Lecture 11: Deep Reinforcement Learning

Radoslav Neychev

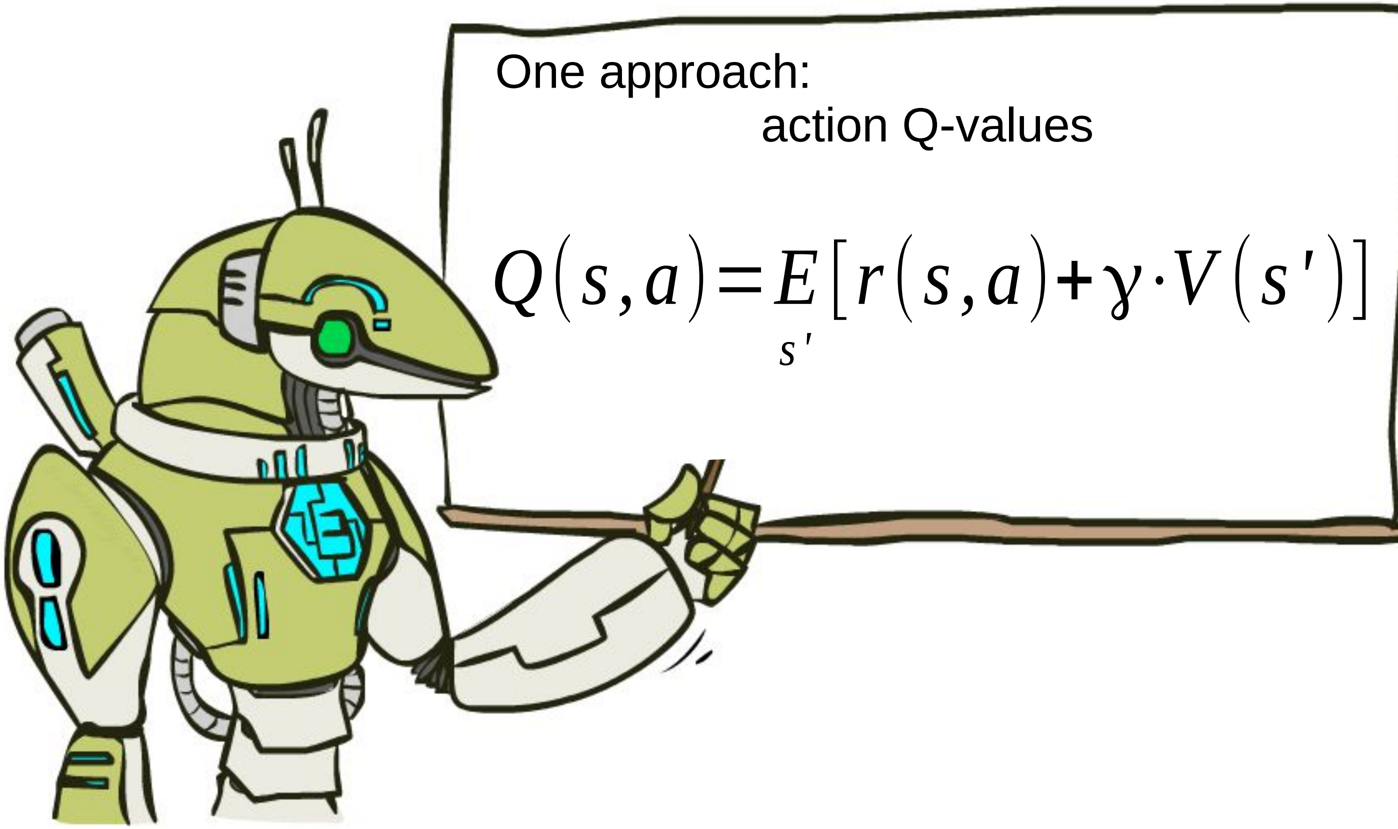
Harbour.Space University
22.07.2019, Barcelona, Spain

References

These slides are almost the exact copy of Practical RL course week 4 slides.
Special thanks to YSDA team for making them publicly available.

Original slides link: week04_approx_rl

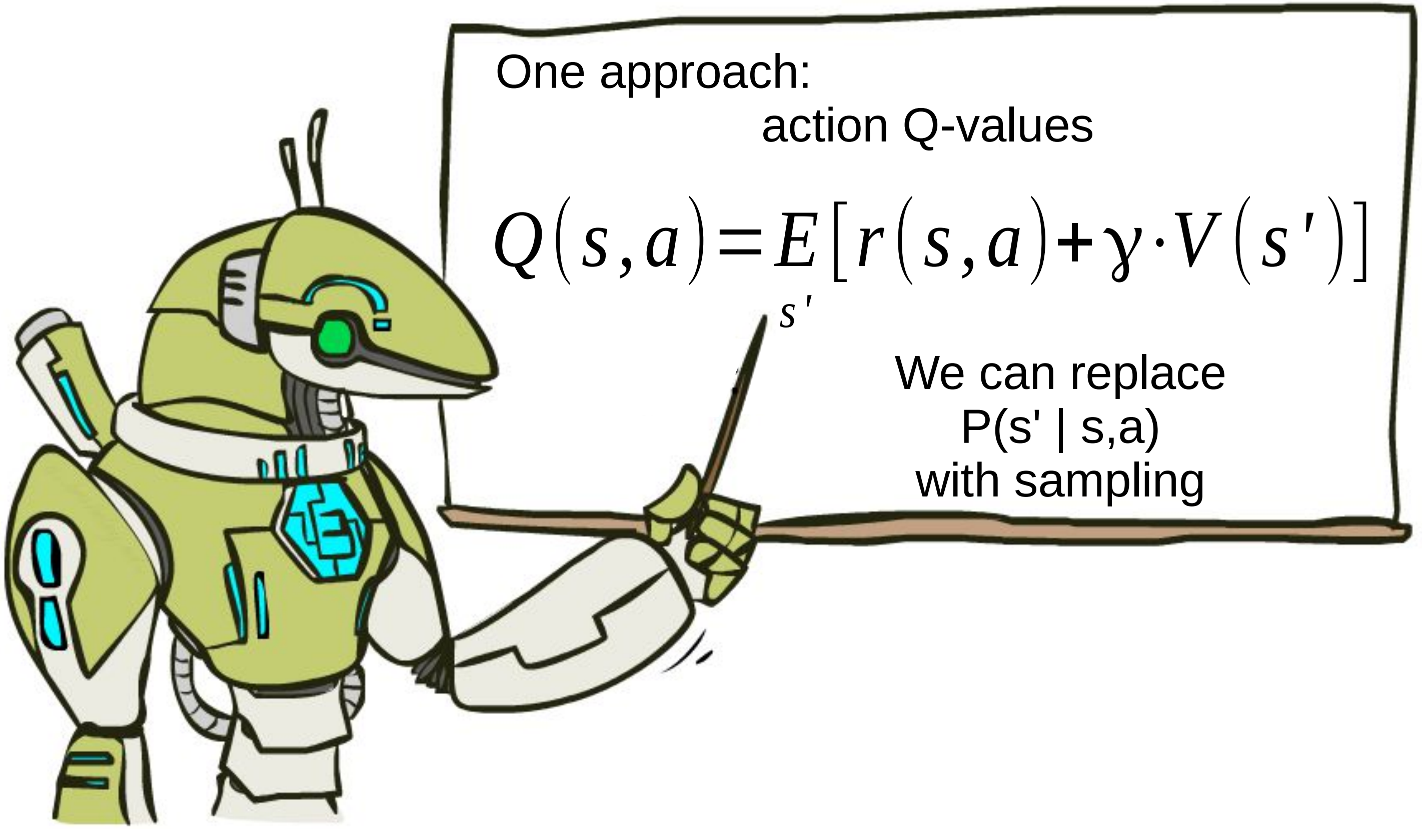
Recap: Q-learning



Action value Q(s,a) is the expected total reward **G** 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

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

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

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)

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

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

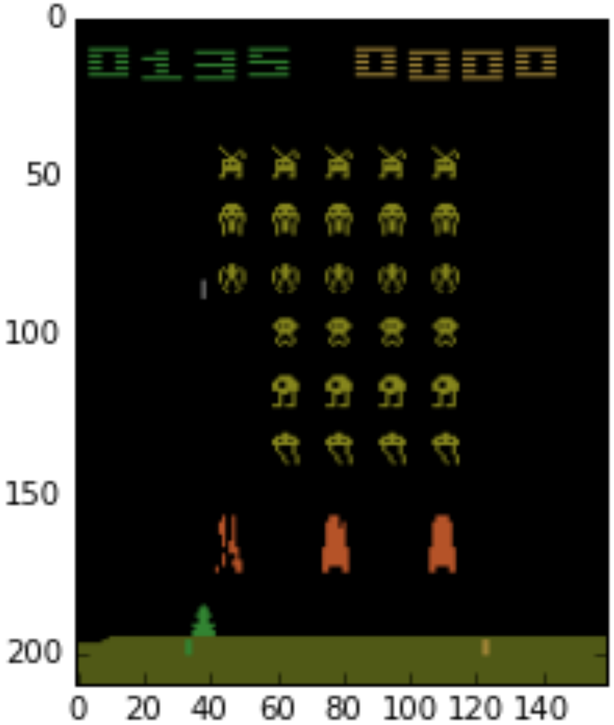
Gradient descent step:

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

= moving average formula

(define alpha' = 2*alpha)

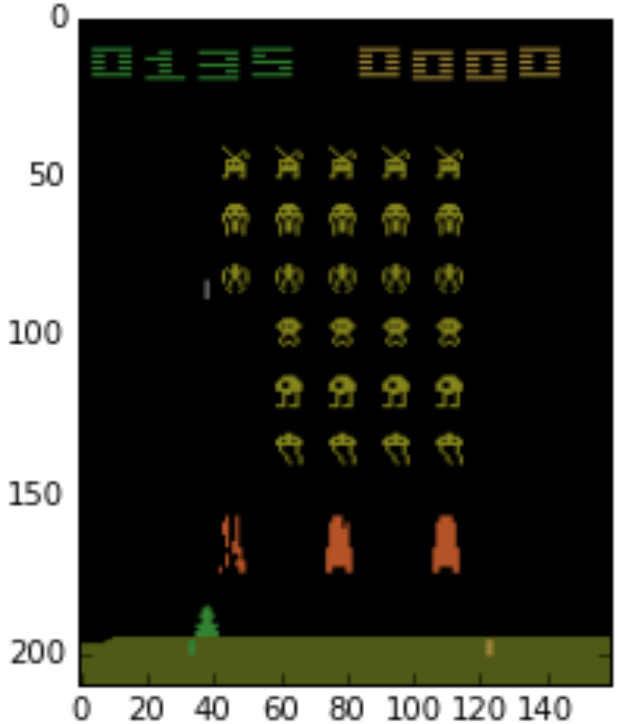
Real world



How many states are there?

approximately

Real world



$$|S| \approx 2^{8 \cdot 210 \cdot 160} = 729179546432630\ldots$$

80917 digits :)

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 (last week)
- Approximate agent with a function (crossentropy method)

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

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

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

From tables to approximations

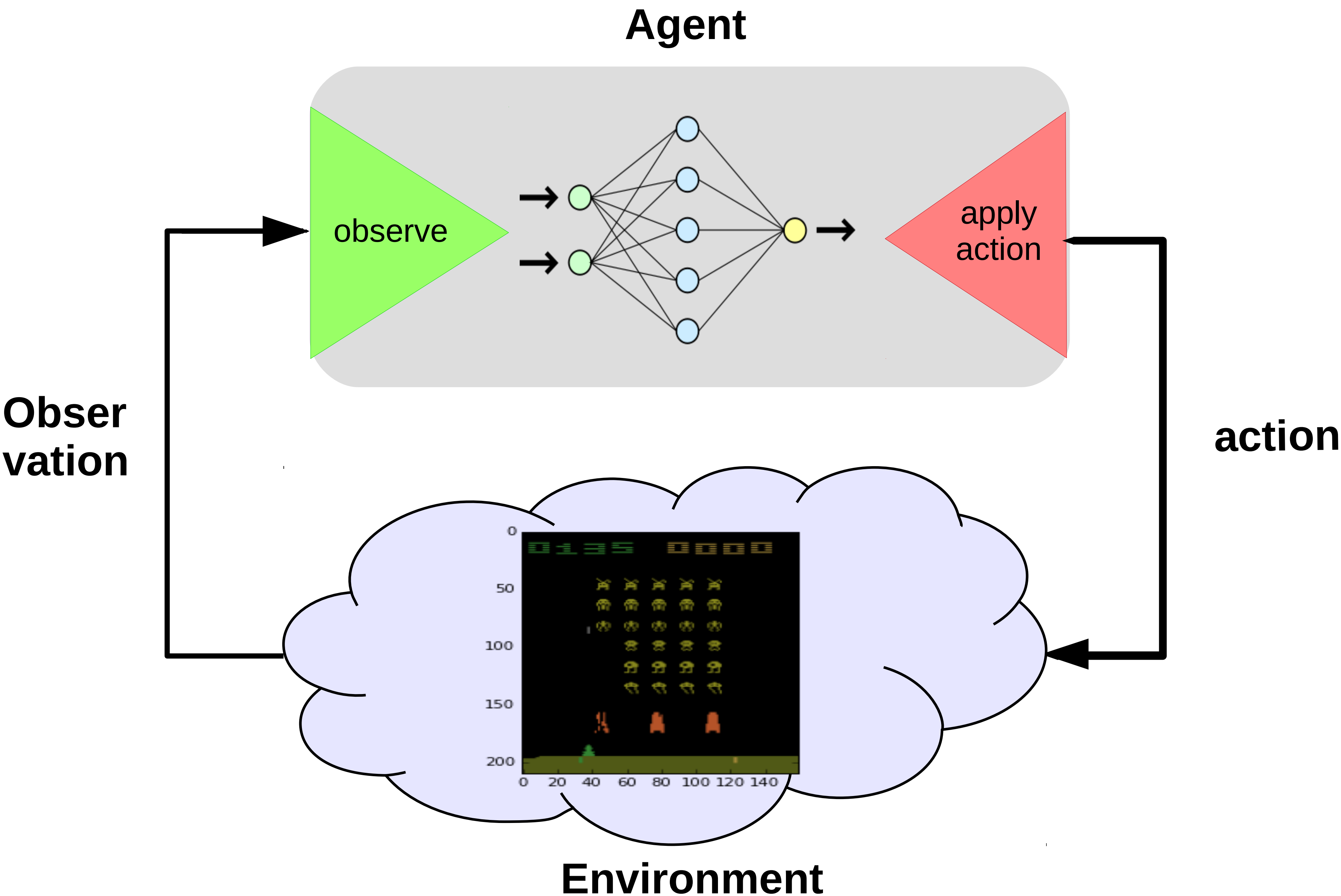
- Before:
  - For all states, for all actions, remember Q(s,a)

- Now:
  - Approximate Q(s,a) with some function
  - e.g. linear model over state features

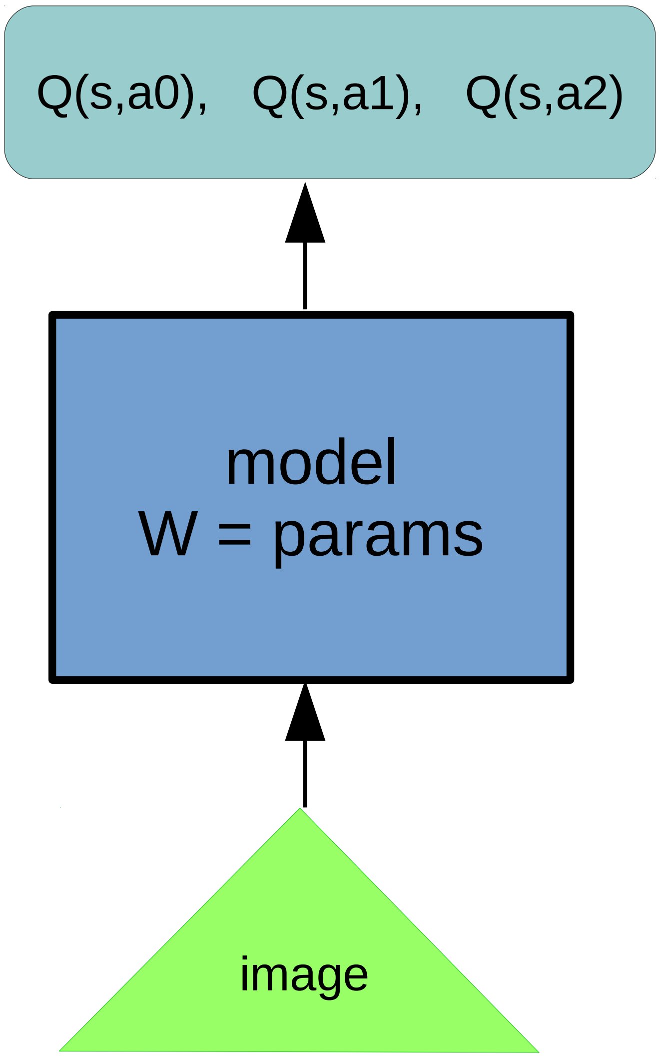
$$argmin_{w,b}\left(Q\left(s_t, a_t\right) - \left[r_t + \gamma \cdot max_{a'} Q\left(s_{t+1}, a'\right)\right]\right)^2$$

- 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:

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

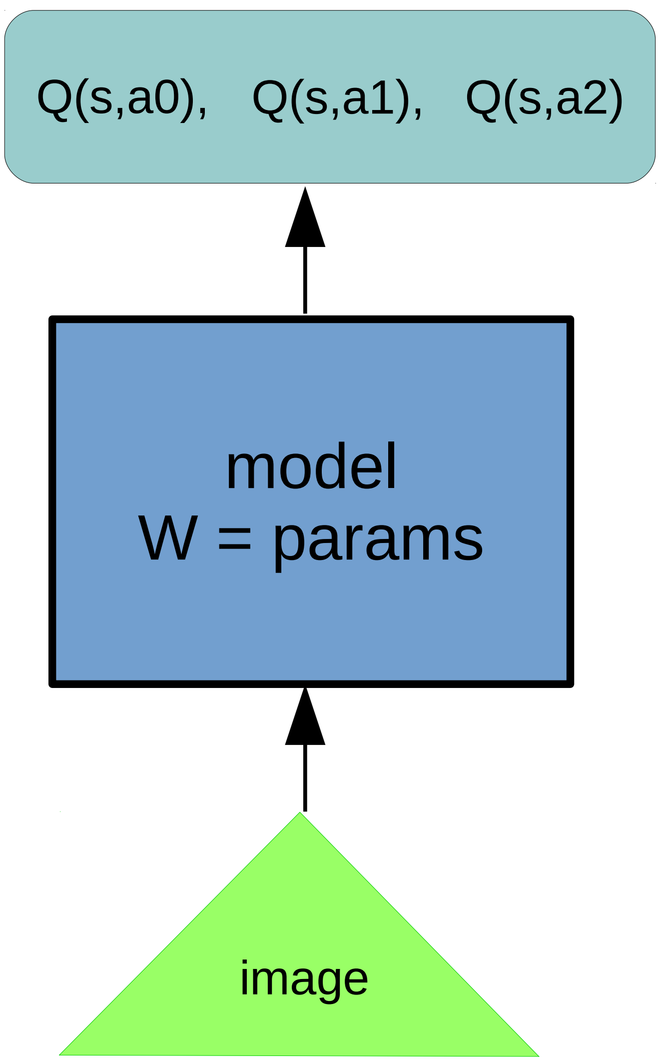
Objective:

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

Gradient step:

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

Approximate Q-learning



Q-values:

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

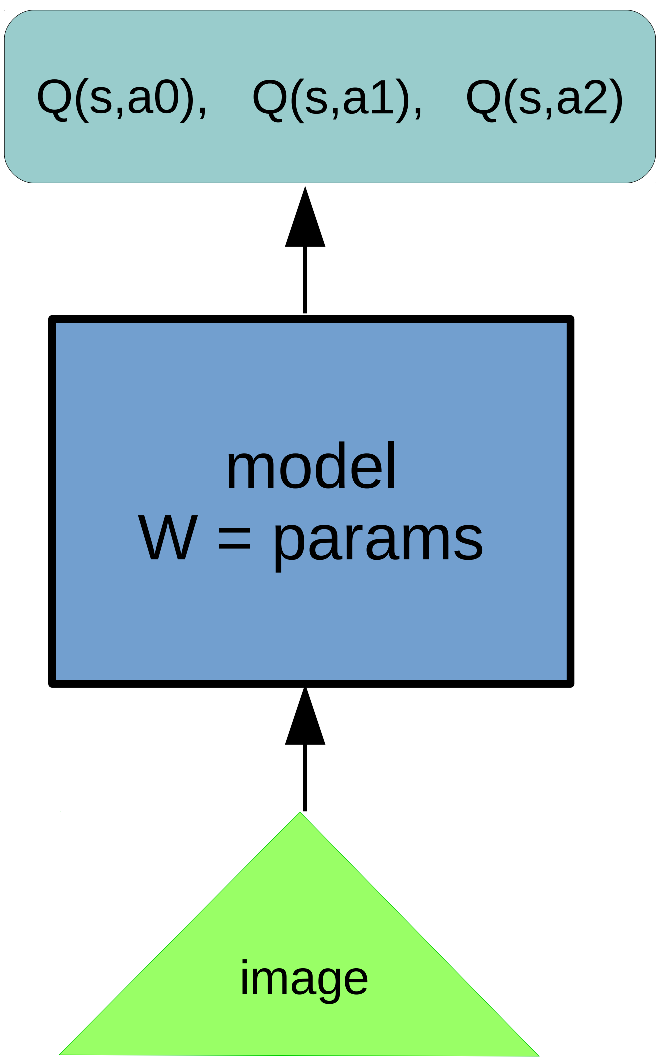
Objective:

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

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:

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

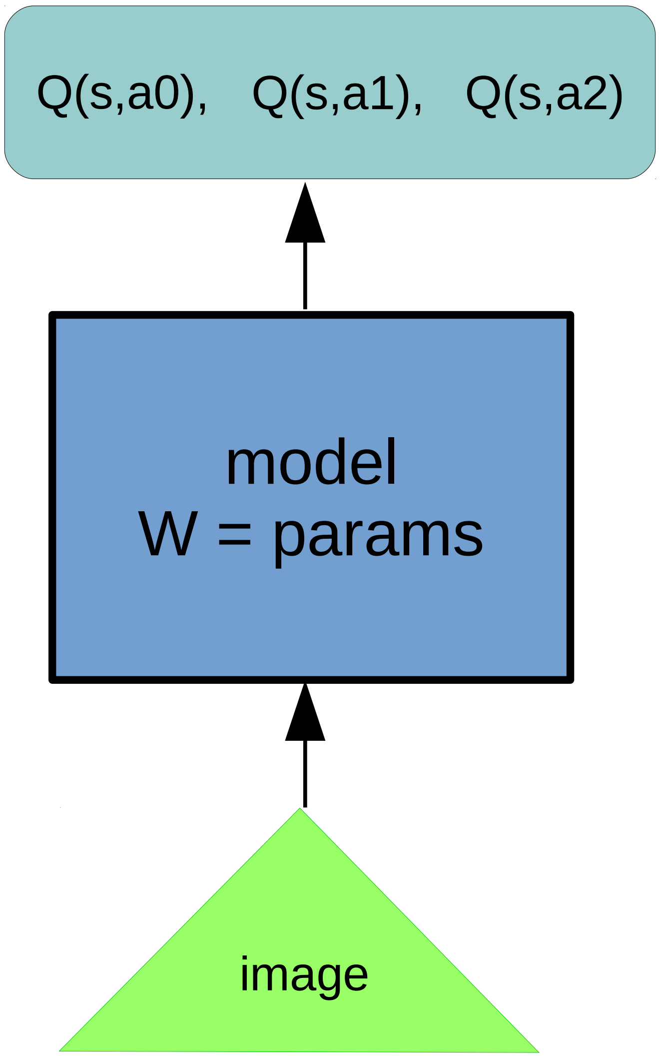
SARSA:

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

Expected Value SARSA:

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

Approximate SARSA



Objective:

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

Q-learning:

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

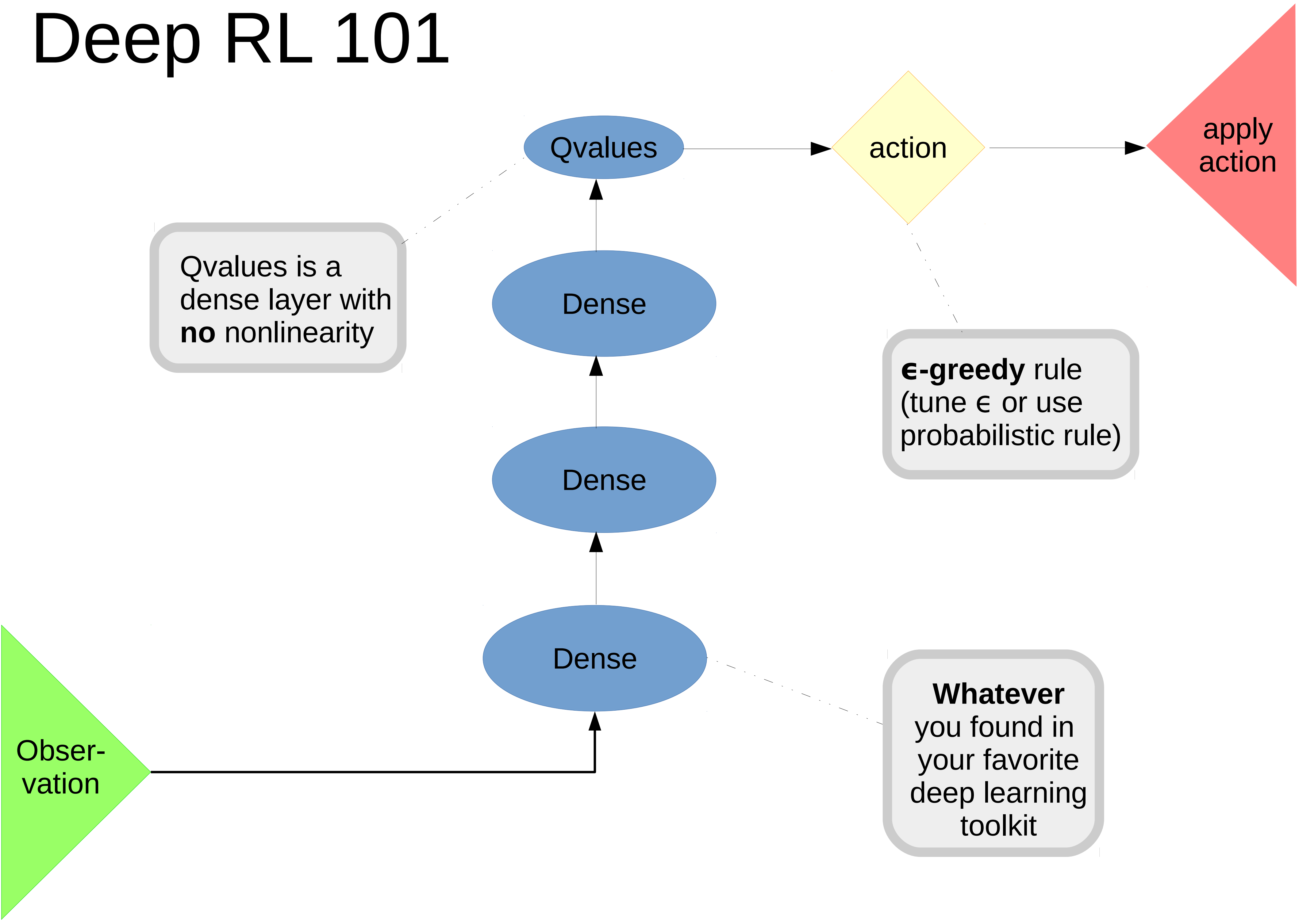
SARSA:

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

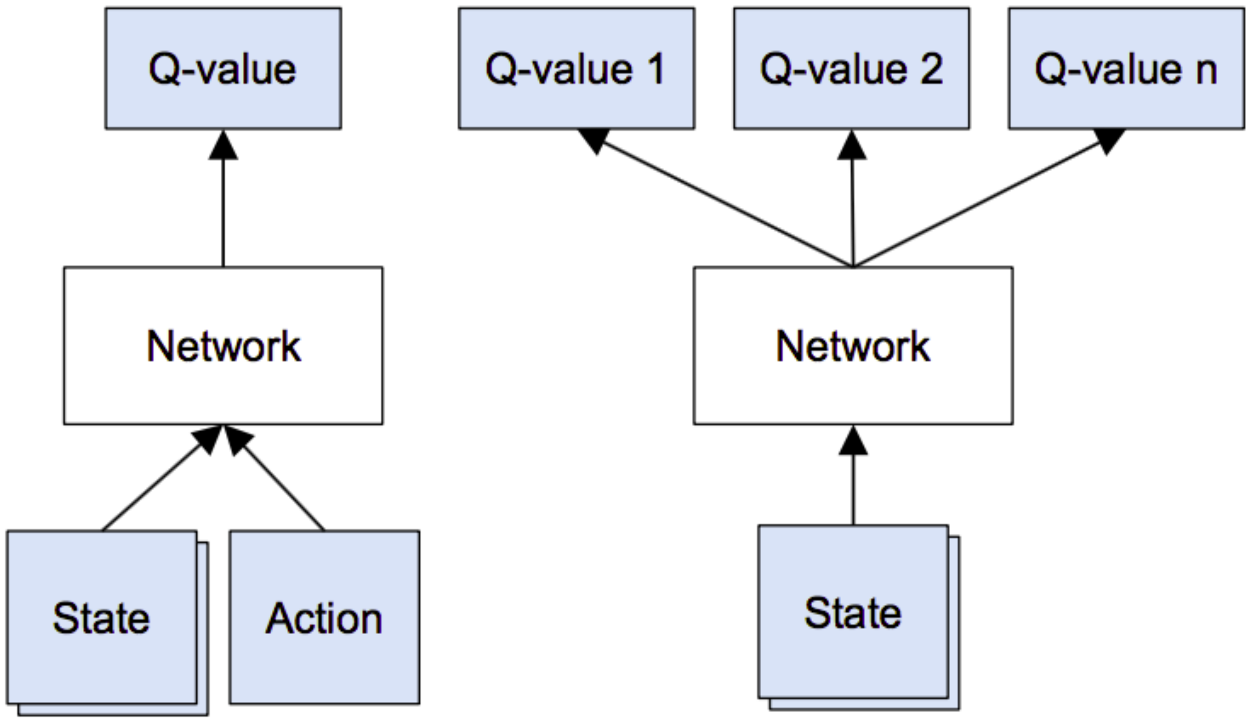
Expected Value SARSA:

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

Deep RL 101


Qvalues
action
apply action
Qvalues is a dense layer with **no** nonlinearity
Dense
Dense
ϵ-greedy rule (tune ϵ or use probabilistic rule)
Dense
Obser-vation
Whatever you found in your favorite deep learning toolkit

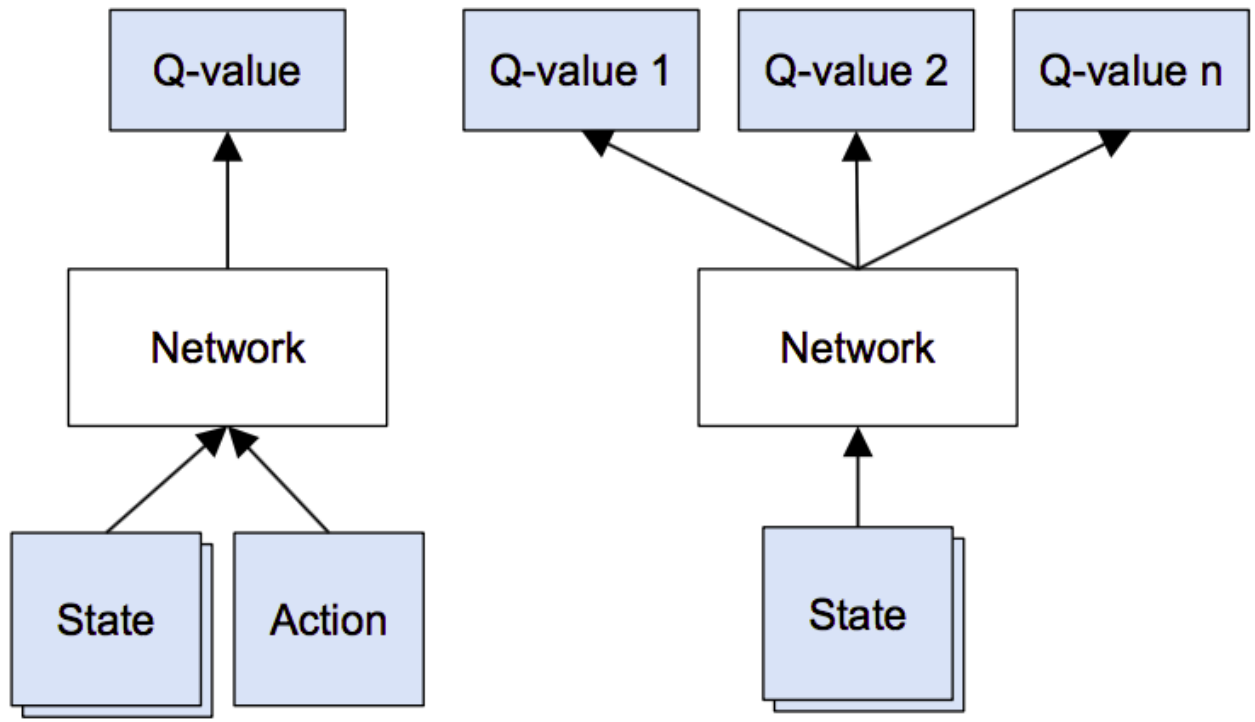
Architectures



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

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

Architectures

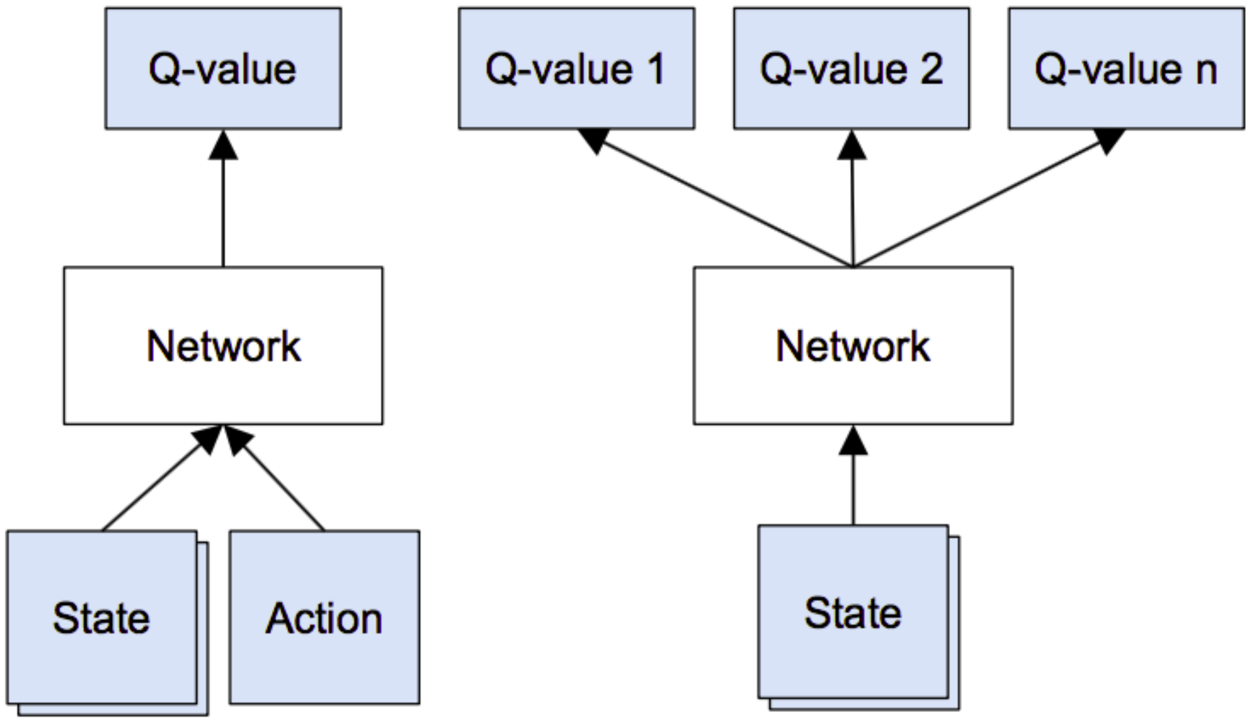


Given **(s,a)**
Predict Q(s,a)

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?

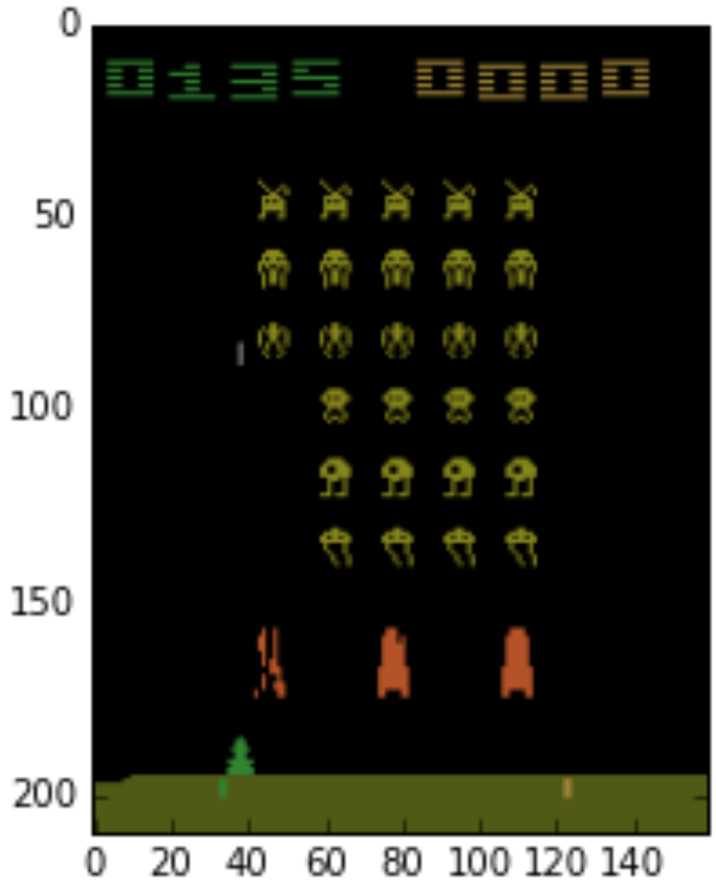
Architectures



Needs one forward pass
for **each action**

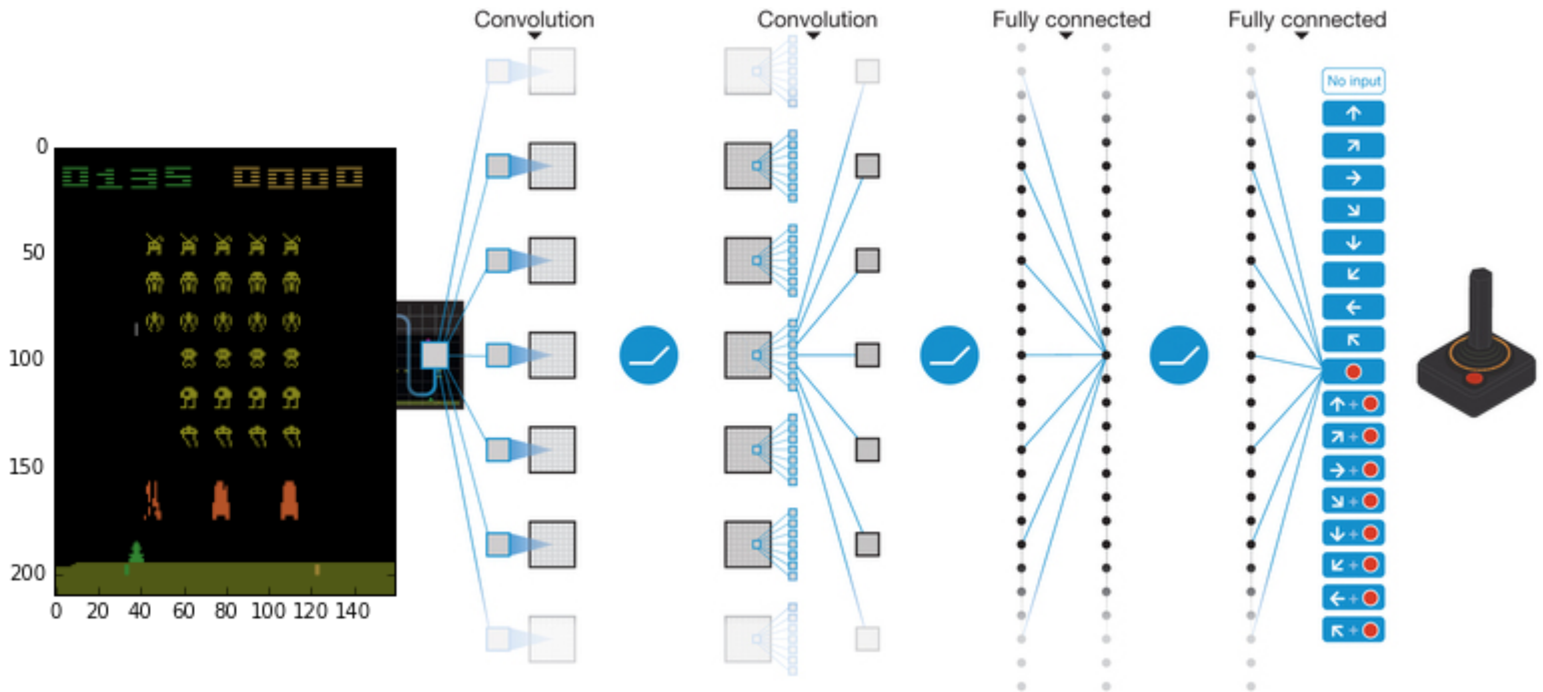
Works if action space is large
efficient when not all actions
are available from each state

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

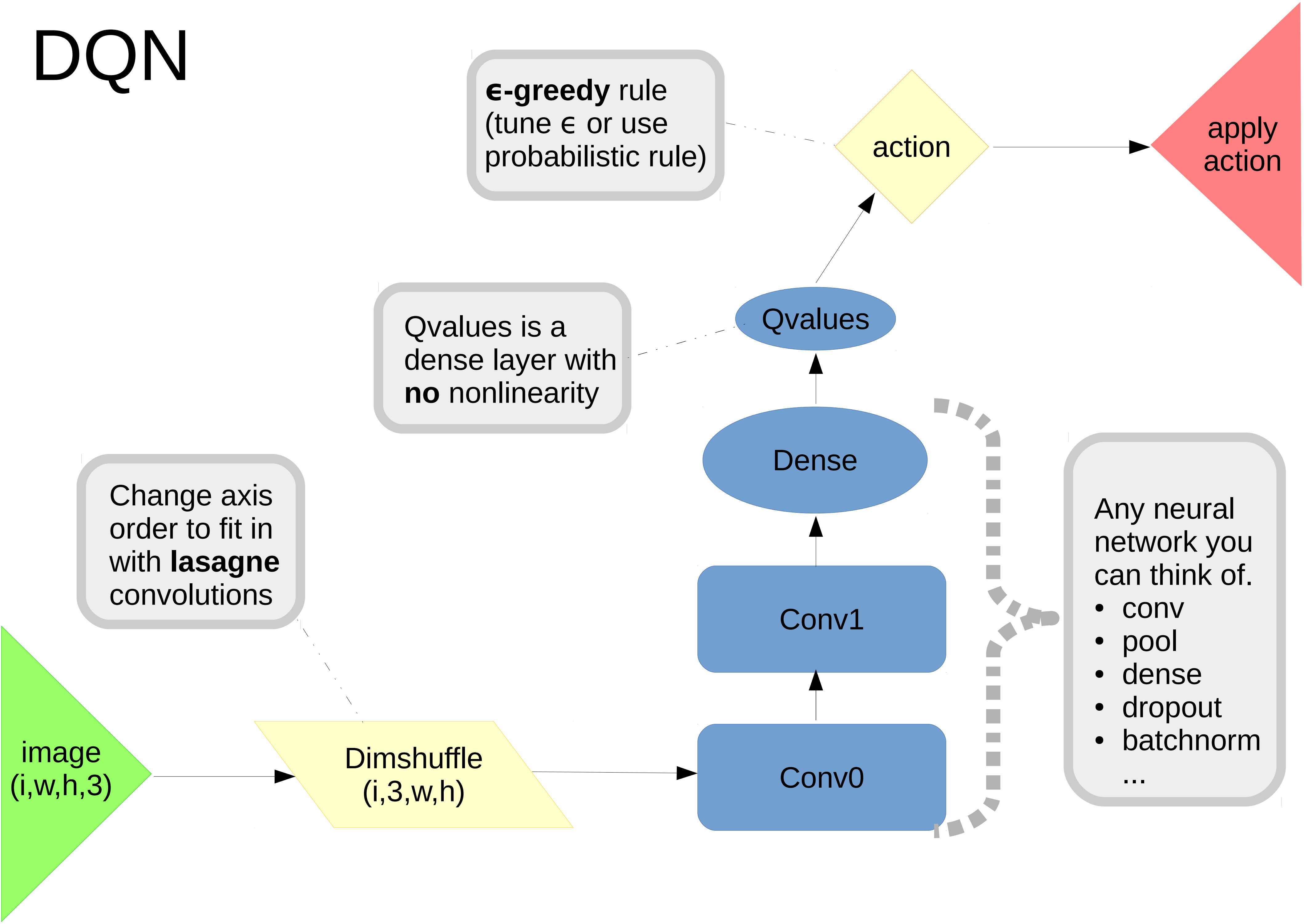


What kind of network digests images well?

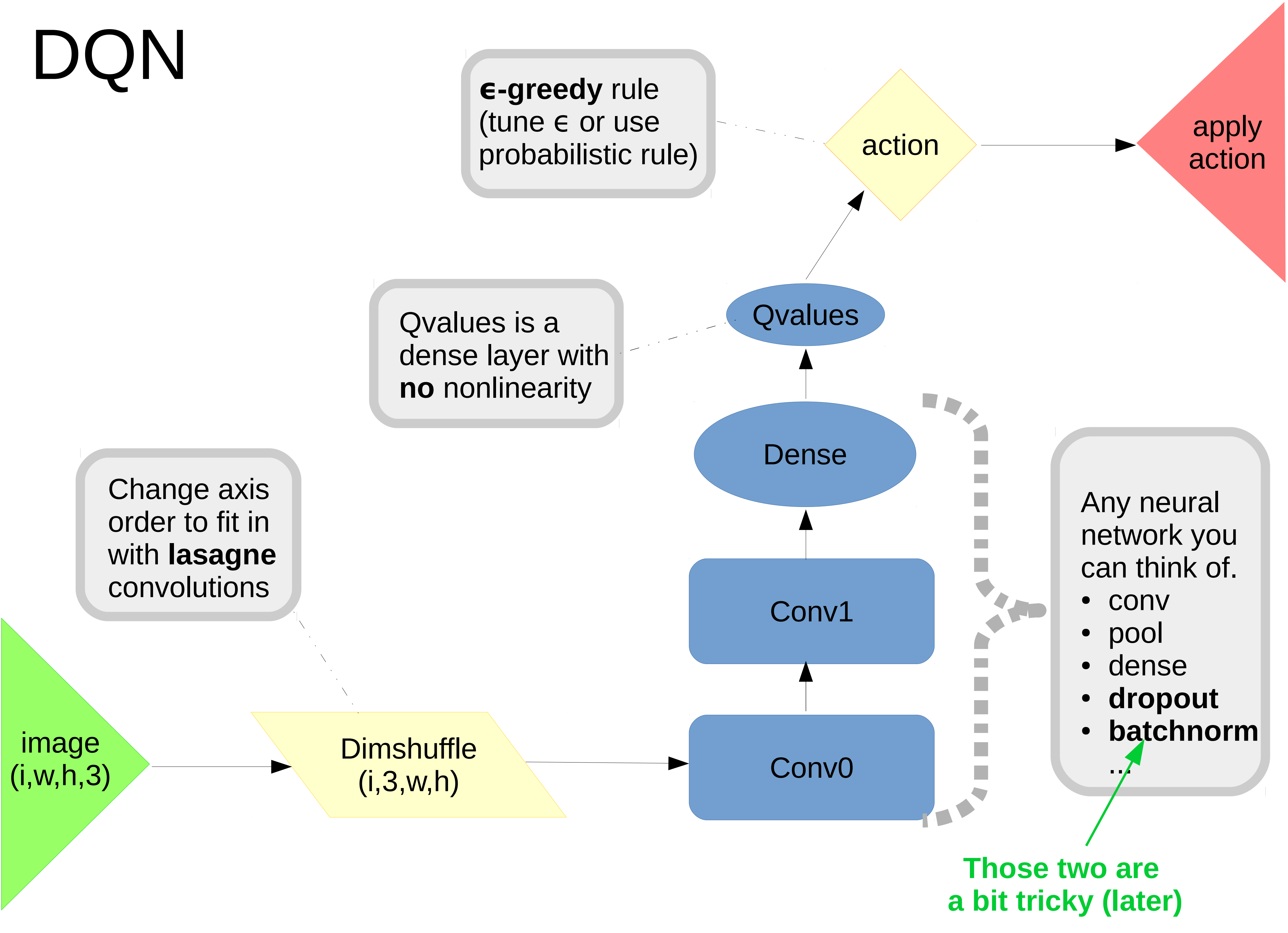
Deep learning approach: DQN



DQN

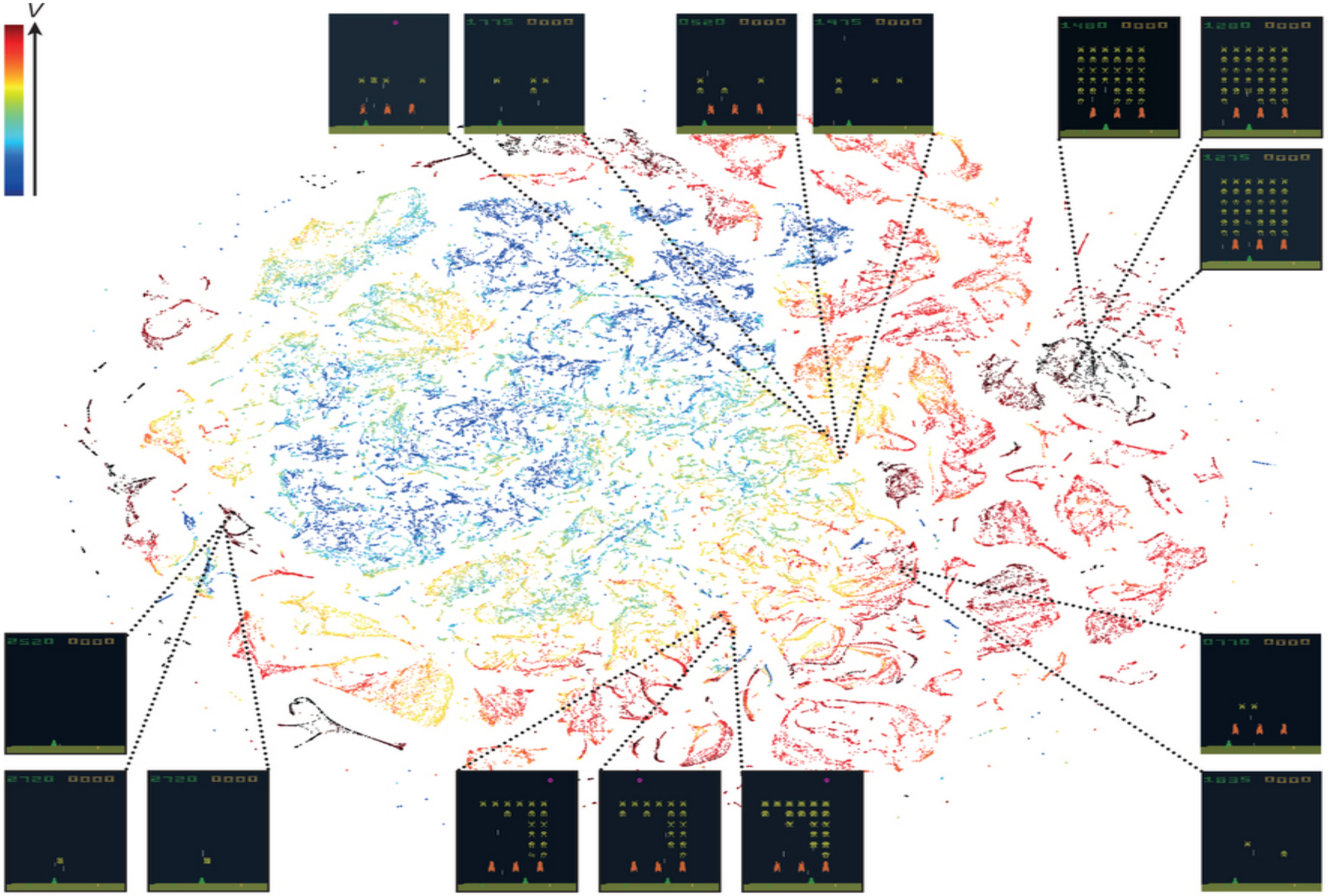

ϵ-greedy rule
(tune ϵ or use
probabilistic rule)
action
apply
action
Qvalues is a
dense layer with
no nonlinearity
Qvalues
Dense
Change axis
order to fit in
with lasagne
convolutions
Conv1
Any neural
network you
can think of.
• conv
• pool
• dense
• dropout
• batchnorm
...
image
(i,w,h,3)
Dimshuffle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

DQN

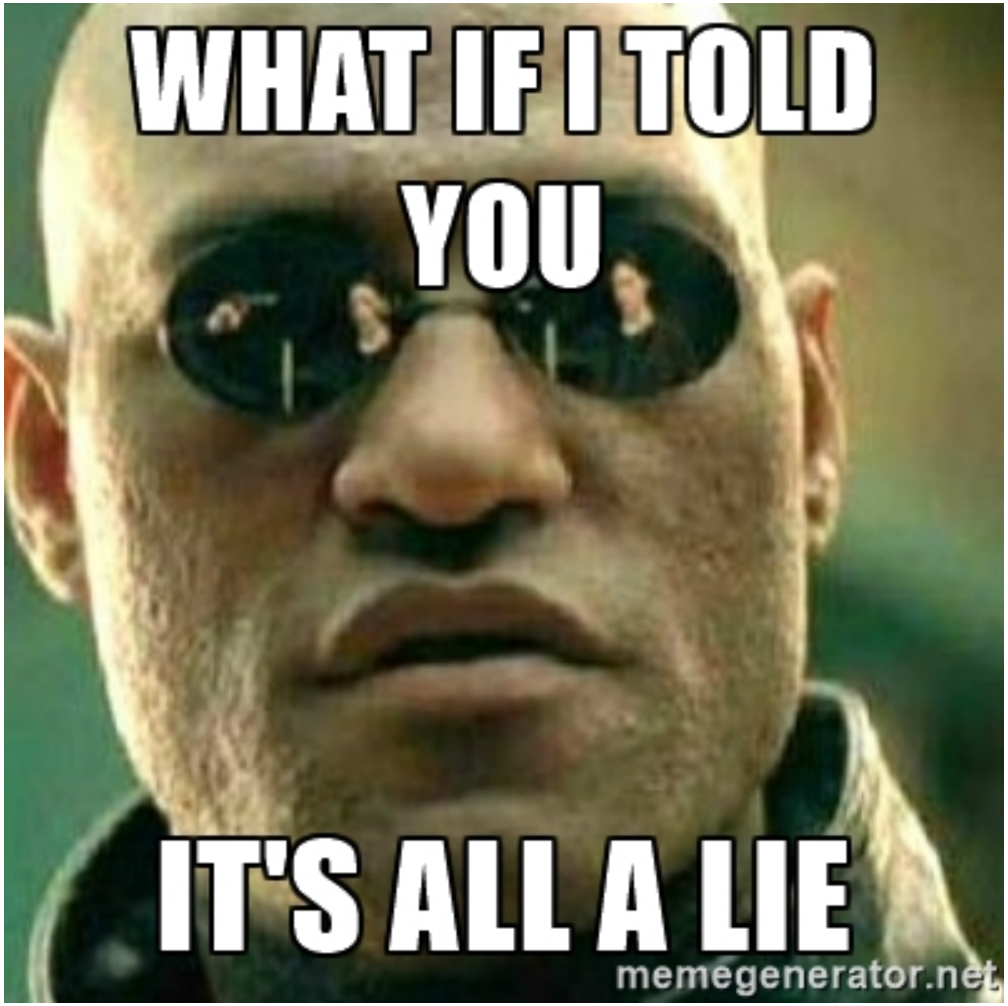


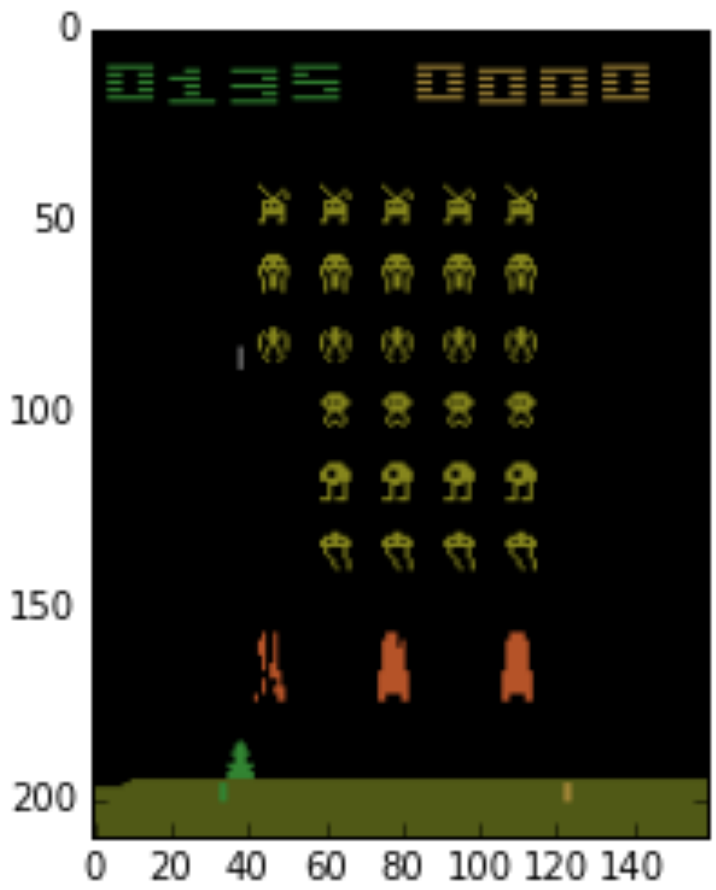
ϵ-greedy rule (tune ϵ or use probabilistic rule)
action
apply action
Qvalues is a dense layer with **no** nonlinearity
Qvalues
Dense
Conv1
Conv0
Change axis order to fit in with **lasagne** convolutions
image (i,w,h,3)
Dimshuffle (i,3,w,h)
Any neural network you can think of.
• conv
• pool
• dense
•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

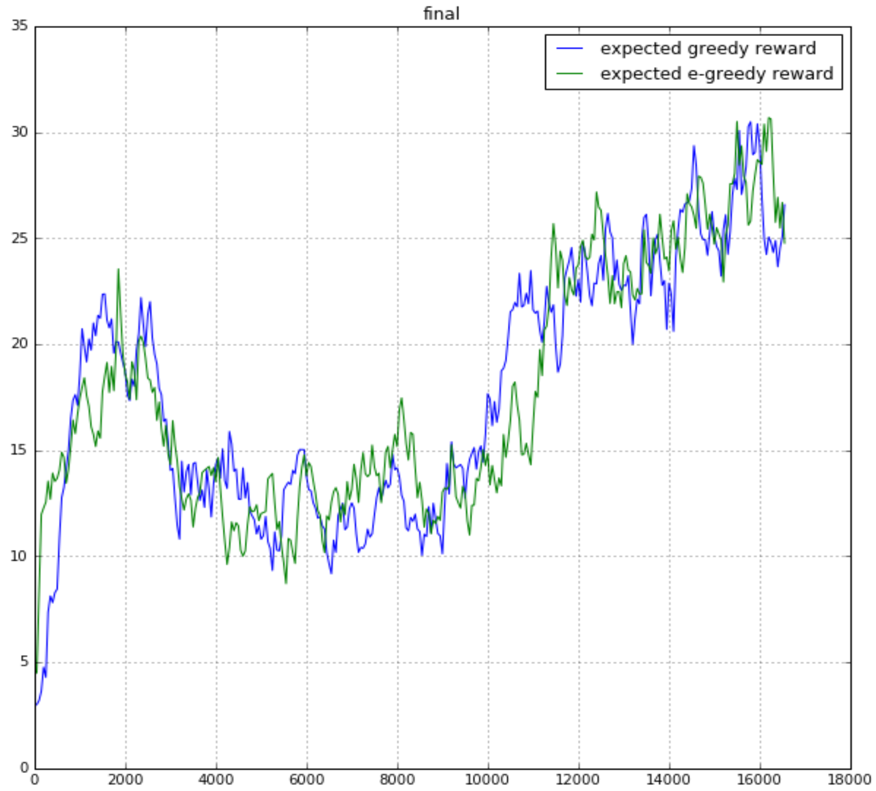


How bad it is if agent spends

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

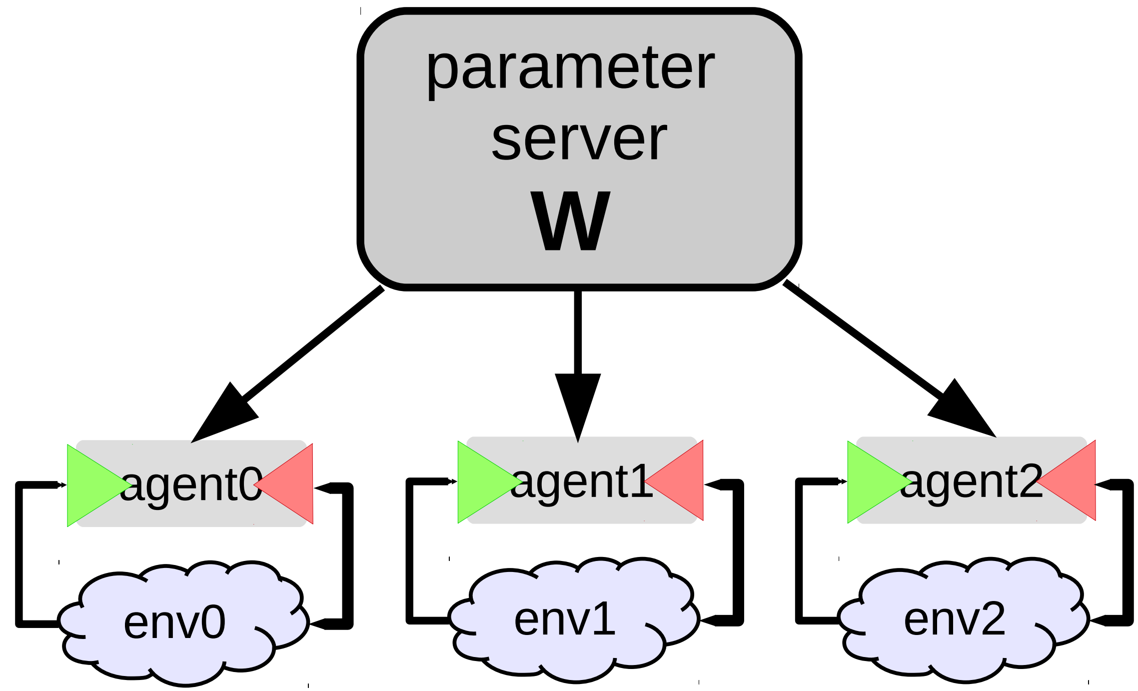
Problem

- 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?**



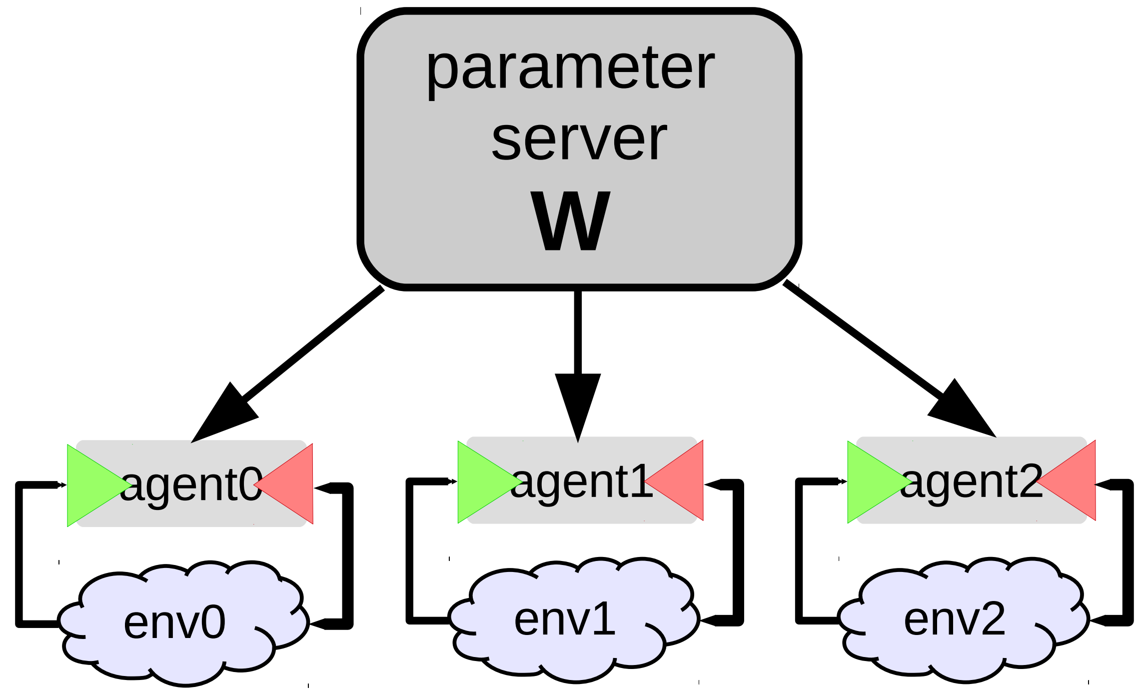
Multiple agent trick

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



Multiple agent trick

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



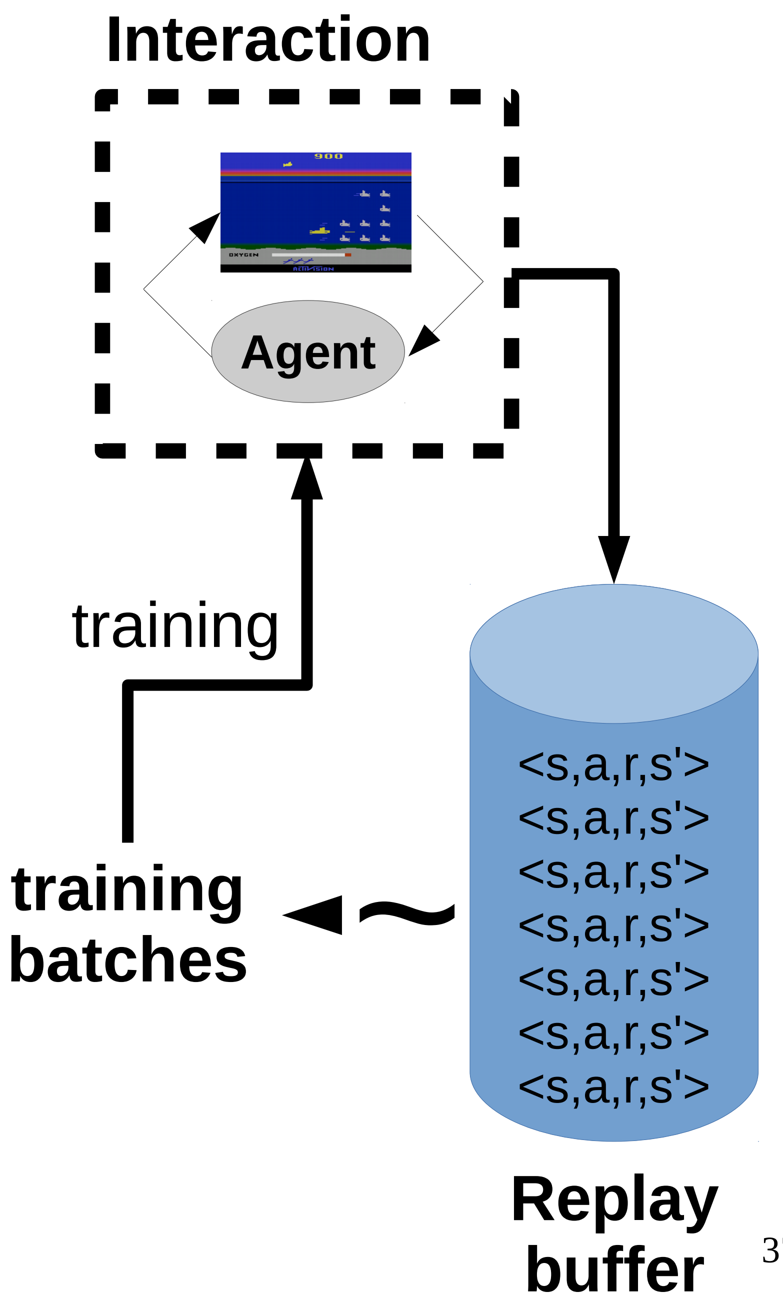
- Chances are, they will be exploring different parts of the environment,
- More stable training,
- Requires a lot of interaction

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 +/- ?

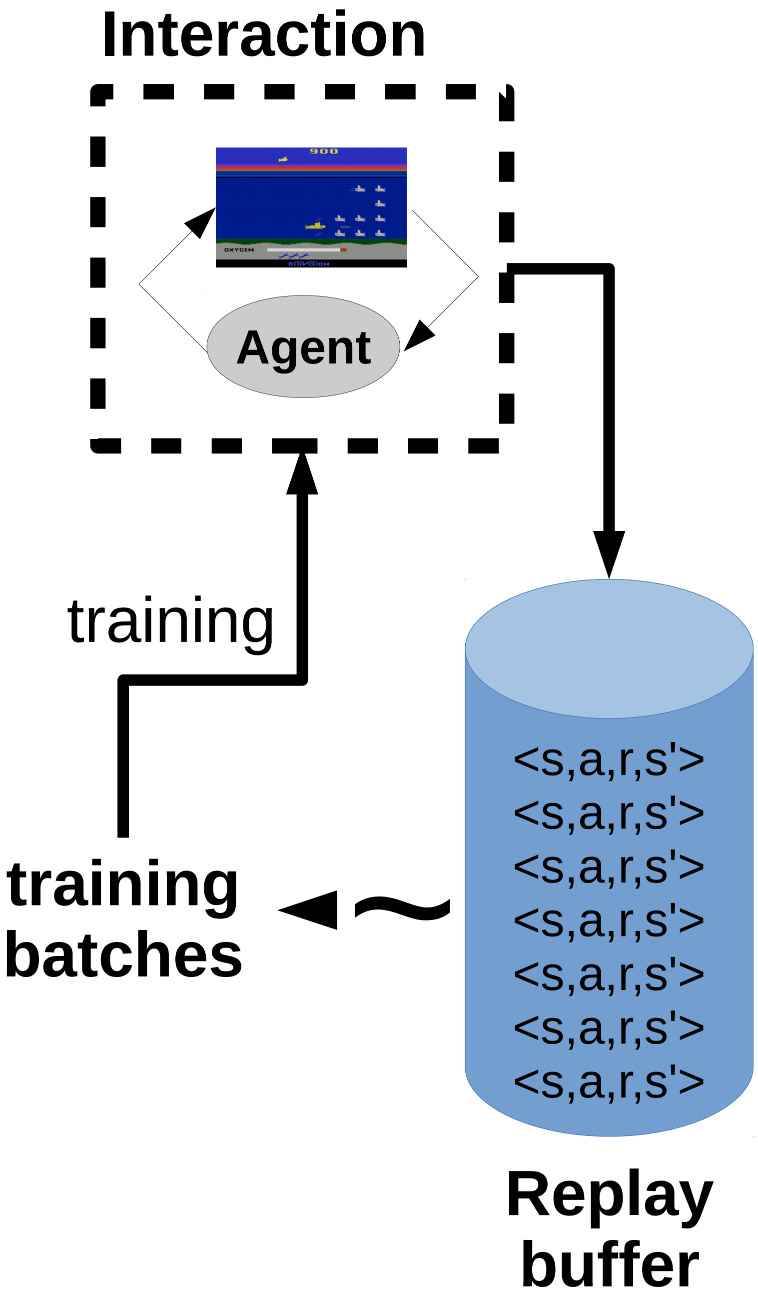


Experience replay

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 :)

advanced stuff coming next lecture

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
- CEM
- 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
- CEM
- 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:

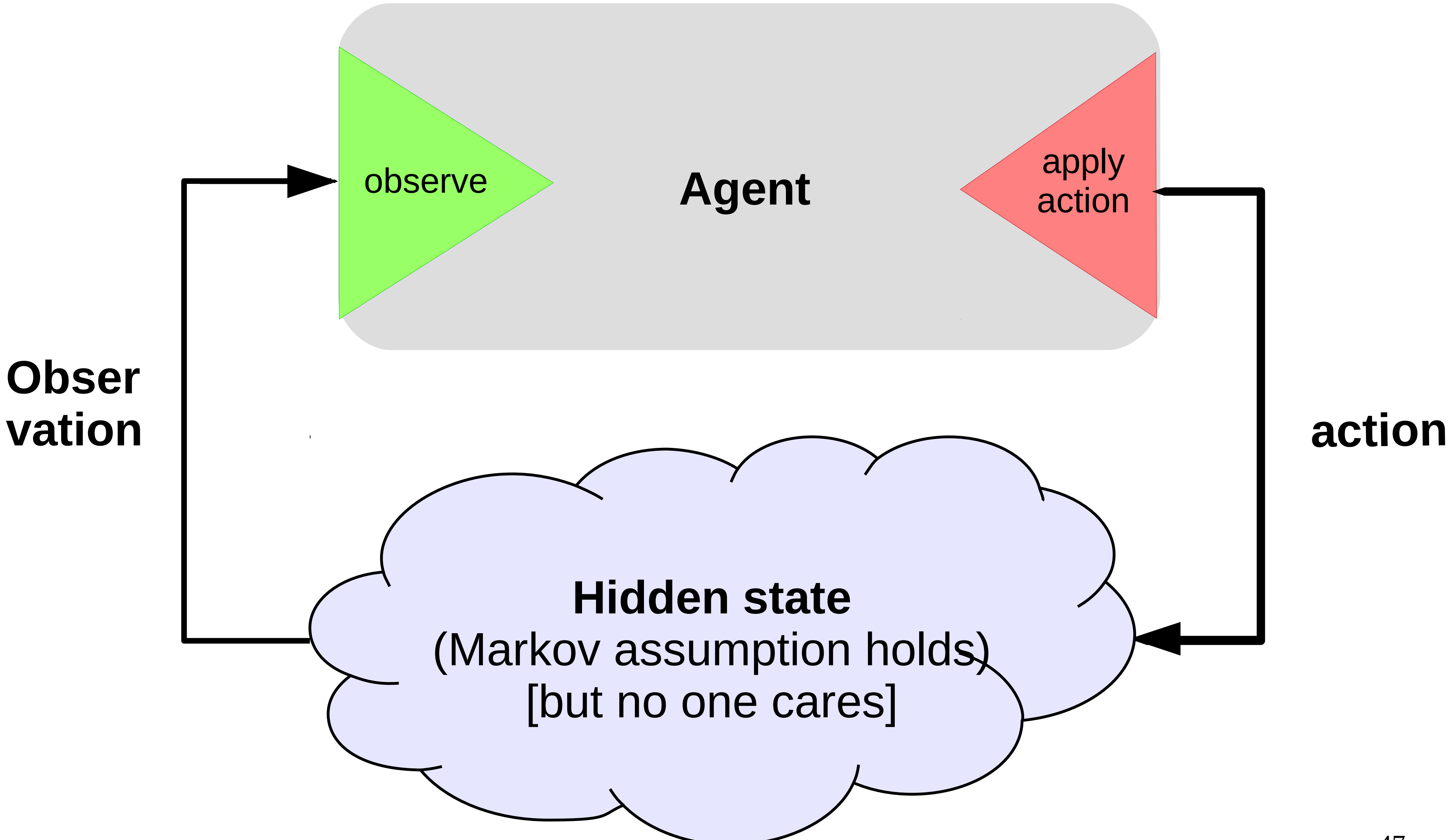
Agent observation does not hold all information about process state (e.g. human field of view).

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

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

- Intuitively that's agent memory state.

Partially observable MDP



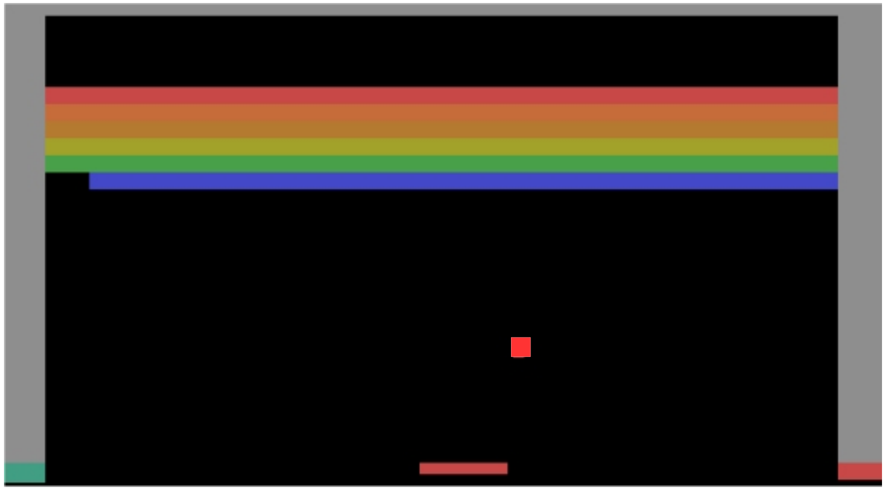
N-gram heuristic

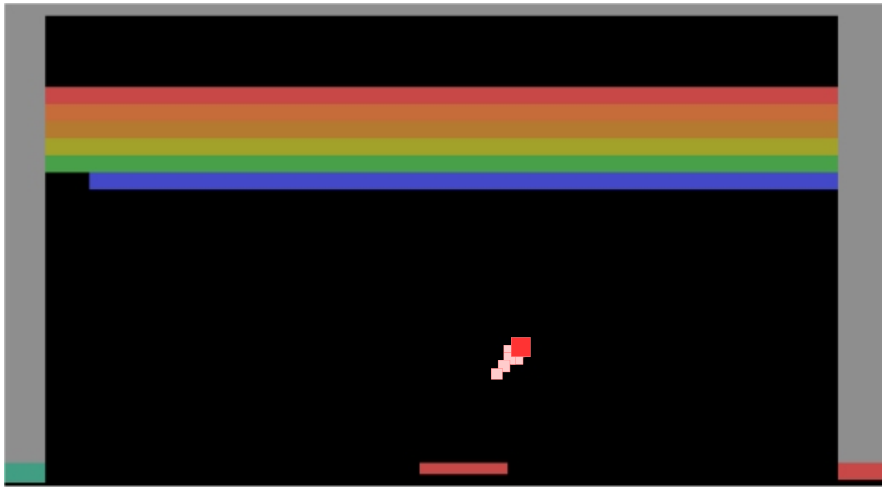
Idea:

$$s_t \neq o(s_t)$$

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

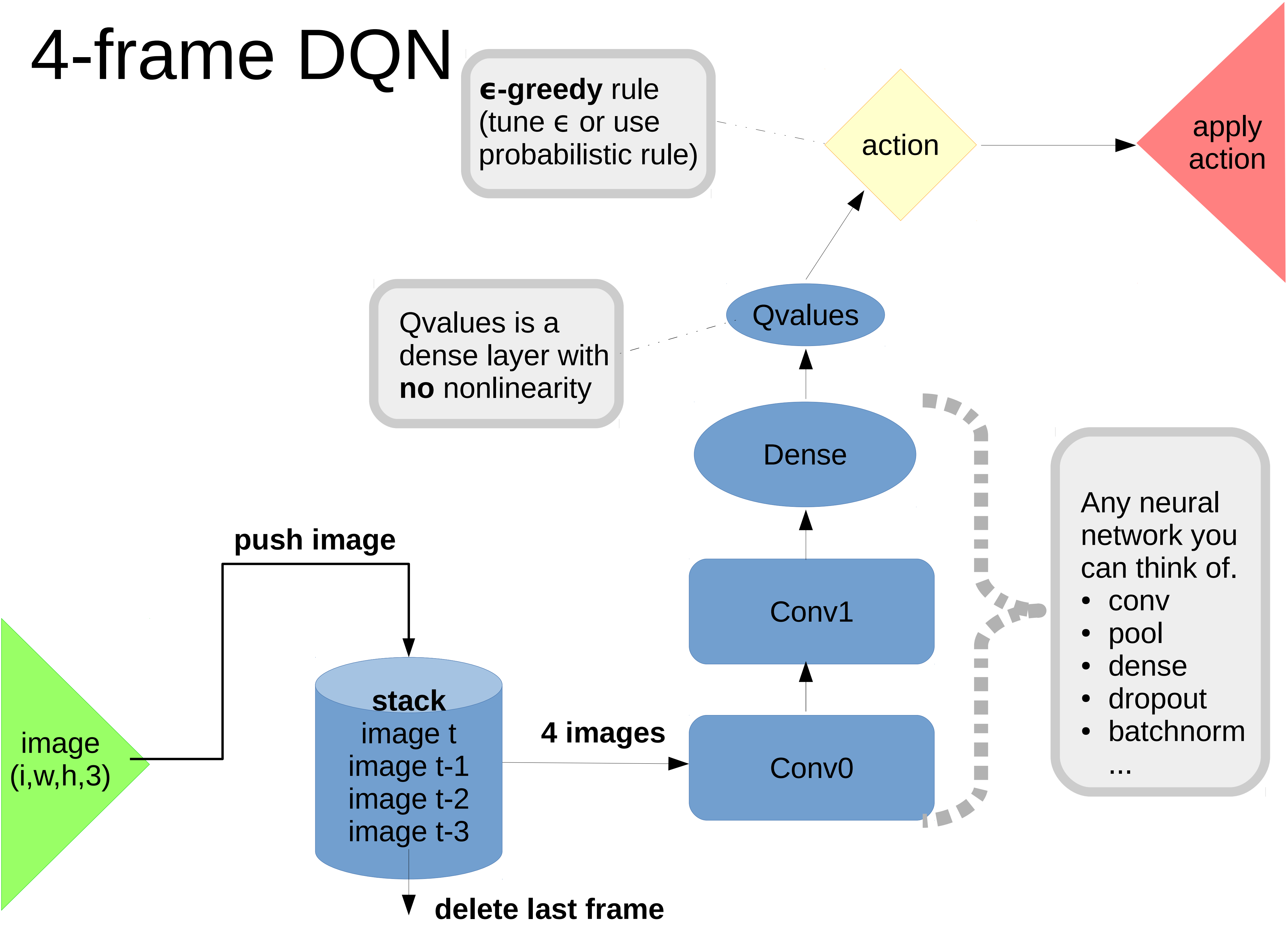
e.g. ball movement in breakout





- One frame
- Several frames

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
Any neural
network you
can think of.
• conv
• pool
• dense
• dropout
• batchnorm
...
push image
Conv1
stack
image t
image t-1
image t-2
image t-3
4 images
Conv0
image
(i,w,h,3)
delete last frame

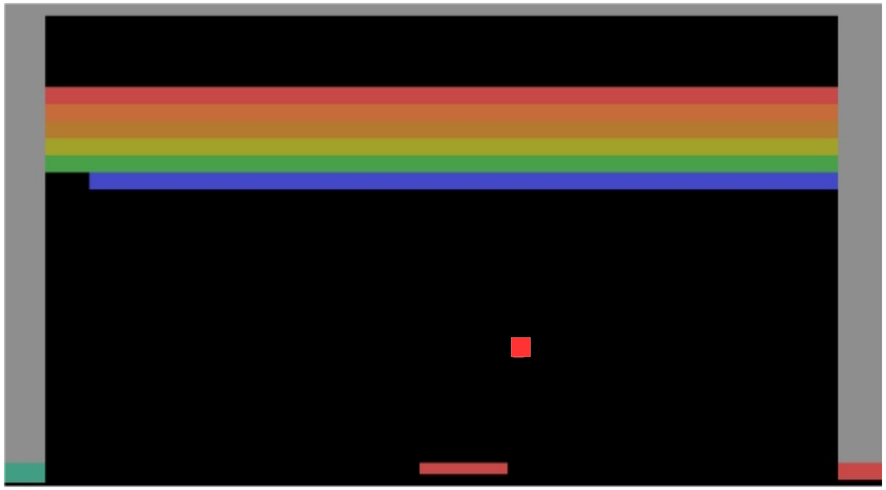
N-gram heuristic

Idea:

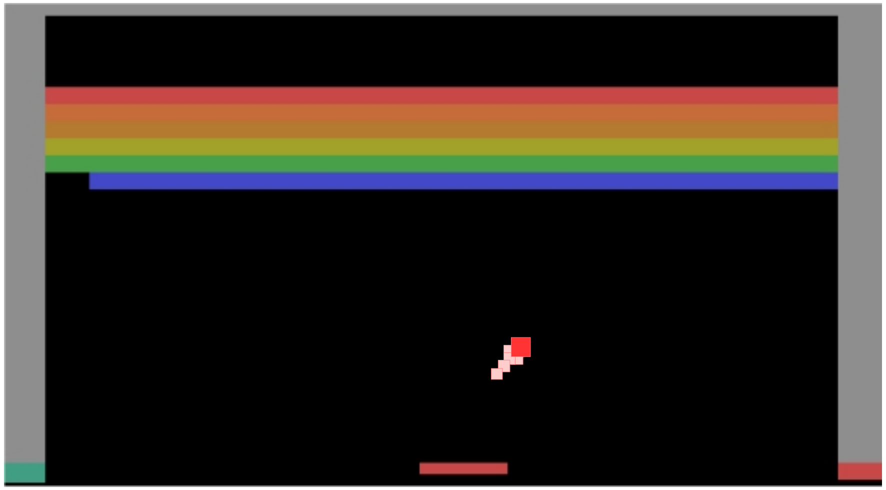
$$s_t \neq o(s_t)$$

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

e.g. ball movement in breakout



- One frame



- Several frames

Alternatives

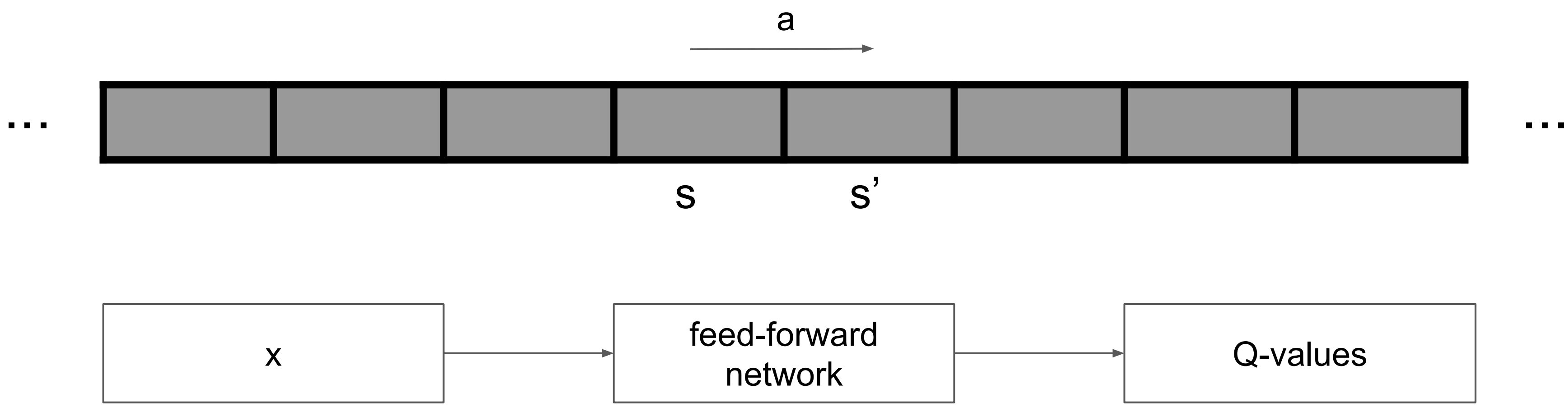
Ngrams:

- 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

Autocorrelation



Target is based on prediction

Q(s, a) correlates with Q(s’, a)

Target network

Idea: use network with frozen weights to compute the target

$$L(\Theta) = E_{s\sim S, a\sim A}[(Q(s,a,\Theta) - (r + \gamma \underbrace{\boxed{\max_{a'} Q(s', a', \Theta^-)}}_{\textbf{Const}}))^2]$$

where Θ^- is the frozen weights

Hard target network:

Update Θ^- every **n** steps and set its weights as Θ

Target network

Idea: use network with frozen weights to compute the target

$$L(\Theta) = E_{s\sim S,a\sim A}[(Q(s,a,\Theta) - (r + \gamma \underbrace{\boxed{\max_{a'} Q(s',a',\Theta^-)}}_{\textbf{Const}}))^2]$$

where Θ^- is the frozen weights

Hard target network:

Update Θ^- every **n** steps and set its weights as Θ

Target network

Idea: use network with frozen weights to compute the target

$$L(\Theta) = E_{s\sim S,a\sim A}[(Q(s,a,\Theta) - (r + \gamma \underbrace{\boxed{\max_{a'} Q(s',a',\Theta^-)}}_{\textbf{Const}}))^2]$$

where Θ^- is the frozen weights

Hard target network:

Update Θ^- every **n** steps and set its weights as Θ

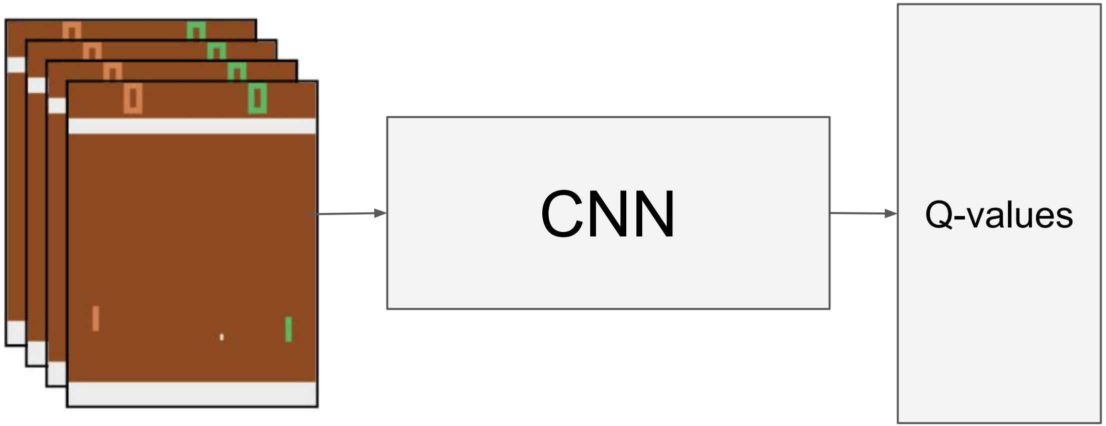
Soft target network:

Update Θ^- every step:

$$\Theta^- = (1-\alpha)\Theta^- + \alpha\Theta$$

Playing Atari with Deep Reinforcement Learning

(2013, Deepmind)



4 last frames as input

Experience replay

10^6 last transitions

Update weights using:

$$L(\Theta) = E_{s\sim S, a\sim A}[(Q(s,a,\Theta) - (r + \gamma \max_{a'} Q(s', a', \Theta^-)))^2]$$

Update Θ^- every 5000 train steps

Problem of overestimation

We use “max” operator to compute the target

$$L(s,a) = (Q(s,a) - (r + \gamma \max_{a'} Q(s',a')))^2$$

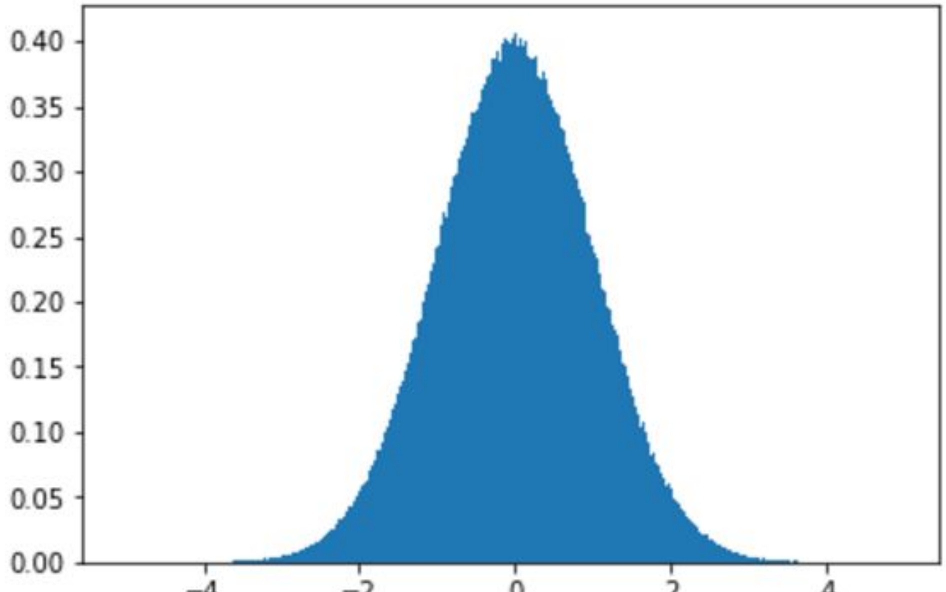
We have a problem

(although we want $\mathrm{E}_{s\sim S,a\sim A}[L(s,a)]$ to be equal zero)

Problem of overestimation

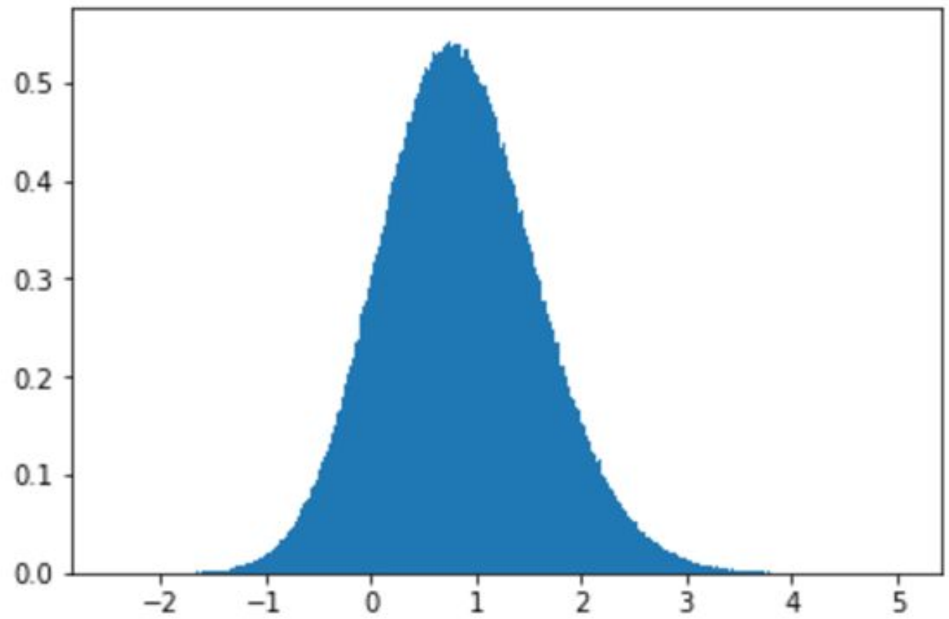
Normal distribution
$3*10^6$ samples

mean: ~0.0004



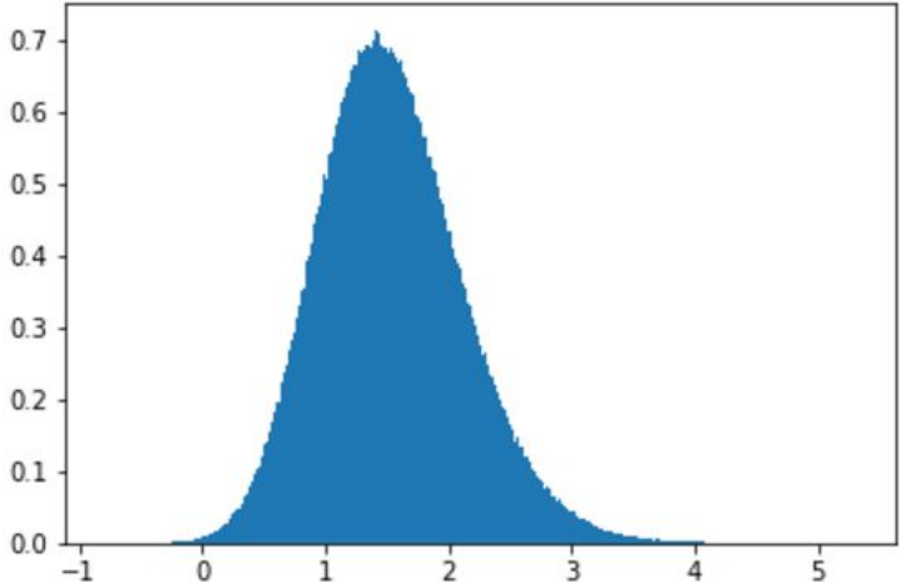
Problem of overestimation

Normal distribution
$3*10^6$ x 3 samples
Then take maximum of every tuple
mean: ~0.8467

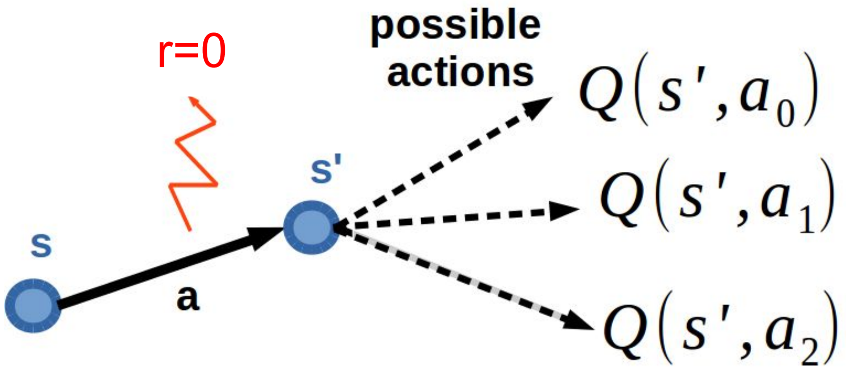


Problem of overestimation

Normal distribution
$3*10^6$ x 10 samples
Then take maximum of every tuple
mean: ~1.538



Problem of overestimation

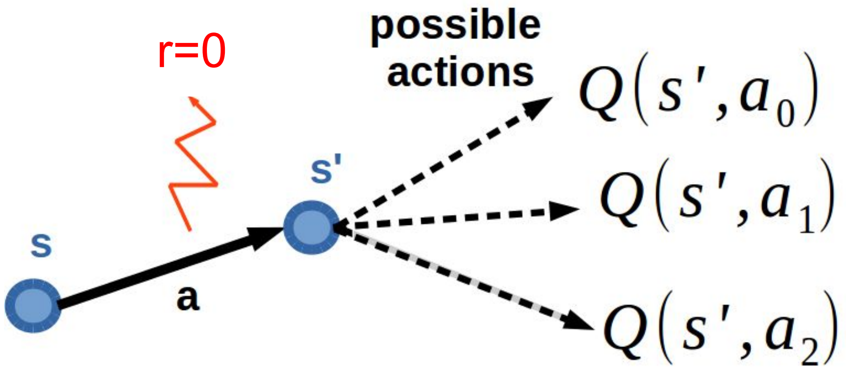


Suppose true $Q(s', a')$ are equal to **0** for all a'

But we have an approximation (or other) error $\sim N(0, \sigma^2)$

So $Q(s, a)$ should be equal to **0**

Problem of overestimation



But if we update $Q(s, a)$ towards $r + \gamma \max_{a'} Q(s', a')$

we will have overestimated $Q(s, a)$ > **0** because

$$\mathrm{E}[\max_{a'} Q(s', a')] >= \max_{a'} E[Q(s', a')]$$

Double Q-learning

(NIPS 2010)

$$y = r + \gamma \max_{a'} Q(s', a')$$

- Q-learning target

$$y = r + \gamma Q(s', \operatorname{argmax}_{a'} Q(s', a'))$$

- Rewritten Q-learning target

Idea: use two estimators of q-values: Q^A, Q^B
They should compensate mistakes of each other because they will be independent
Let's get argmax from another estimator!

$$y = r + \gamma Q^A(s', \operatorname{argmax}_a Q^B(s', a'))$$

- Double Q-learning target

Double Q-learning

(NIPS 2010)

Algorithm 1 Double Q-learning

```
Initialize Q^A, Q^B, s
repeat
    Choose a, based on Q^A(s,·) and Q^B(s,·), observe r, s'
    Choose (e.g. random) either UPDATE(A) or UPDATE(B)
    if UPDATE(A) then
        Define a* = arg max_a Q^A(s', a)
        Q^A(s,a) ← Q^A(s,a) + α(s,a)(r + γQ^B(s', a*) − Q^A(s,a))
    else if UPDATE(B) then
        Define b* = arg max_a Q^B(s', a)
        Q^B(s,a) ← Q^B(s,a) + α(s,a)(r + γQ^A(s', b*) − Q^B(s,a))
    end if
    s ← s'
until end
```

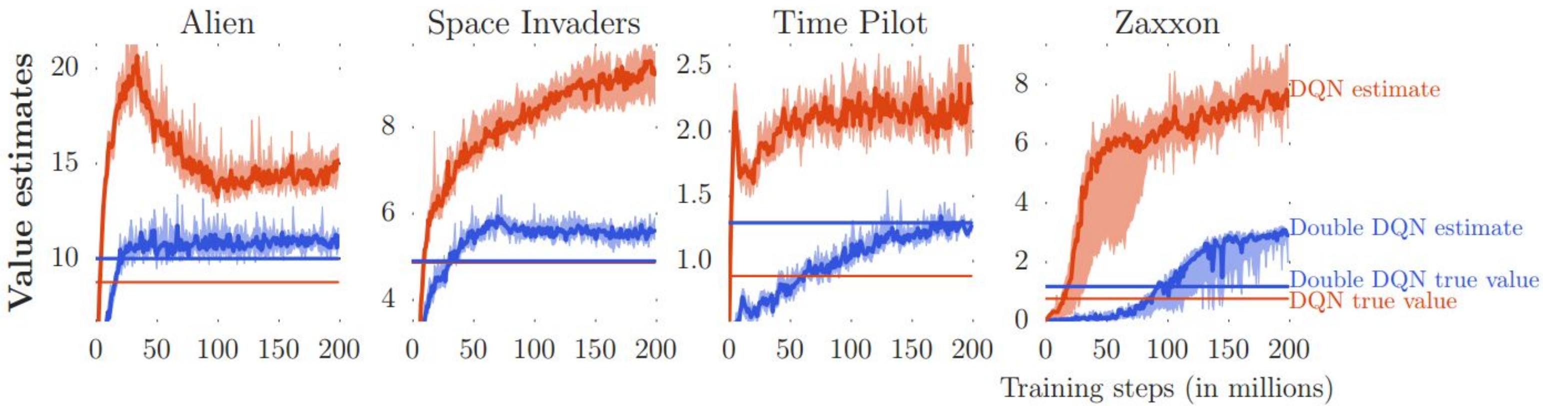
Can we combine this algorithm with DQN?

Deep RL with Double Q-learning

(Deepmind, 2015)

Idea: use main network to choose action!

$$y_{dqn} = r + \gamma \max_{a'} Q(s', a', \Theta^-)$$
$$y_{ddqn} = r + \gamma Q(s', \mathrm{argmax}_{a'} Q(s', a', \Theta), \Theta^-)$$



	DQN	Double DQN	Double DQN (tuned)
Median	47.5%	88.4%	116.7%
Mean	122.0%	273.1%	475.2%

Experience Replay

State	Action	Reward	Next state
s_0	a_0	0	s_1
s_1	a_1	0	s_2
...	...	...	...
s_(n-1)	a_(n-1)	0	s_n
s_n	**a_n**	**100**	**s_(n+1)**
s_(n+1)	a_(n+1)	0	s_(n+2)
...	...	...	...

Prioritized Experience Replay

(2016, Deepmind)

Idea: sample transitions from xp-replay cleverly

We want to set probability for every transition. Let's use the absolute value of TD-error of transition as a probability!

$$\text{TD-error } \delta = Q(s,a) - (r + \gamma Q(s', \operatorname{argmax}_{a'} Q(s', a', \Theta), \Theta^-))$$

$$p = |\delta|$$

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$ where α is the priority parameter (when α is 0 it's the uniform case)

Do you see the problem?

Transitions become non i.i.d. and therefore we introduce the bias.

Prioritized Experience Replay

(2016, Deepmind)

Solution: we can correct the bias by using importance-sampling weights

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta}$$

where β is the parameter

So we sample using $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ and multiply error by w_i

Prioritized Experience Replay
(2016, Deepmind)

Additional details

We also normalize weights by $1/\max_i w_i$ (here is no mathematical reason)

When we put transition into experience replay, we set maximal priority $p_t = \max_{i<t} p_i$

Double Q-learning visualization

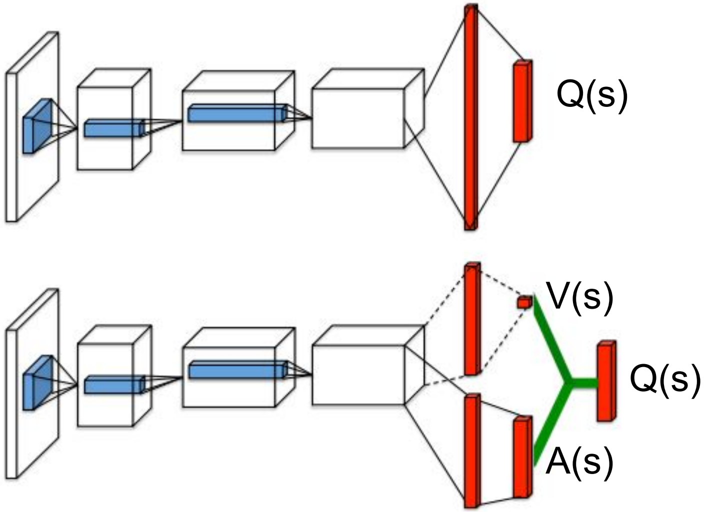
Dueling Network Architectures for Deep RL

(2016, Deepmind)

Idea: change the network's architecture.

Recall:
Advantage Function A(s,a) = Q(s,a) - V(s)

So, Q(s,a) = A(s,a) + V(s)



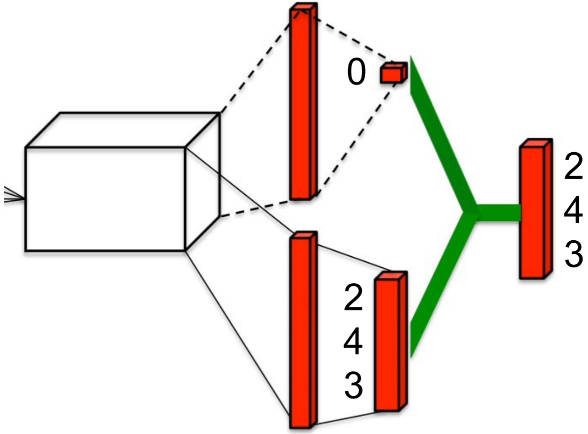
Do you see the problem?

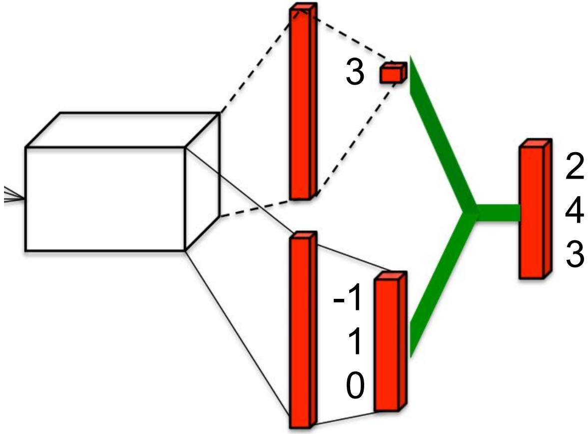
Dueling Network Architectures for Deep RL

(2016, Deepmind)

Here is one extra freedom degree!

Example:





Which one is good?

Dueling Network Architectures for Deep RL

(2016, Deepmind)

Solution: require $\max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha)$ to be equal to zero!

So the **Q-function** is computed as:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha) \right)$$

Dueling Network Architectures for Deep RL

(2016, Deepmind)

Solution: require $\max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha)$ to be equal to zero!

So the **Q-function** is computed as:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha) \right)$$

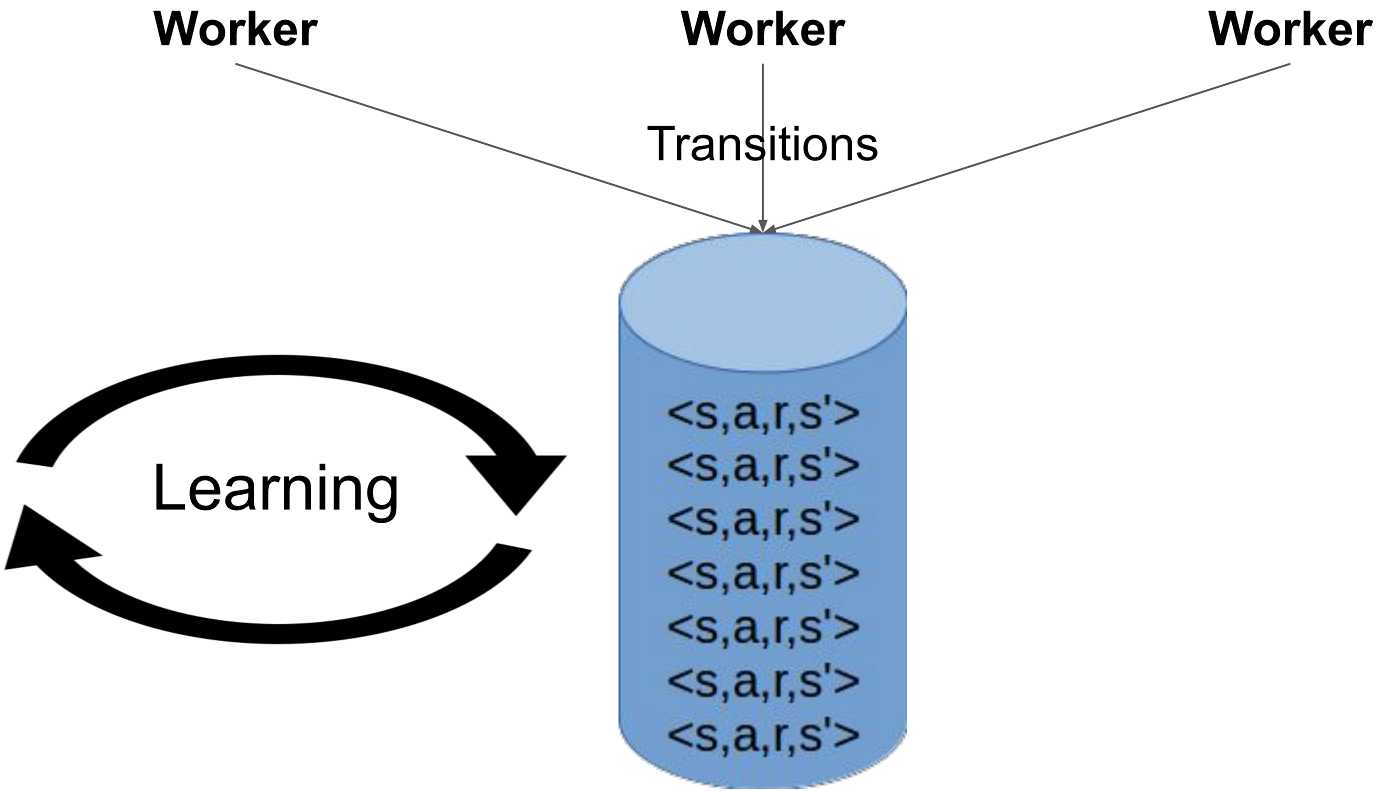
Authors of this papers also introduced this way to compute Q-values:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right)$$

They wrote that this variant increases stability of the optimization
(The fact that this loses the original semantics of Q doesn't matter)

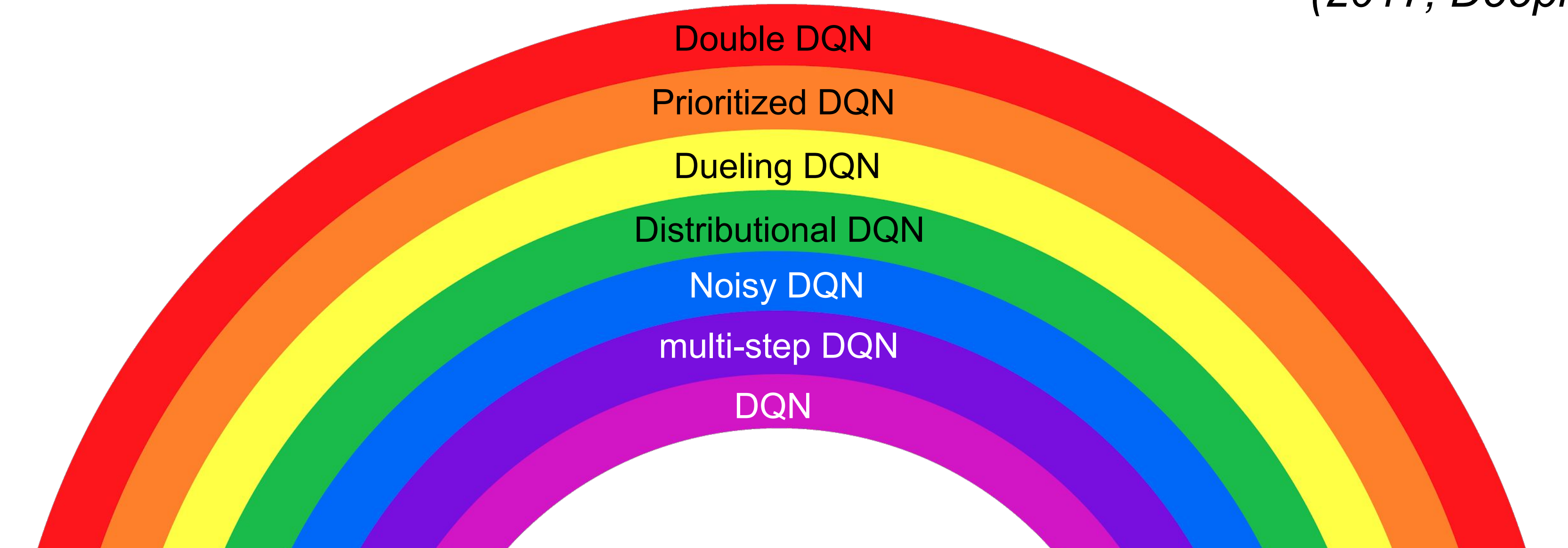
Asynchronous Methods for Deep RL

(2016, Deepmind)



Rainbow

(2017, Deepmind)



Algorithm	Median
DQN	79.5%
Double DQN	117%
Rainbow	223%

R2D2

(2018, Deepmind)

LSTM

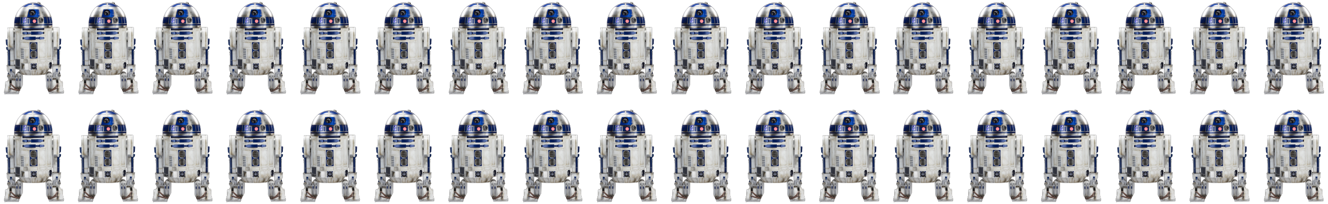
Distributed Prioritized
Experience Replay

n-step DQN

Reward re-scaling

Double DQN

Dueling DQN



Median performance: **1920%** of human performance!

Thanks for your attention!