **regression** model?
(e.g. logistic regression Vs linear regression)

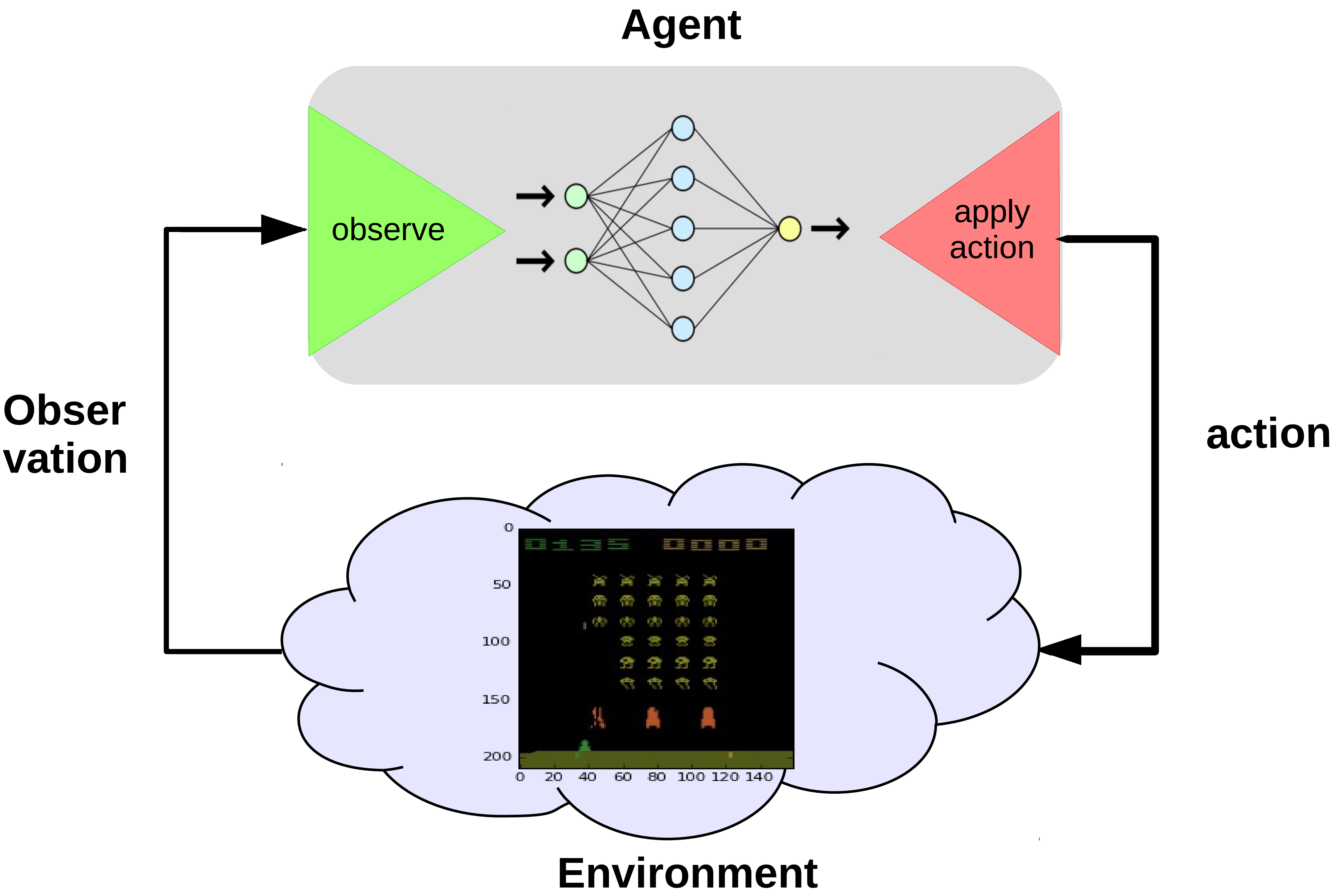
From tables to approximations

- 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

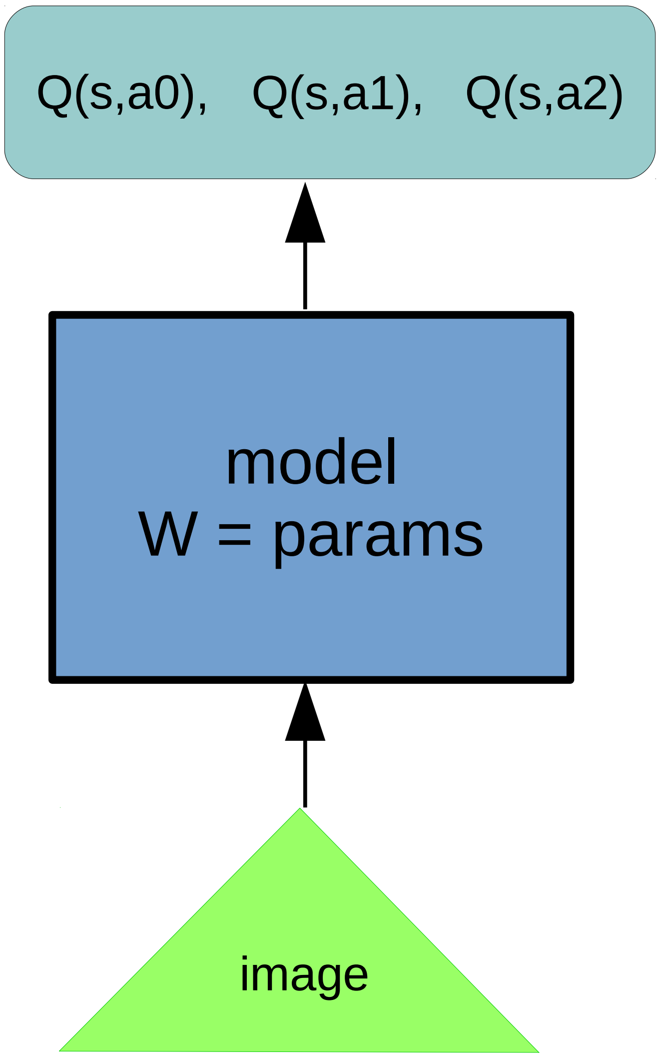
$$argmin_{w,b}\left(Q(s_t, a_t) - \left[r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a')\right]\right)^2$$

- Solve it as a **regression** problem!

MDP again


Agent
observe
apply
action
Obser
vation
action
0
50
100
150
200
0 20 40 60 80 100 120 140
Environment

Approximate Q-learning



Q-values:

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

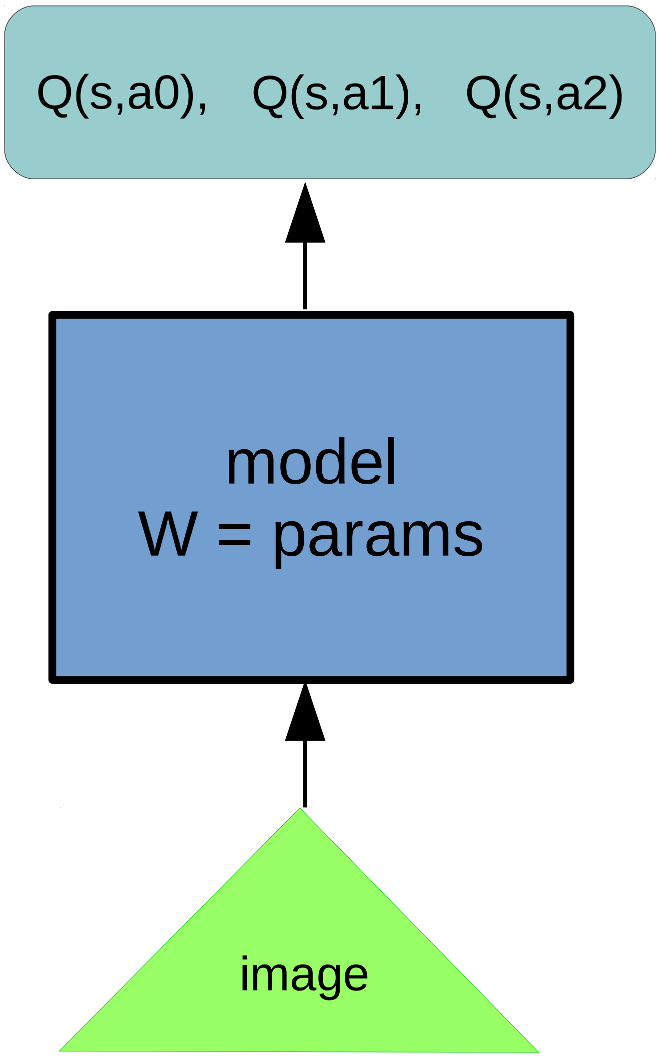
Objective:

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

Gradient step:

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

Approximate Q-learning



Q-values:

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

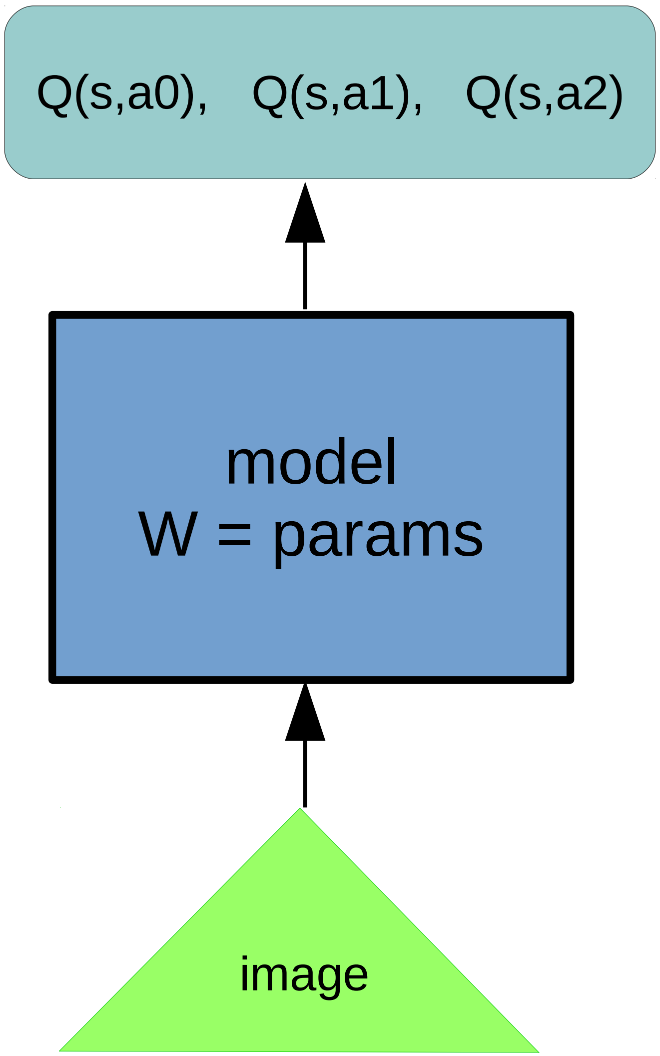
Objective:

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

Gradient step:

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

Approximate SARSA



Objective:

$$L = \left(Q(s_t, a_t) - \underbrace{\hat{Q}(s_t, a_t)}_{\text{consider const}}\right)^2$$

Q-learning:

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

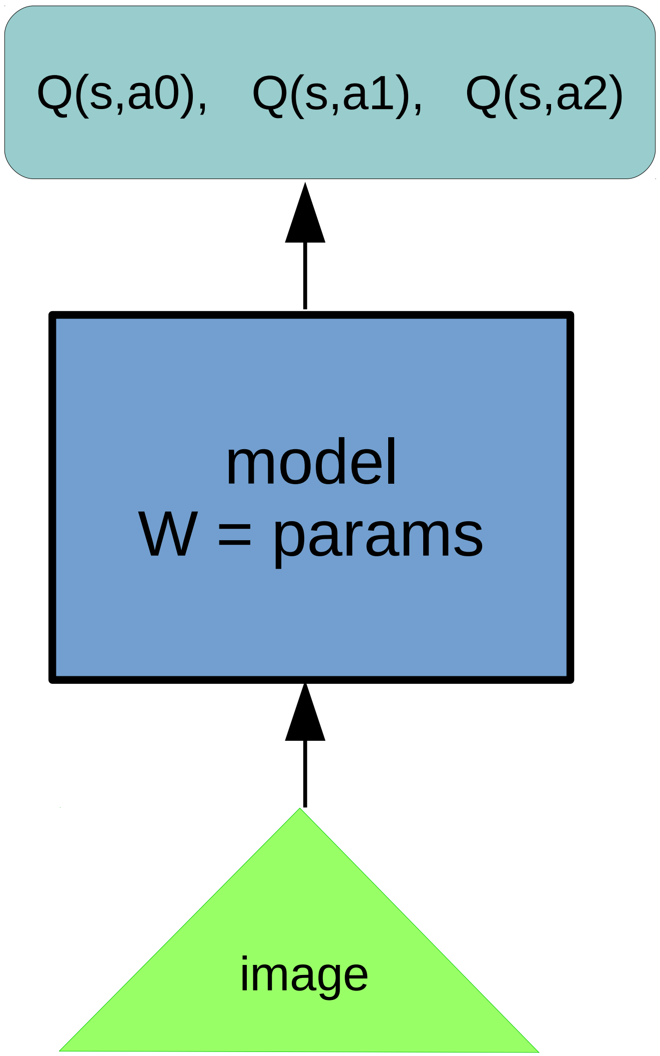
SARSA:

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot Q(s_{t+1}, a_{t+1})$$

Expected Value SARSA:

$\hat{Q}(s_t, a_t) =$ **???**

Approximate SARSA



Objective:

$$L = \left(Q(s_t, a_t) - \underbrace{\hat{Q}(s_t, a_t)}_{\text{consider const}}\right)^2$$

Q-learning:

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

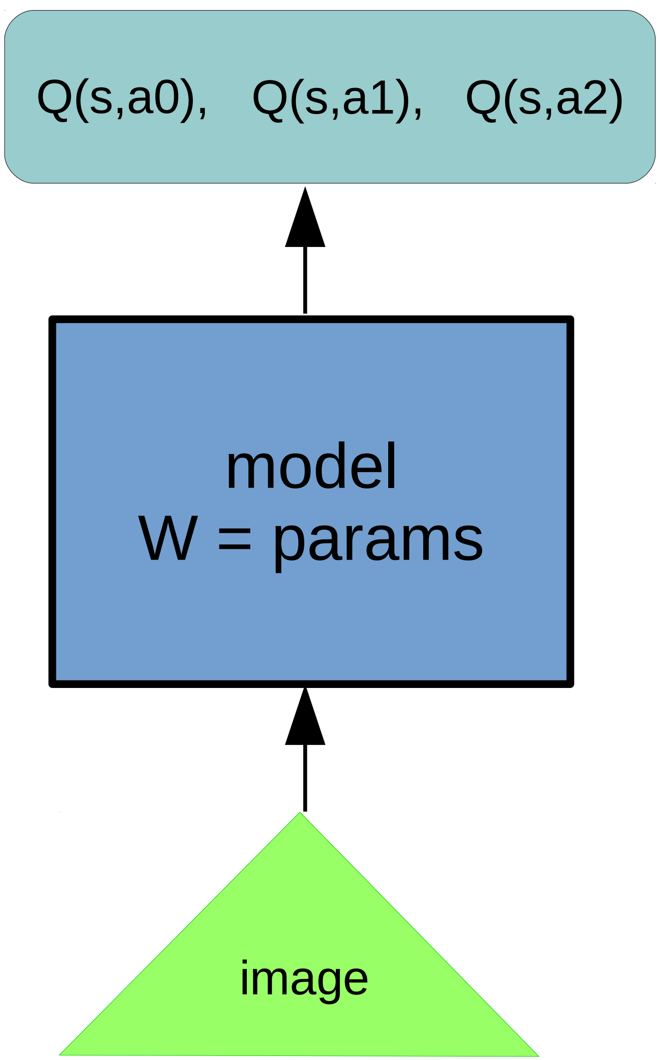
SARSA:

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot Q(s_{t+1}, a_{t+1})$$

Expected Value SARSA:

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \underset{a' \sim \pi(a|s)}{E} Q(s_{t+1}, a')$$

Approximate n-step methods



Objective:

$$L = \left(Q(s_t, a_t) - \underbrace{\hat{Q}(s_t, a_t)}_{\text{consider const}}\right)^2$$

Q-learning:

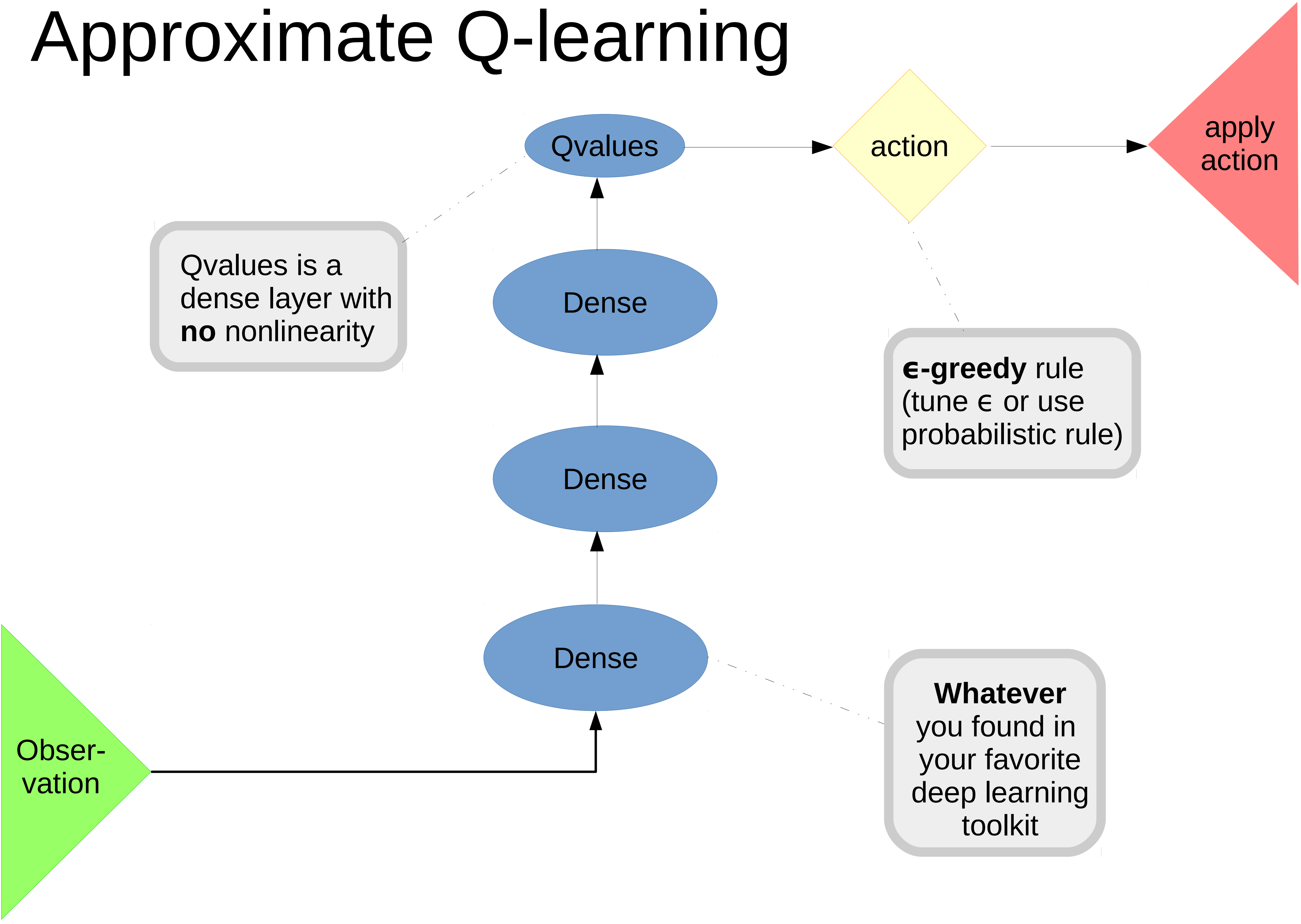
$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \max_{a'} Q(s_{t+1}, a')$$

Q-learning n-step:

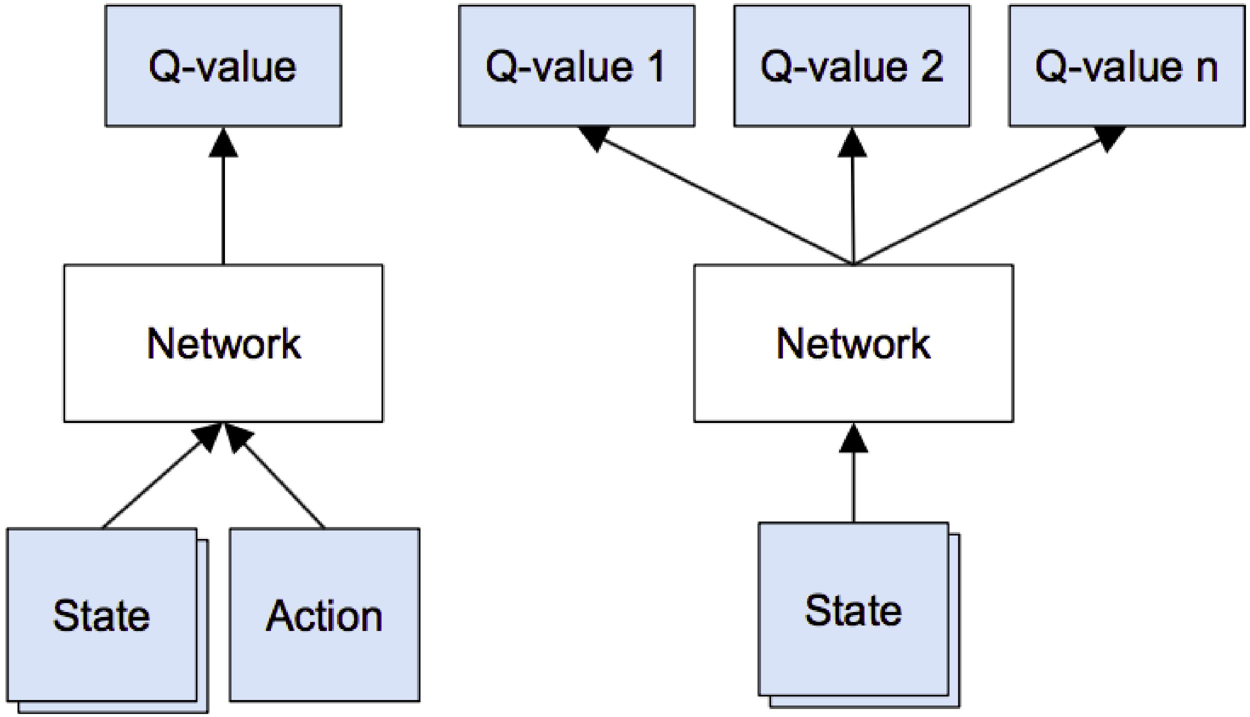
$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 Q(s_{t+2}, a_{t+2})$$

$$\hat{Q}(s_t, a_t) = \left[\sum_{\tau=t}^{\tau<t+n} \gamma^{\tau} r(s_{t+\tau}, a_{t+\tau})\right] + \gamma^n \cdot \max_{a} Q(s_{t+n}, a)$$

Approximate Q-learning



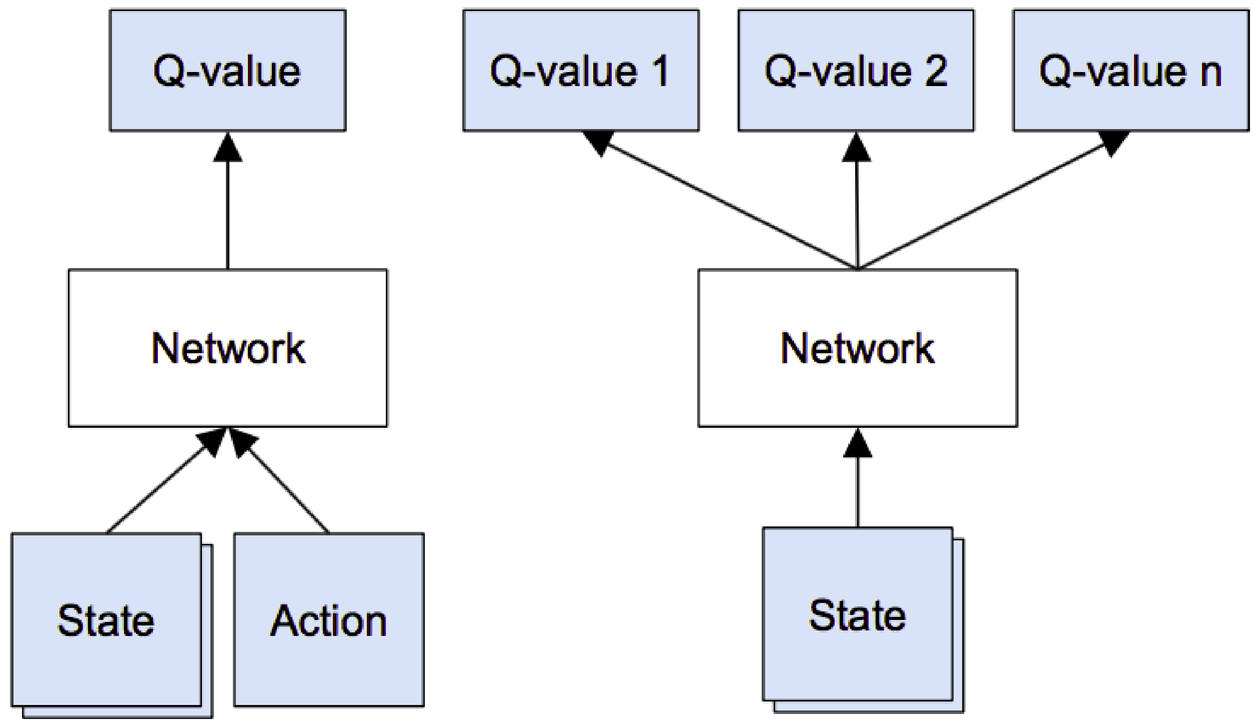
Architectures



Given **(s,a)**
Predict Q(s,a)

Given **s** predict all q-values
Q(s,a0), Q(s,a1), Q(s,a2)

Architectures

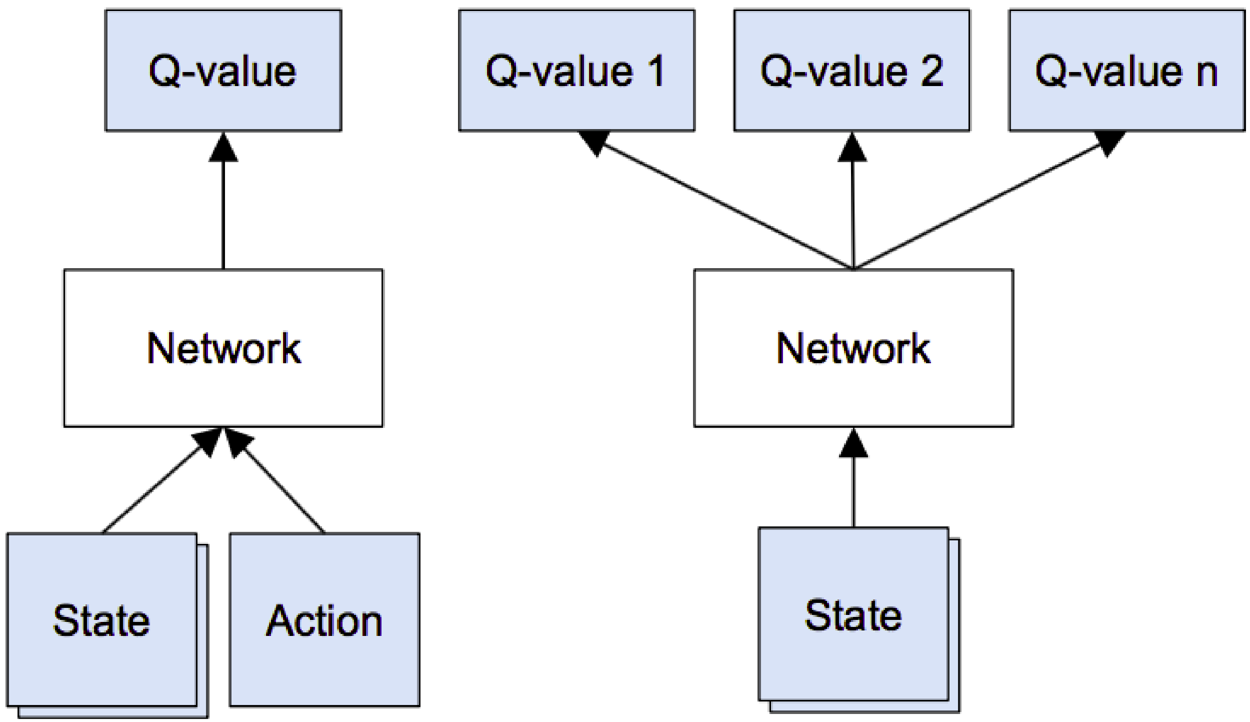


Given **(s,a)**
Predict Q(s,a)

Given **s** predict all q-values
Q(s,a0), Q(s,a1), Q(s,a2)

Trivia: in which situation does **left** model work better?
And right?

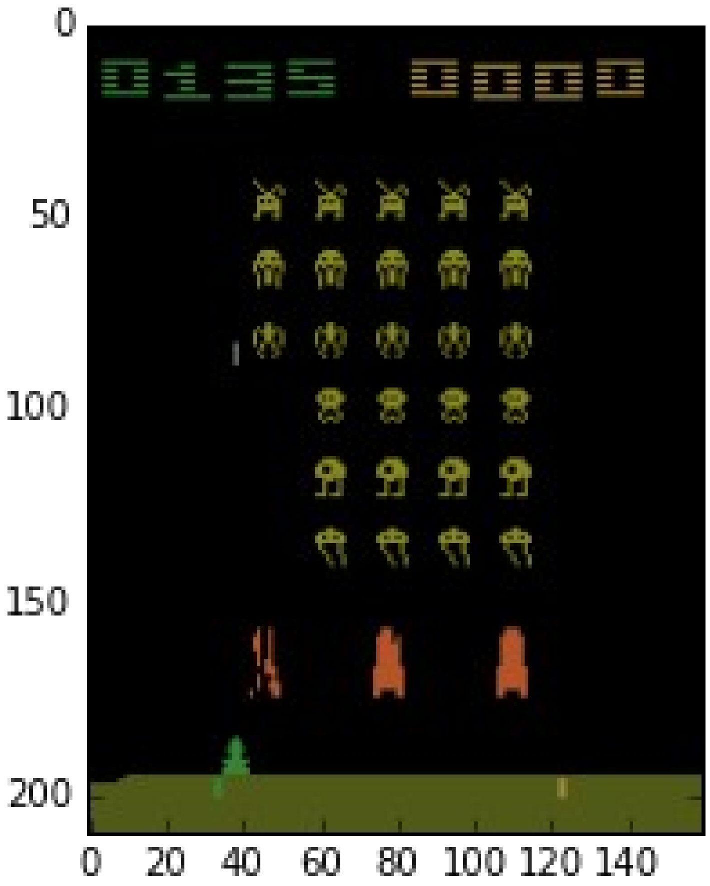
Architectures



Needs one forward pass
for **each action**

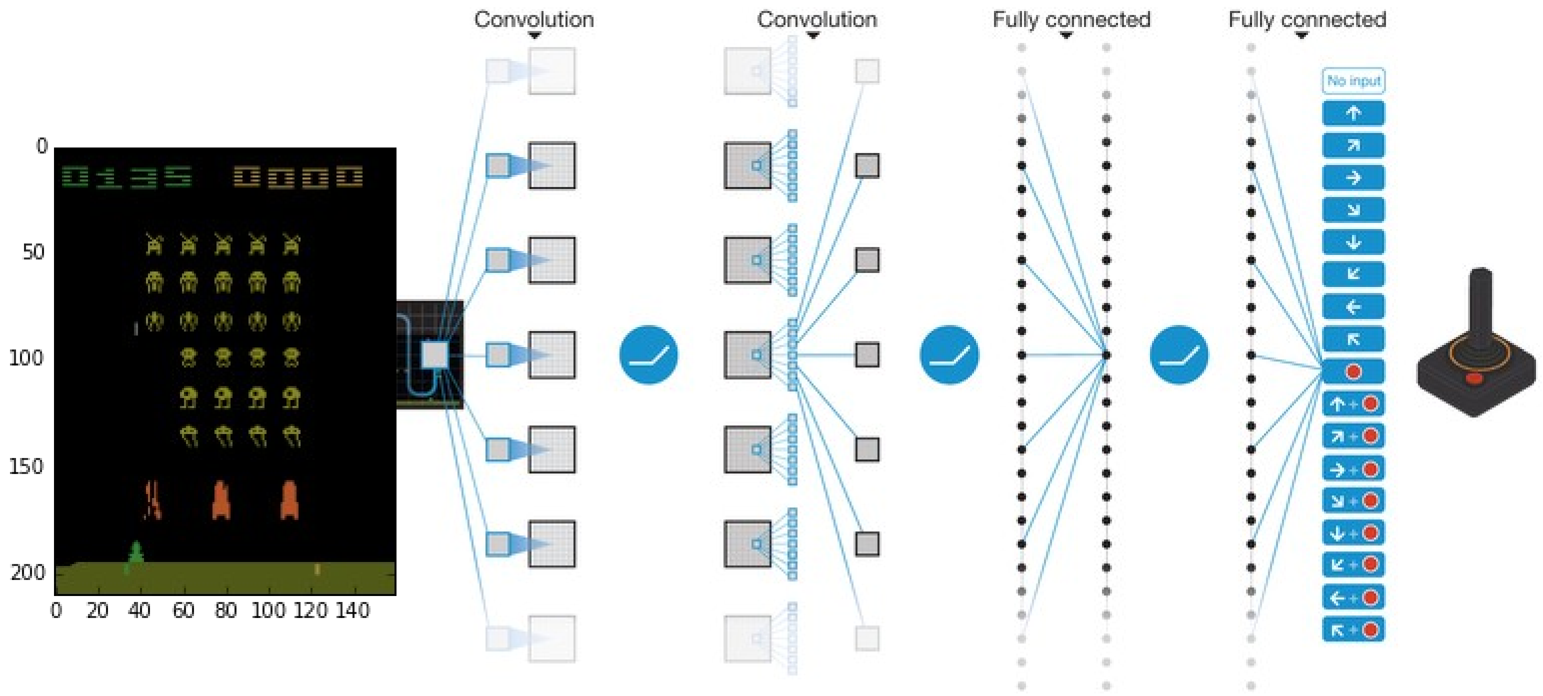
Works if action space is
large / continuous

Needs one forward pass
for **all actions**
(faster)

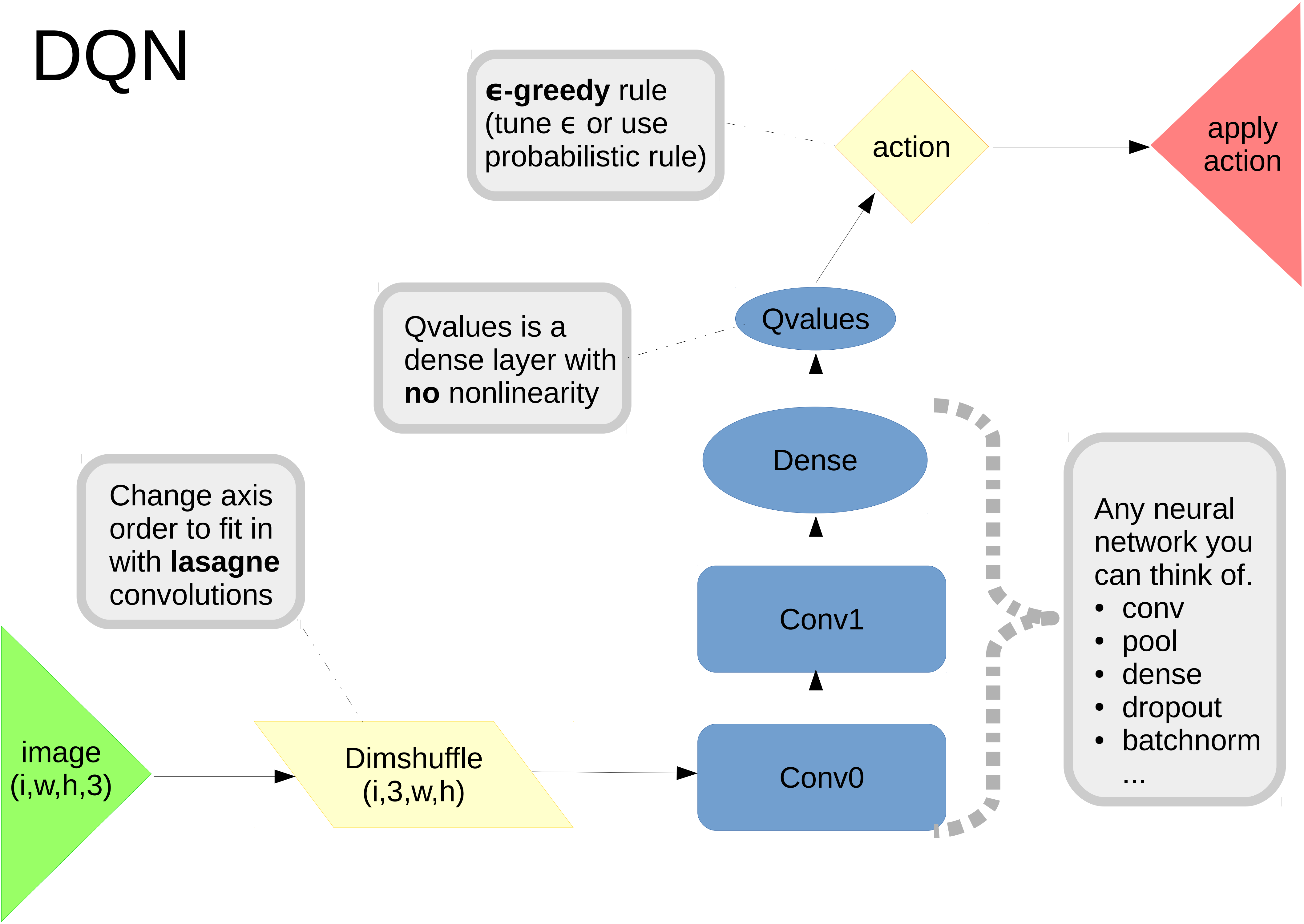


What kind of network digests images well?

Deep learning approach: DQN

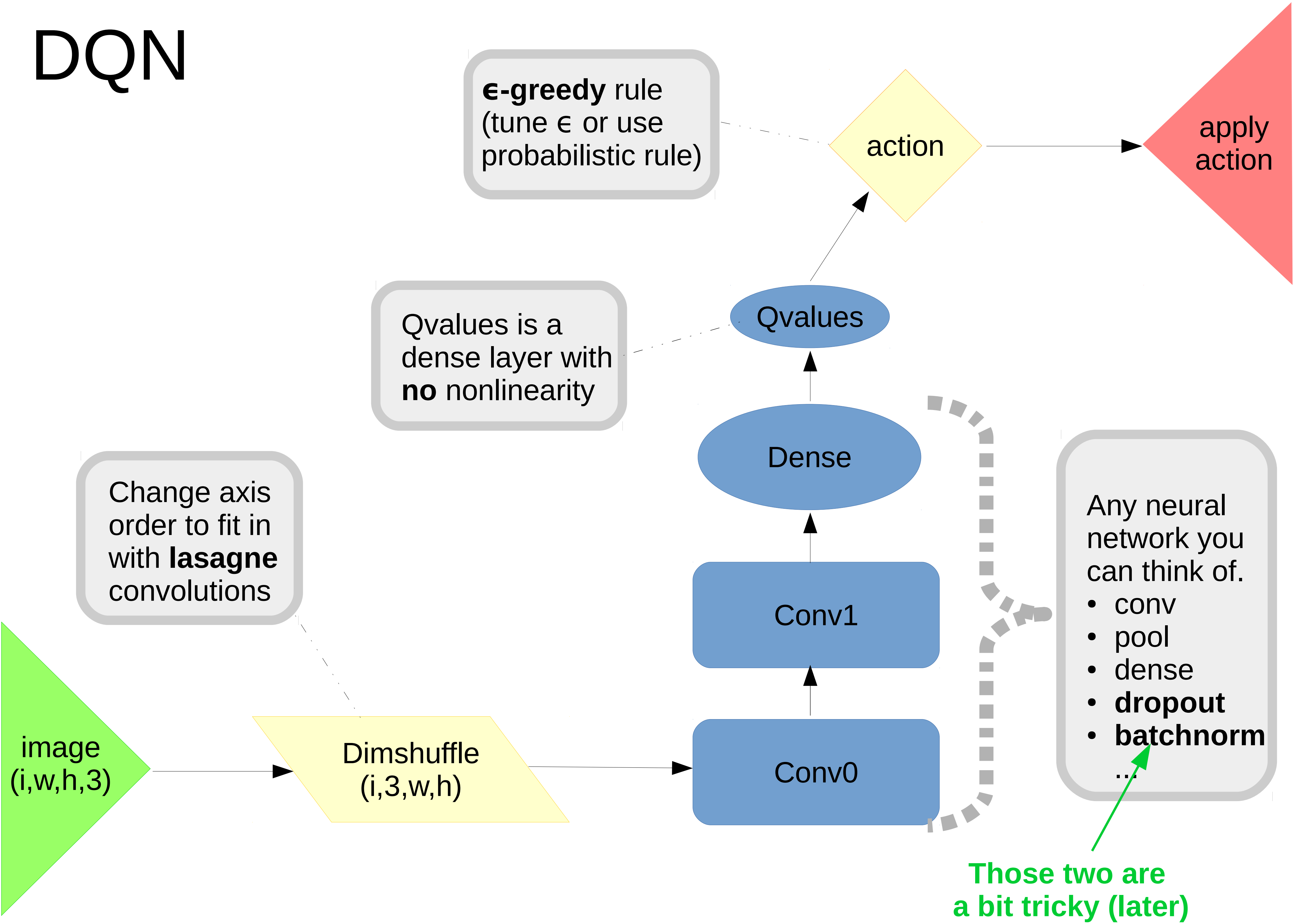


DQN



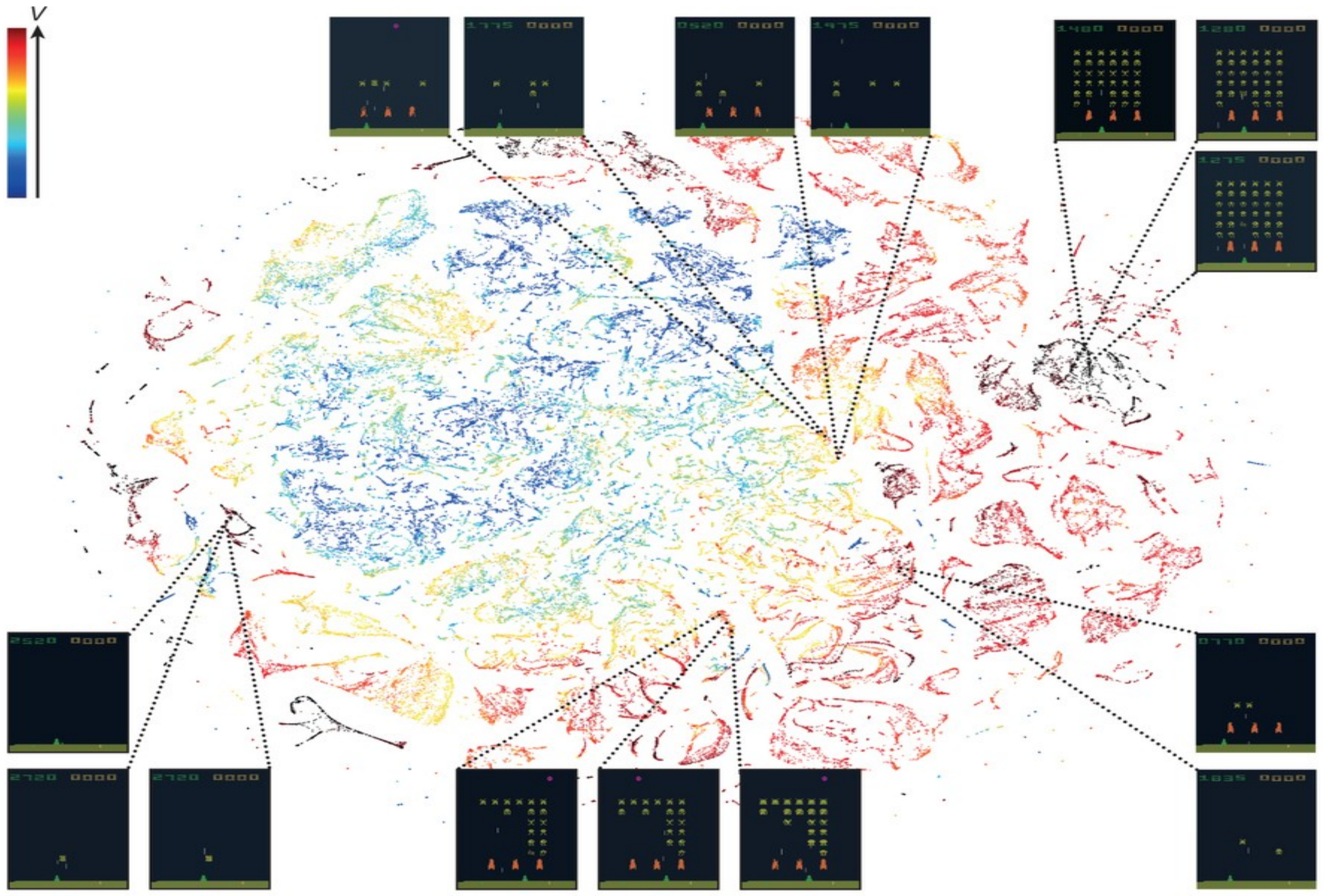
ϵ-greedy rule
(tune ϵ or use
probabilistic rule)
action
apply
action
Qvalues is a
dense layer with
no nonlinearity
Qvalues
Dense
Change axis
order to fit in
with **lasagne**
convolutions
Conv1
Any neural
network you
can think of.
• conv
• pool
• dense
• dropout
• batchnorm
...
image
(i,w,h,3)
Dimshuffle
(i,3,w,h)
Conv0

DQN

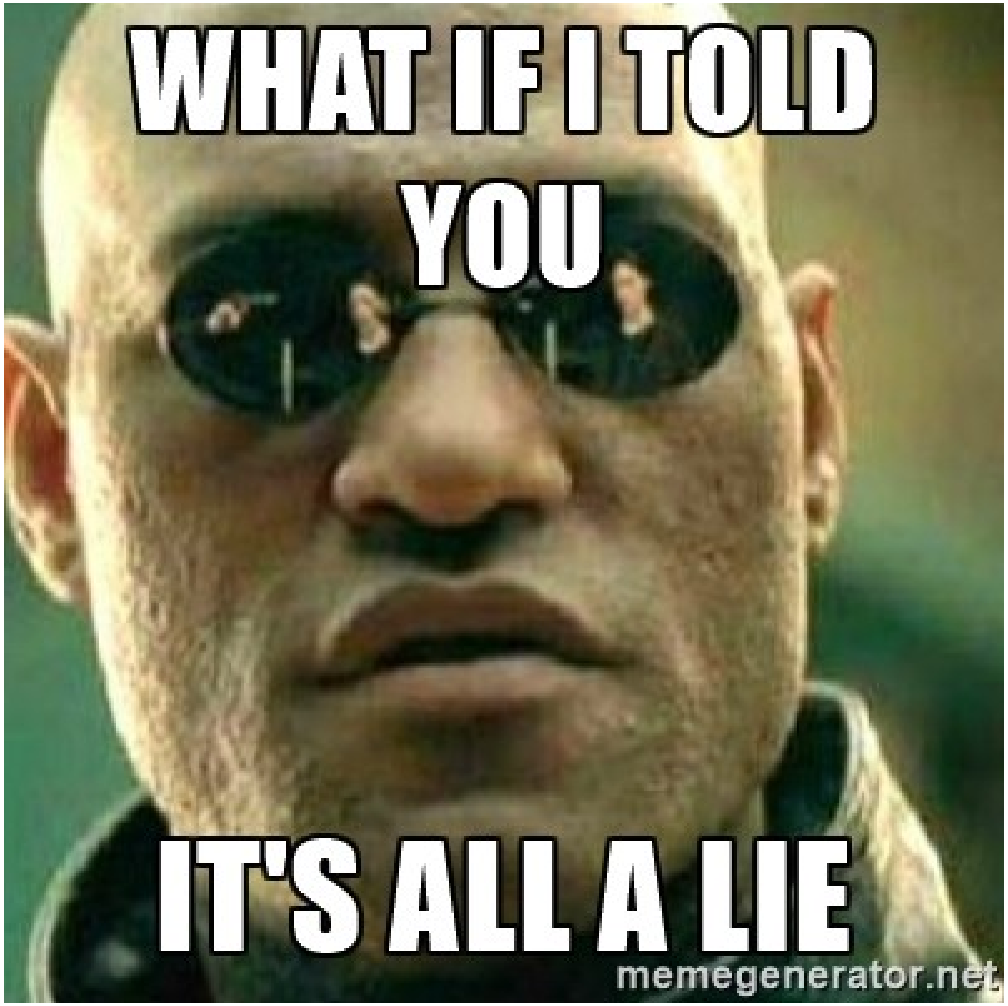


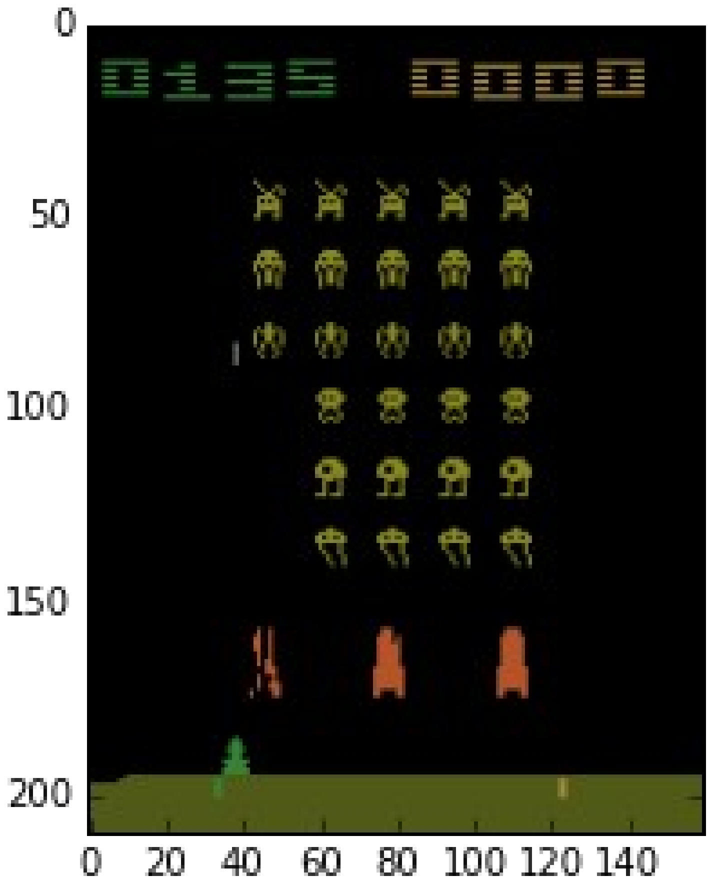
ϵ-greedy rule
(tune ϵ or use
probabilistic rule)
action
apply
action
Qvalues is a
dense layer with
no nonlinearity
Qvalues
Dense
Conv1
Conv0
Change axis
order to fit in
with **lasagne**
convolutions
image
(i,w,h,3)
Dimshuffle
(i,3,w,h)
Any neural
network you
can think of.
• conv
• pool
• dense
• **dropout**
• **batchnorm**
...
Those two are
a bit tricky (later)

TSNE makes every slide 40% better



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


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

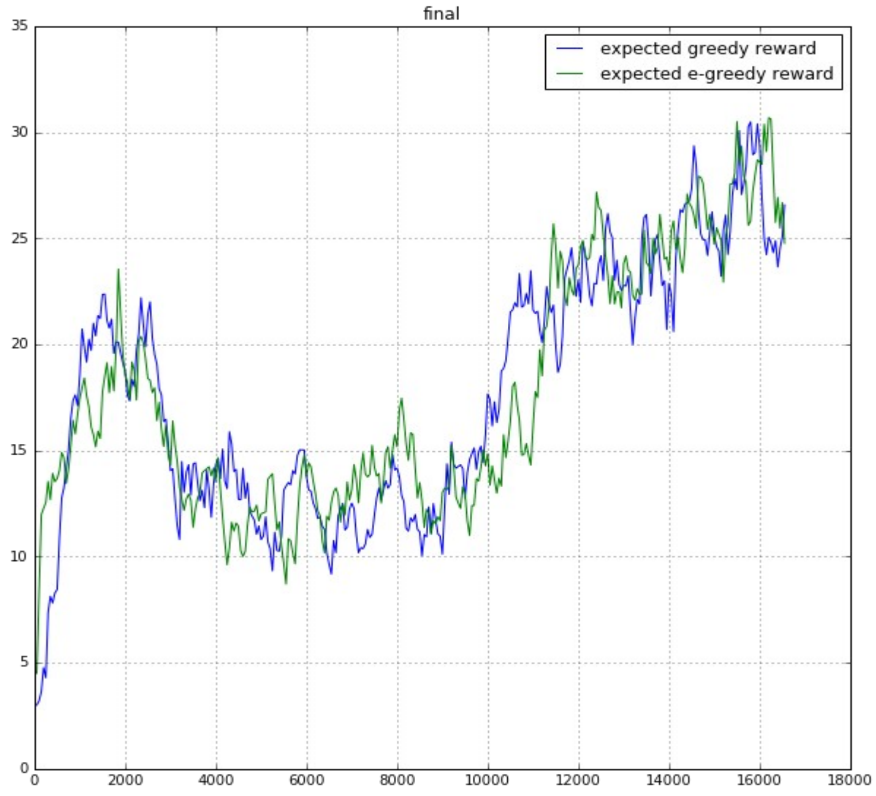


How bad it is if agent spends

next 1000 ticks under the left rock?
(while training)

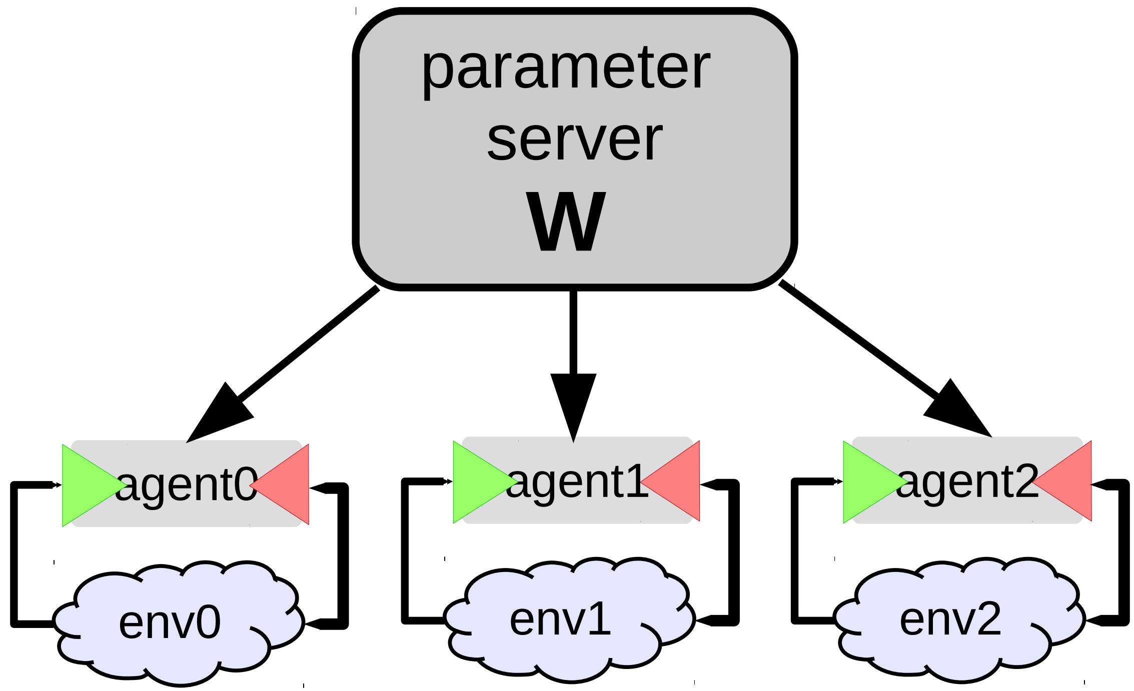
Problem

- 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?**



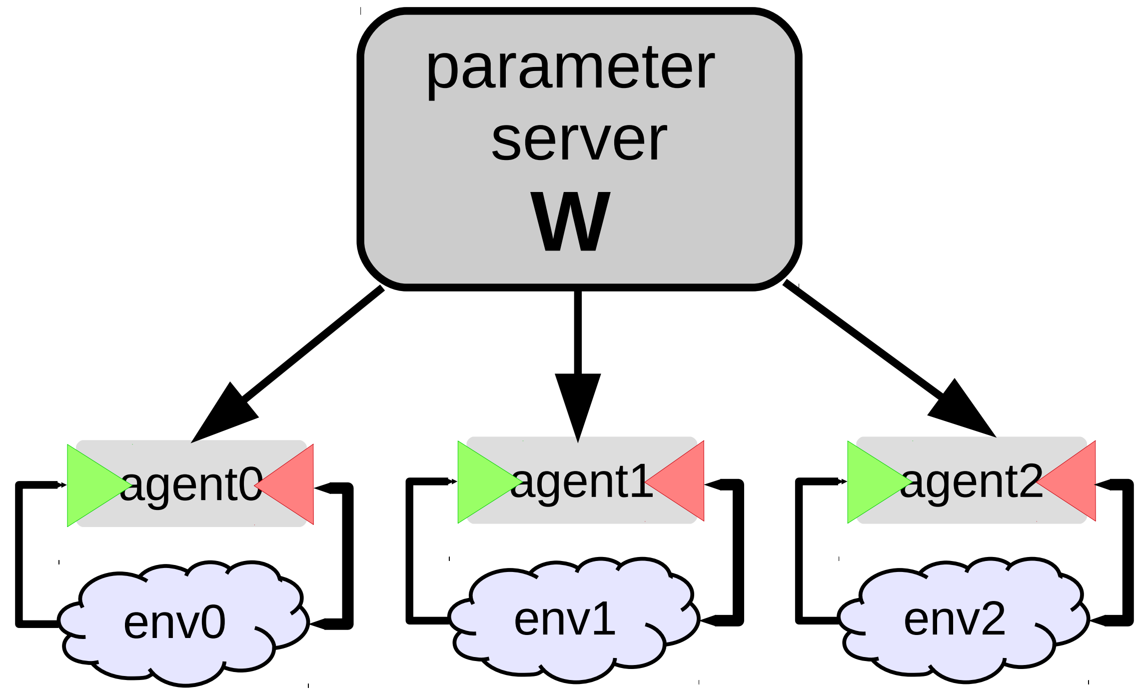
Multiple agent trick

Idea: Throw in several agents with shared **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?

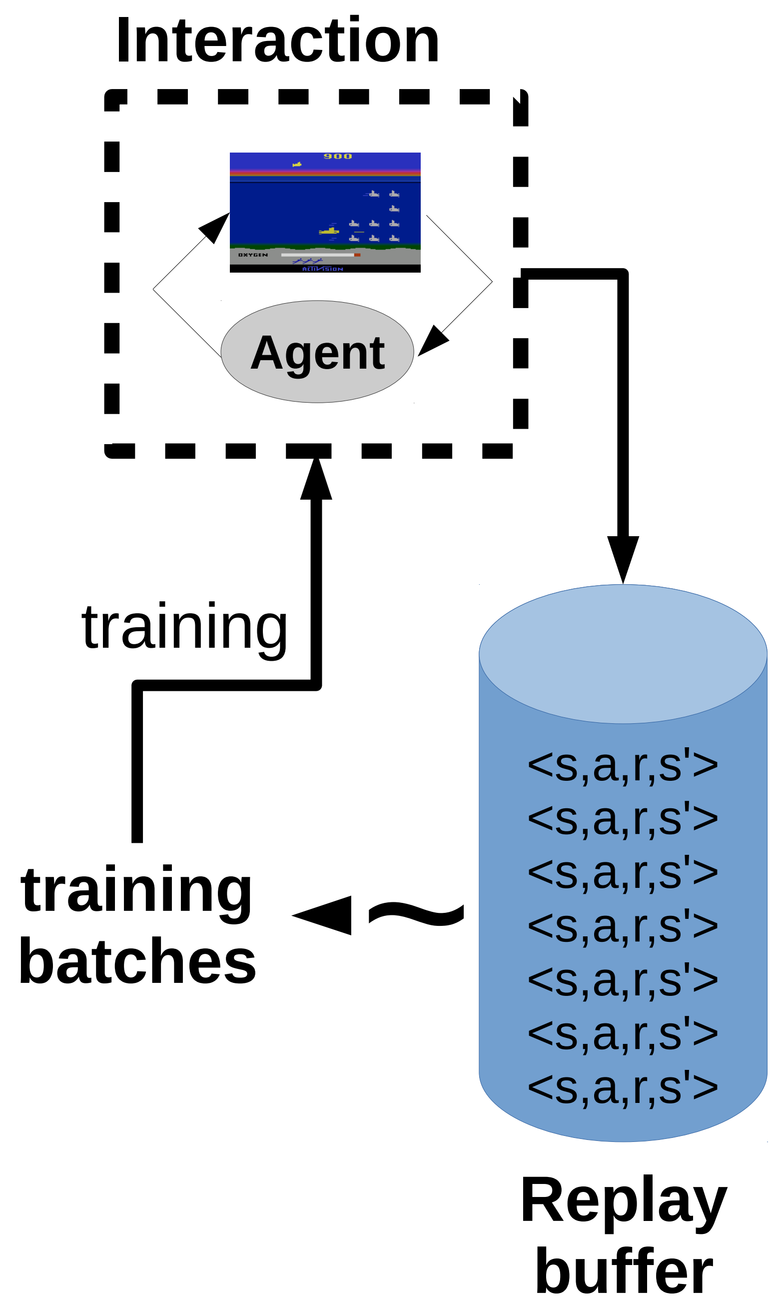
Experience replay

Idea: store several past interactions

<s,a,r,s'>

Train on random subsamples

Any +/- ?

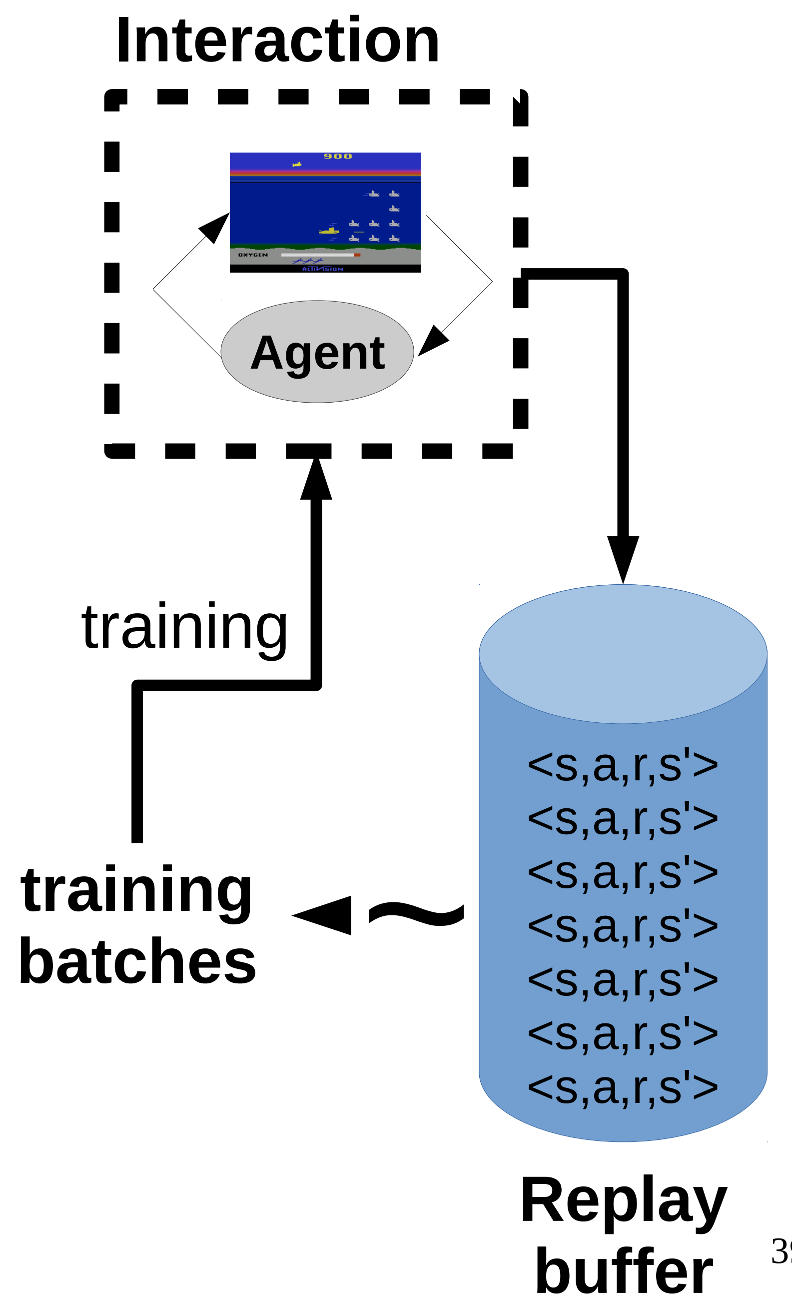


Experience replay

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

Better versions coming next week



Summary so far

to make data closer to i.i.d.

Use one or several of

- **experience replay**
- **multiple agents**
- Infinitely small learning rate :)

better tricks coming next week

An important question

- You approximate Q(s,a) with a neural network
- You use **experience replay** when training

Trivia: which of those algorithms will fail?

- Q-learning
- SARSA
- 15-step q-learning
- Expected Value SARSA

An important question

- You approximate Q(s,a) with a neural network
- You use **experience replay** when training

Agent trains off-policy on an older version of him

Trivia: which of those algorithms will fail?

Off-policy methods work, On-policy is super-slow (fail)

- Q-learning
- SARSA
- 15-step q-learning
- Expected Value SARSA

When training with on-policy methods,

– use no (or small) experience replay

– compensate with parallel game sessions

Deep learning meets MDP

- Dropout, noize
  - **Used in experience replay only:** like the usual dropout
  - **Used when interacting:** a special kind of exploration
  - You may want to decrease **p** over time.

- Batchnorm
  - Faster training but may break moving average
  - **Experience replay:** may break down if buffer is too small
  - **Parallel agents:** may break down under too few agents

<same problem of being non i.i.d.>

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:

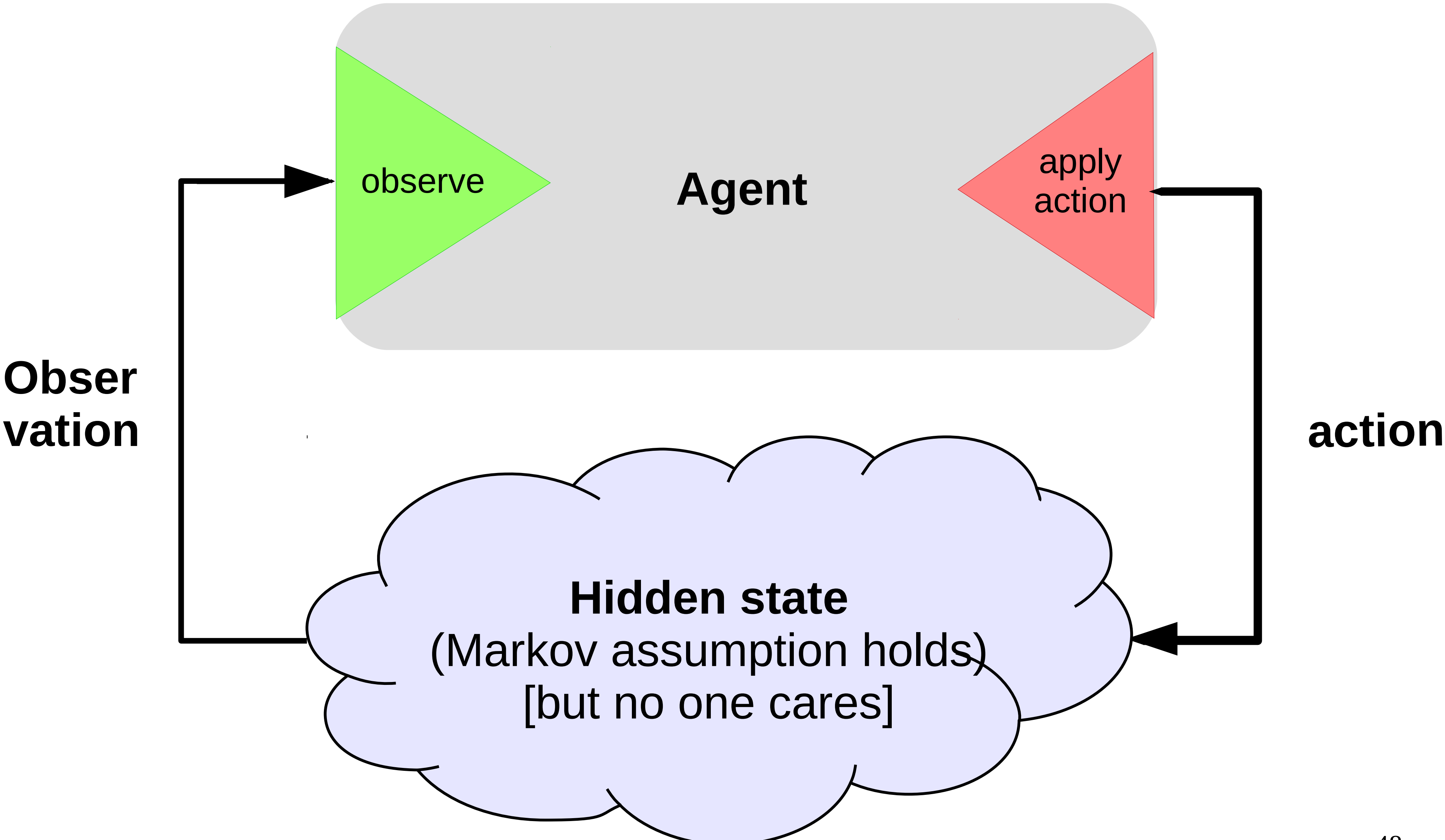
Agent observation does not hold all information about process state (e.g. human field of view).

- 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



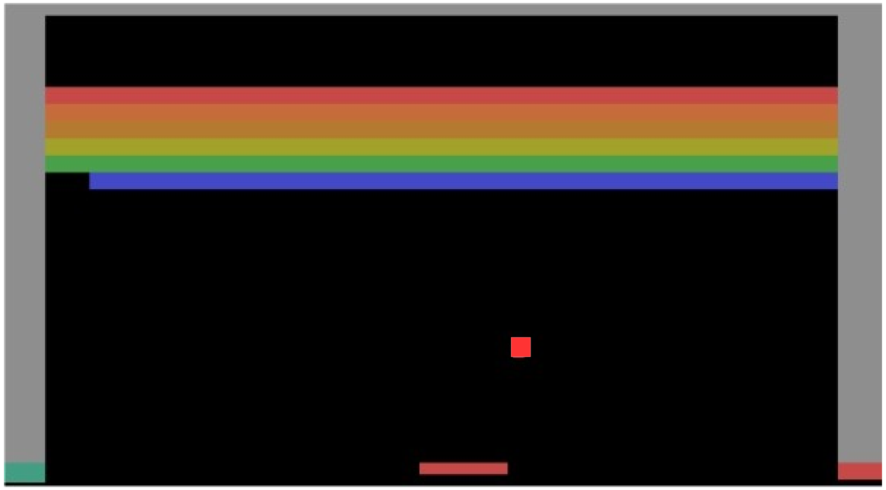
N-gram heuristic

Idea:

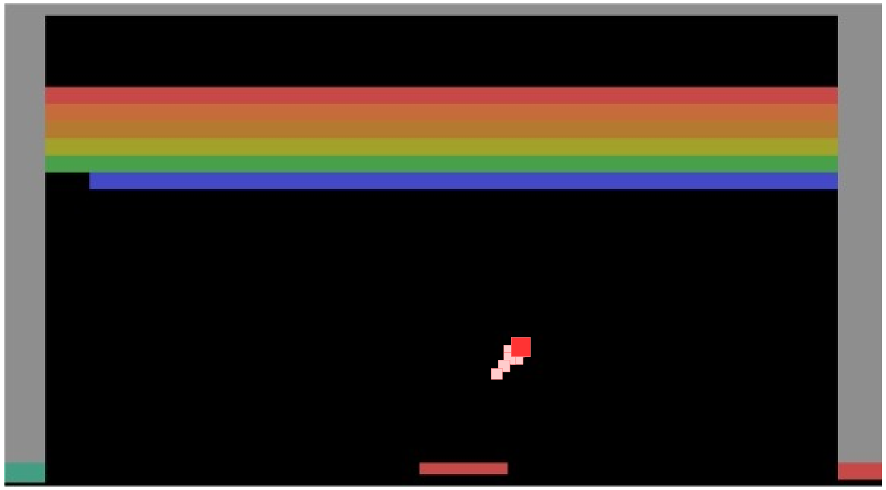
$$s_t \neq o(s_t)$$

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

e.g. ball movement in breakout

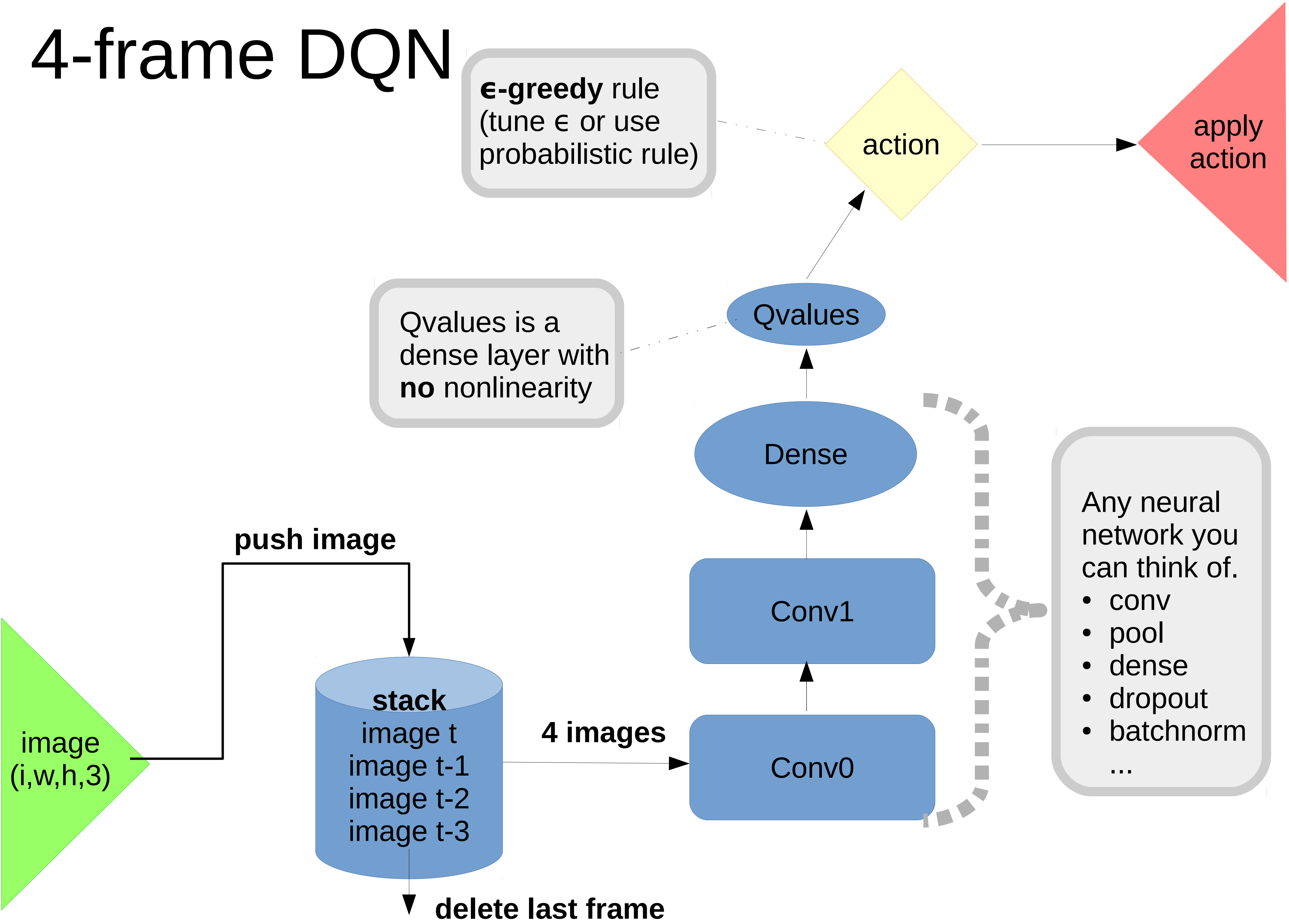


- One frame



- 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
delete last frame

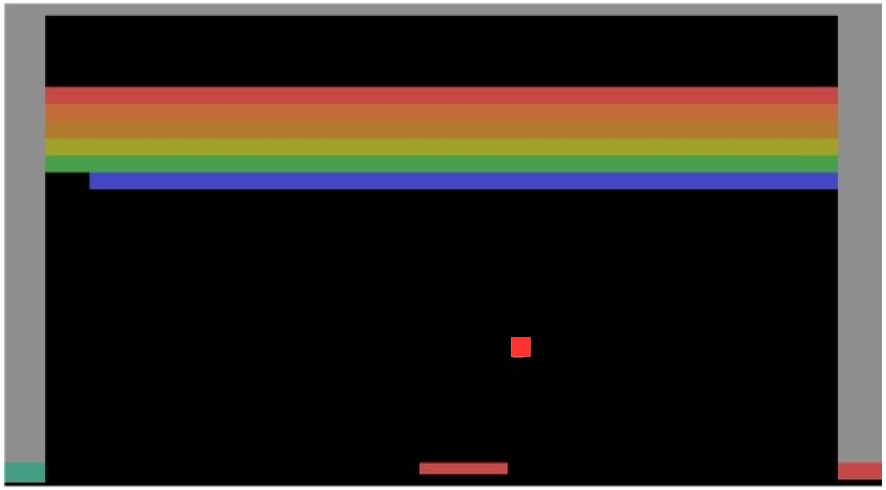
N-gram heuristic

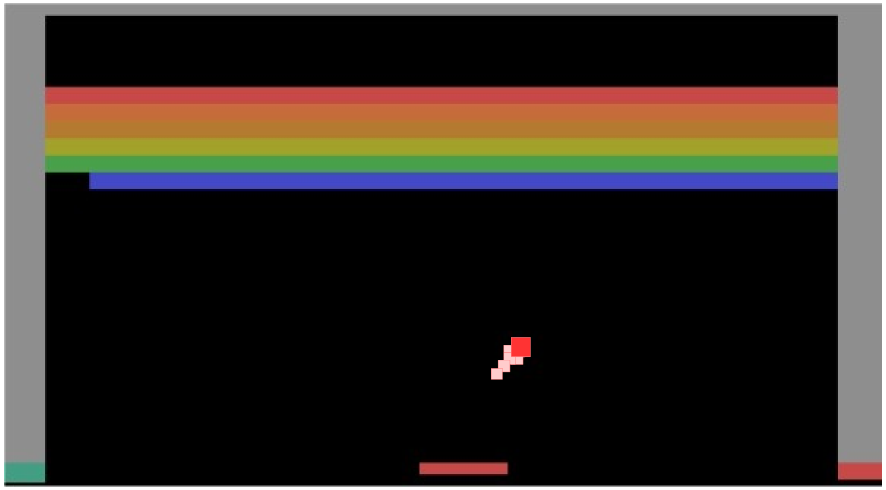
Idea:

$$s_t \neq o(s_t)$$

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

e.g. ball movement in breakout





- One frame
- Several frames

Alternatives

Ngrams:

- Nth-order markov assumption
- Works for velocity/timers
- Fails for anything longer that 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

Seminar

Autocorrelation

- Reference is based on predictions

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

- 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

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

- Update Q old periodically
  - Slows down training

Target networks

Idea: use older network snapshot
to compute reference

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

- 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