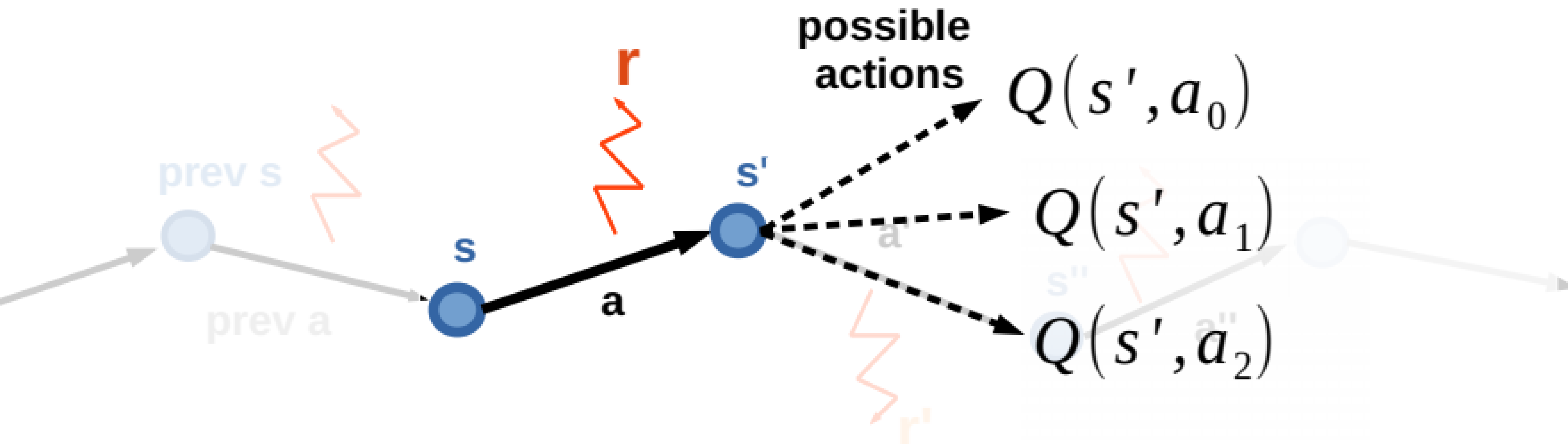
Lecture 1.4

Reinforcement Learning: Value Function Approximation

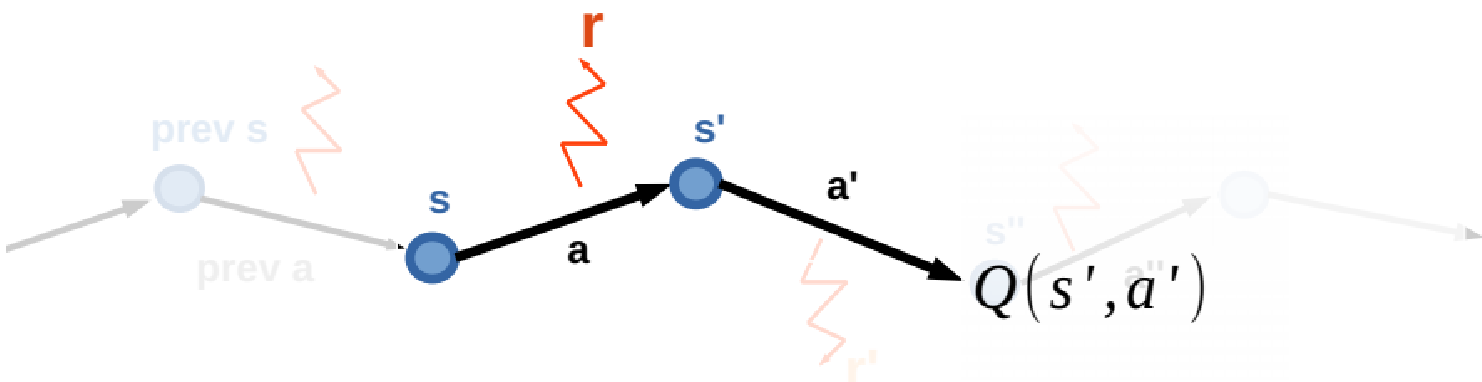
Alexey Gruzdev
alexey.s.gruzdev@gmail.com
HSE, Winter 2019

Recap: Q-learning



- Initialize $\boldsymbol{Q(s,a)}$ with zeros
- Cycle:
  - Sample **<s, a, r, s'>** from environment
  - Compute $\boldsymbol{\hat{Q}(s,a) = r(s,a) + \gamma \max_{a_i} Q(s',a_i)}$
  - Update: $\boldsymbol{Q(s_t, at) = \ \alpha\, \hat{Q}(s,a)}$ **+ (1 -** $\boldsymbol{\alpha}$**)** $\boldsymbol{Q(s_t, at)}$

Recap: S-A-R-S-A

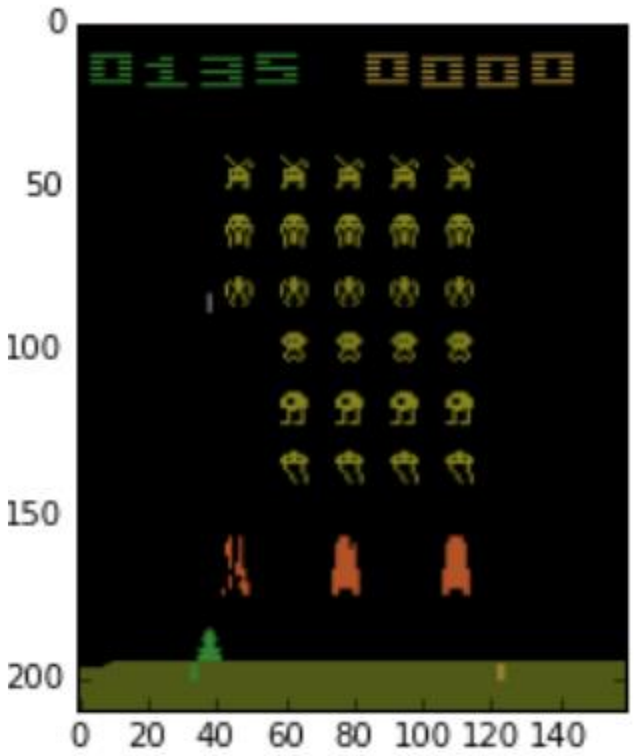


- Initialize $\boldsymbol{Q(s,a)}$ with zeros
- Cycle:
  - Sample **<s, a, r, s', a'>** from environment
  - Compute $\widehat{\boldsymbol{Q}}(\boldsymbol{s},\boldsymbol{a}) = \boldsymbol{r}(\boldsymbol{s},\boldsymbol{a}) + \boldsymbol{\gamma Q}(\boldsymbol{s}',\boldsymbol{a}')$
  - Update: $\boldsymbol{Q(s_t, at)} = \ \boldsymbol{\alpha}\, \widehat{\boldsymbol{Q}}(\boldsymbol{s},\boldsymbol{a})$ **+ (1 -** $\boldsymbol{\alpha}$**)** $\boldsymbol{Q(s_t, at)}$

Reinforcement Learning in the Wild

- Reinforcement learning can be used to solve *large* problems, e.g.
  - Backgammon: ***1020 states***
  - Computer Go: **10170 states**
  - Helicopter: **continuous state space**
- How can we scale up the model-free methods for *prediction* and *control ?*

Reinforcement Learning in the Wild



- How many states we have in this game?

Curse of dimensionality in RL

Problem:

- State space is usually large, sometimes continuous.
- How about action space ?

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

What should we do?

Curse of dimensionality in RL

Problem:

- State space is usually large, sometimes continuous.
- Action space ?

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

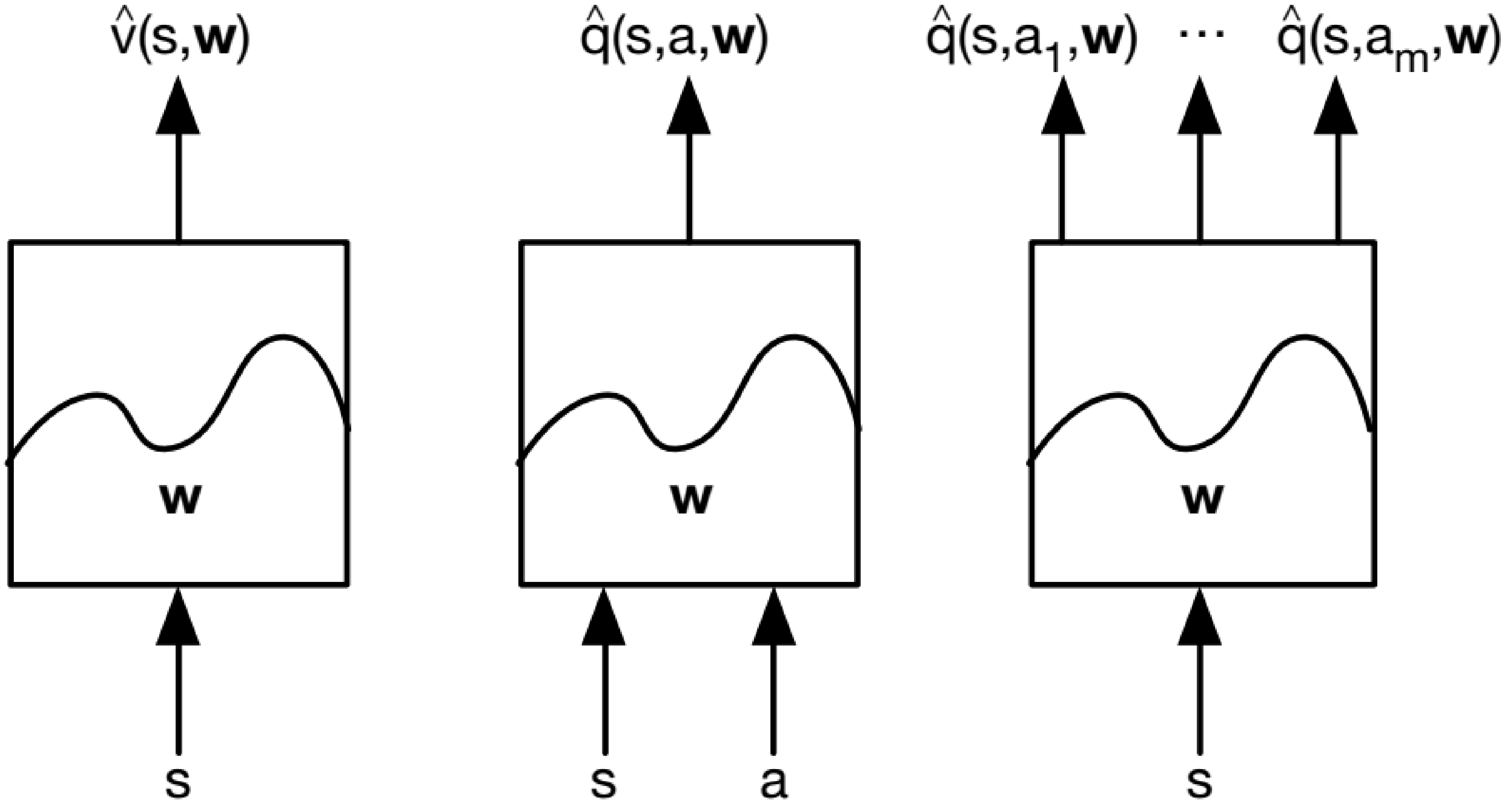
Solutions:

- quantize state space
- approximate agent with a function

Value Function Approximation

- So far we have represented value function by a *lookup table*
  - Every state s has an entry $V(s)$
  - Or every state-action pair $\langle s, a \rangle$ has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with *function approximation*
    - $v'(s, \mathbf{w}) \approx v_\pi(s)$
    - $q'(s, a, \mathbf{w}) \approx q_\pi(s, a)$
  - *Generalize* from seen states to unseen states
  - *Update* parameter $\mathbf{w}$ using MC or TD learning

Types of Value Function Approximation



Which class of function to choose?

- There are many function approximators, e.g.
  - *Linear combinations of features*
  - *Neural network*
  - Decision tree
  - Nearest neighbor
  - Fourier / wavelet bases
  - ...

Gradient Descent

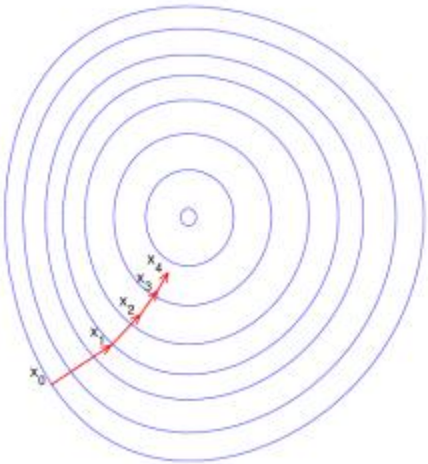
- Let $J(\mathbf{w})$ be a differentiable function of parameter vector $\mathbf{w}$
- Define the *gradient* of $J(\mathbf{w})$ to be

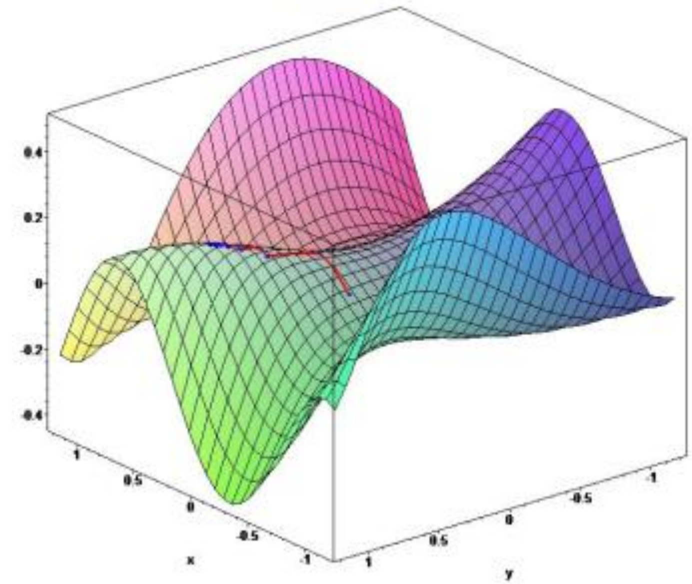
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

- To find a local minimum of $J(\mathbf{w})$:
  - Adjust $\mathbf{w}$ in direction of -ve gradient

$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

where α is a step-size parameter





SGD for Value Function approximation

- Goal: find parameter vector **w** minimizing mean-squared error between approximate value $v'(s, \mathbf{w})$ and true value $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

Feature vectors

- Represent state by a *feature vector*

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- For example:
  - Distance of robot from key landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

Linear Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S) \mathbf{w}_j$$

- Objective function is quadratic in parameters **w**

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

- Stochastic gradient descent converges on *global* optimum
  - Update rule is particularly simple:

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$

$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\mathbf{x}(S)$$

- Update = *step-size* × *prediction error* × *feature value*

Table Lookup Features

- Table lookup is a special case of linear value function approximation
  - Using *table lookup features*

$$\mathbf{x}^{table}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix}$$

- Parameter vector **w** gives value of each individual state

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

Incremental Prediction Algorithms

- Have assumed true value function $v_\pi(s)$ given by supervisor
  - But in RL there is no supervisor, only rewards
  - In practice, we substitute a *target* for $v_\pi(s)$
- For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

- For TD the target is the TD target:

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

Question

Value is good but how to play? How to choose actions?

Action Value function approximation

- Approximate the action-value function

$$q'(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimize mean-squared error between approximate action-value $q'(S, A, \mathbf{w})$ and true action-value $q_\pi(S, A)$

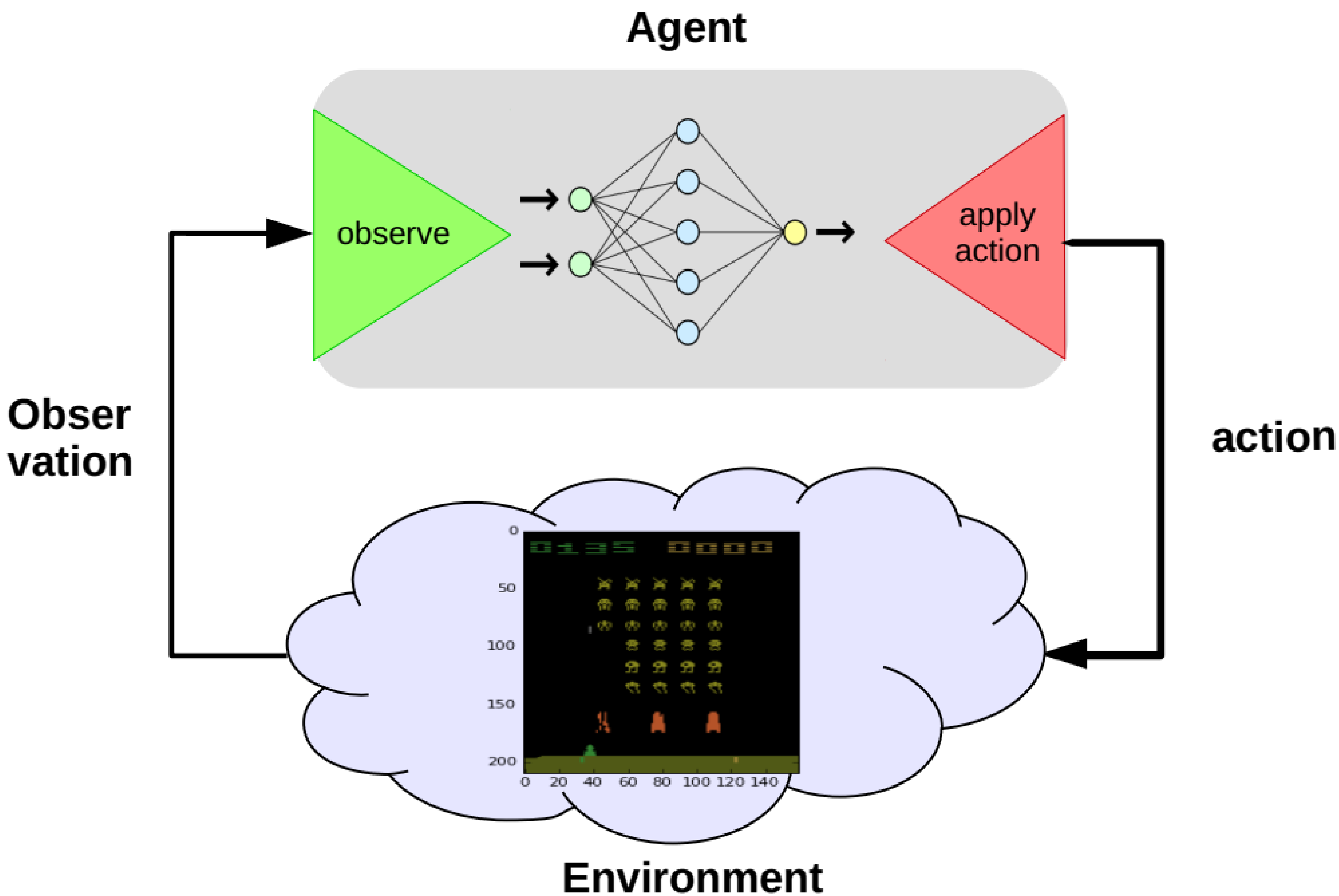
$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right]$$

- Use stochastic gradient descent to find a local minimum

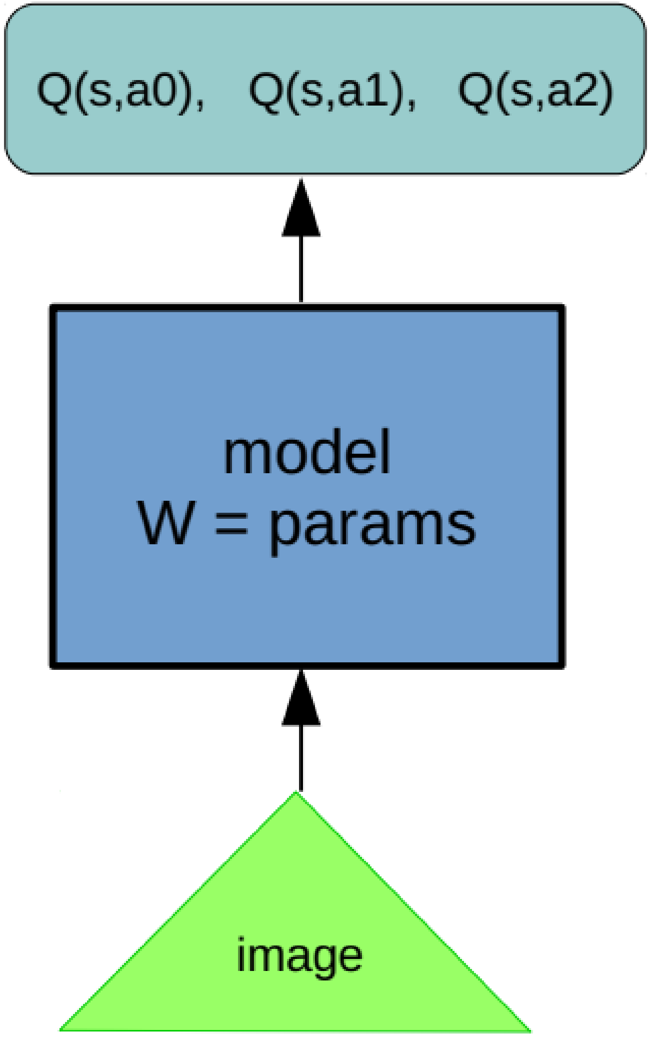
$$-\frac{1}{2} \nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

Atari again


Agent
observe
apply
action
Obser
vation
action
0135 0000
0
50
100
150
200
0 20 40 60 80 100 120 140
Environment

Approximate Q-learning



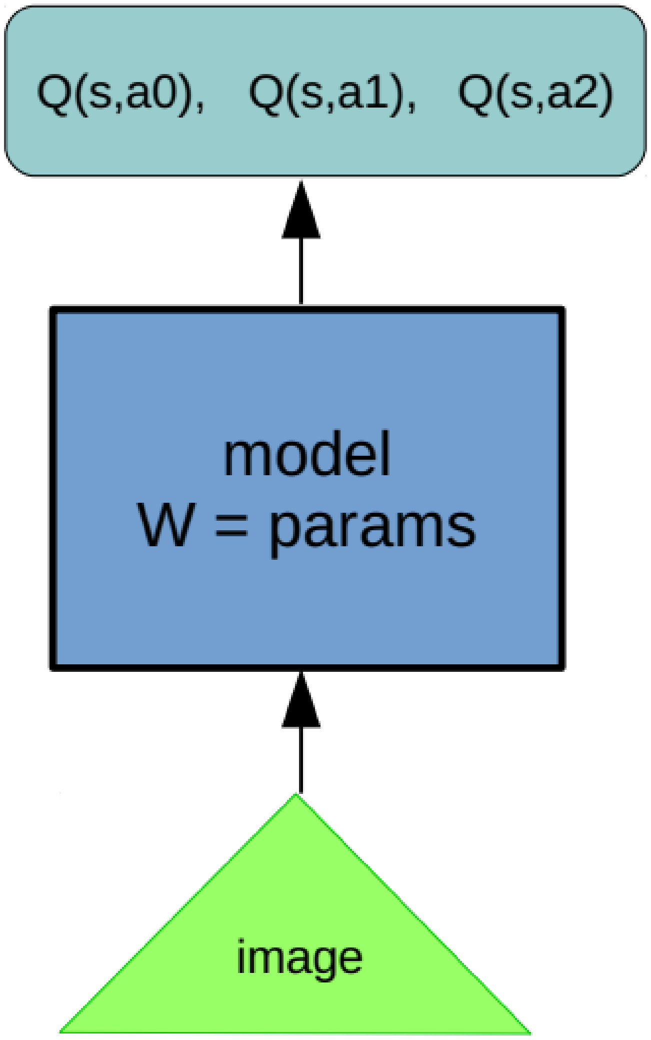
- Initialize **W.**
- Cycle:
  - Sample **<s, a, r, s'>** from environment
  - Compute $\widehat{Q}(s,a) = r(s,a) + \gamma \max_{a_i} Q(s', ai)$
  - Objective:

    $$L = [Q(s_t, at) - \widehat{Q}(s_t, at)]^2$$

  - SGD Update:

    $$W_{t+1} = W_t - \alpha \frac{\partial L}{\partial wt}$$

Approximate SARSA



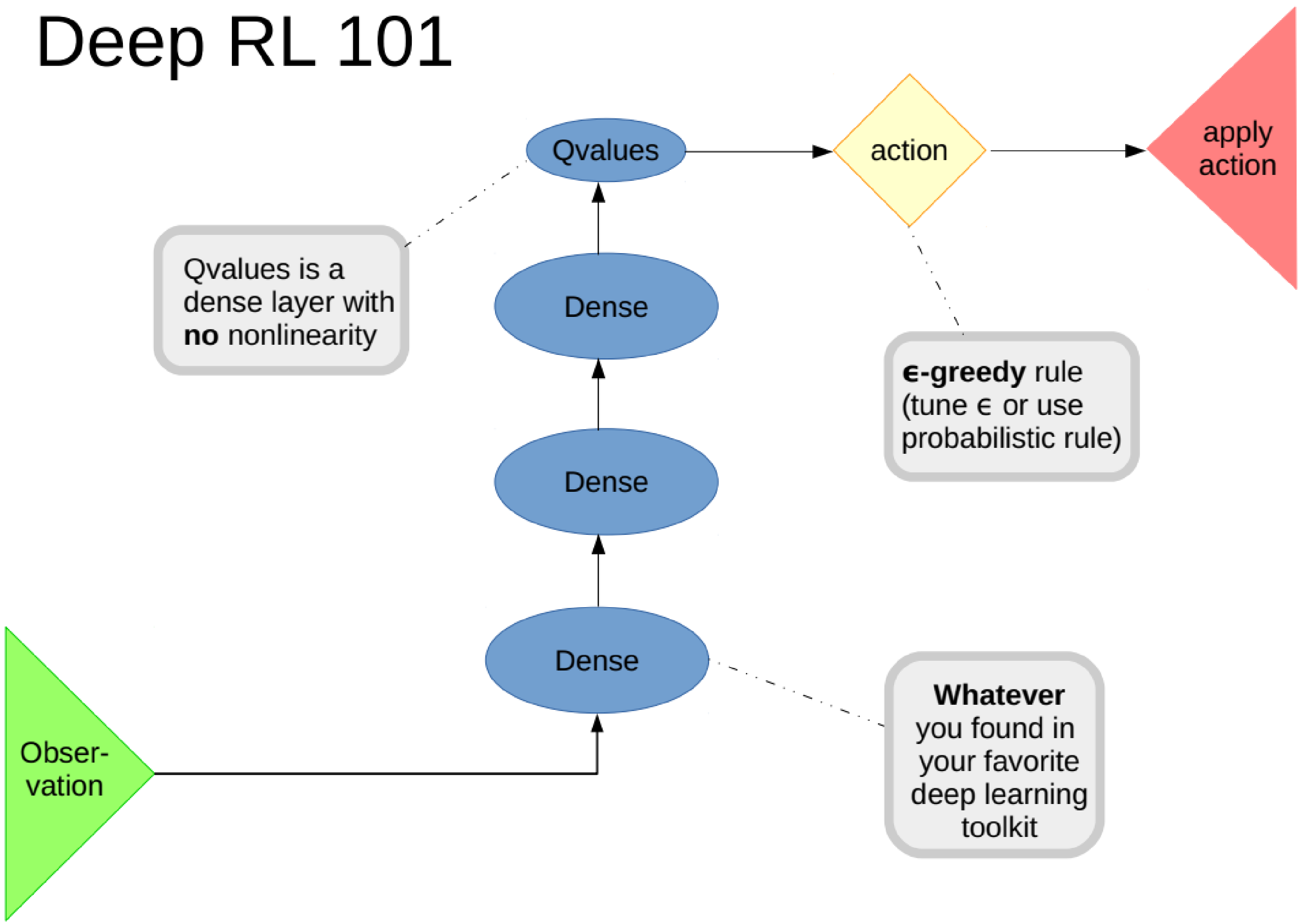
- Initialize **W.**
- Cycle:
  - Sample **<s, a, r, s', a'>** from environment
  - Compute $\hat{Q}(s, a) = r(s, a) + \gamma Q(s', a')$
  - Objective:

    $$L = [Q(s_t, at) - \hat{Q}(s_t, at)]^2$$

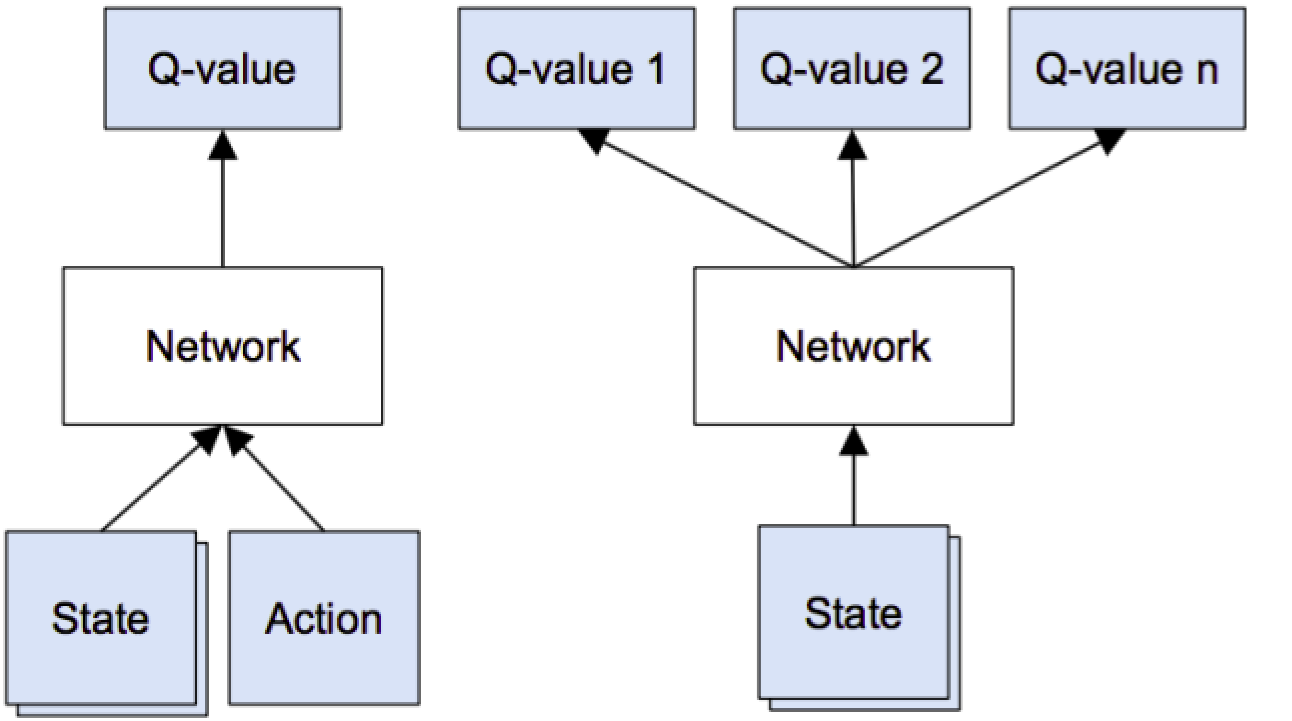
  - SGD Update:

    $$W_{t+1} = W_t - \alpha \frac{\partial L}{\partial wt}$$

RL Mechanics



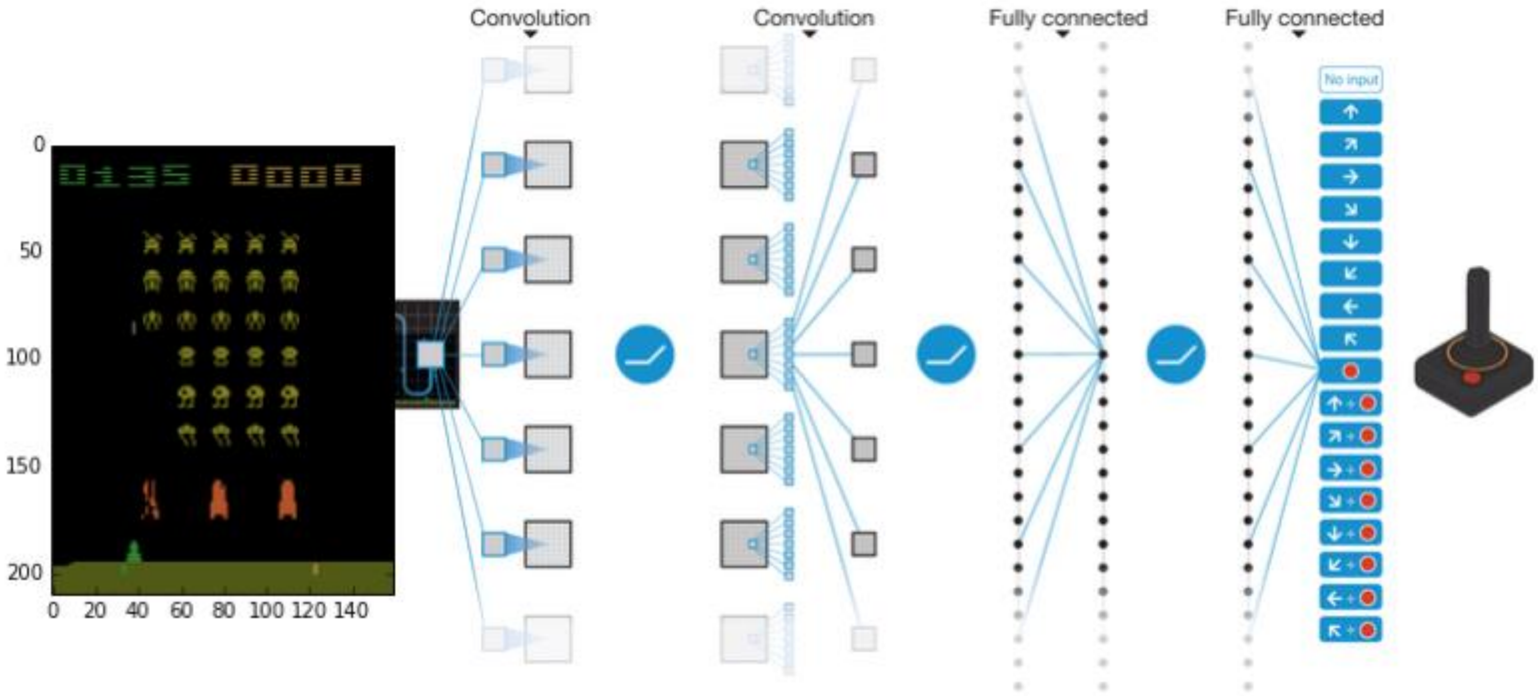
Architectures



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

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

From theory to practice: DQN case



Playing Atari with Deep Reinforcement Learning

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

DQN: under the hood

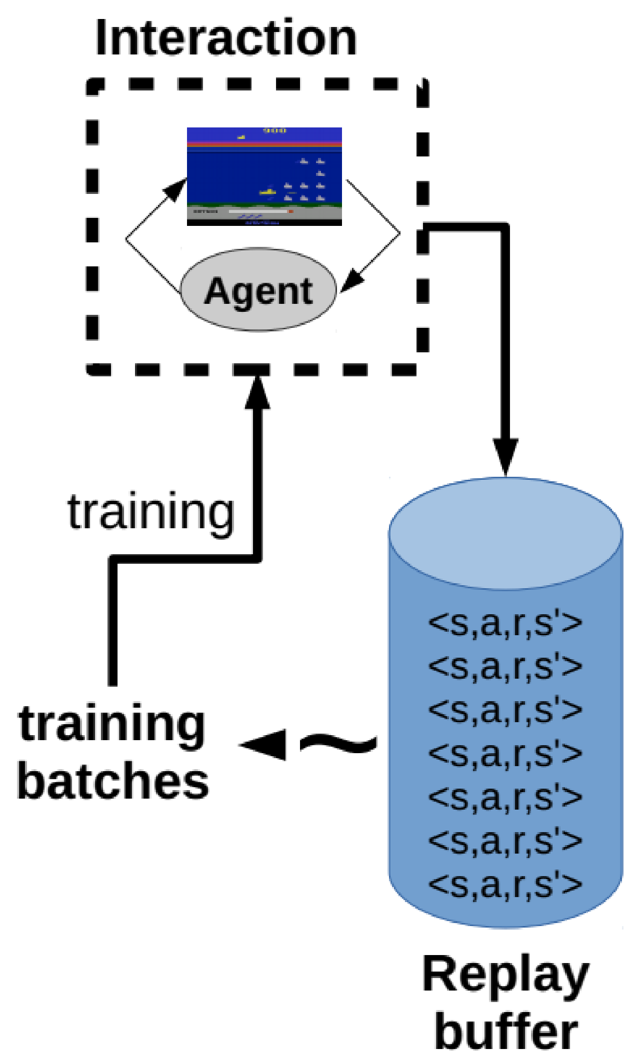
- DQN uses experience replay and fixed Q-targets:
  - Take action $\boldsymbol{a_t}$ according to *e*-greedy policy
  - Store transition $(\boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{r_{t+1}}, \boldsymbol{s_{t+1}})$ in replay memory ***D***
  - Sample random mini-batch of transitions $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s'})$ from ***D***
  - Compute Q-learning targets w.r.t. old, fixed parameters $\boldsymbol{w}^-$
  - Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

  - Using variant of stochastic gradient descent

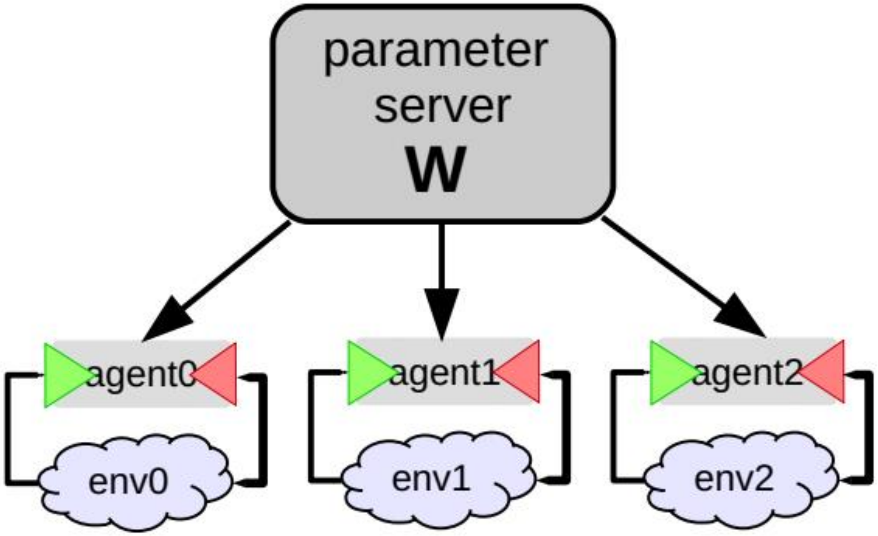
Experience Replay

- **Idea:** store several past interactions <*s*,*a*,*r*,*s'*>
- Train on random subsamples
- **Any +/-** ?



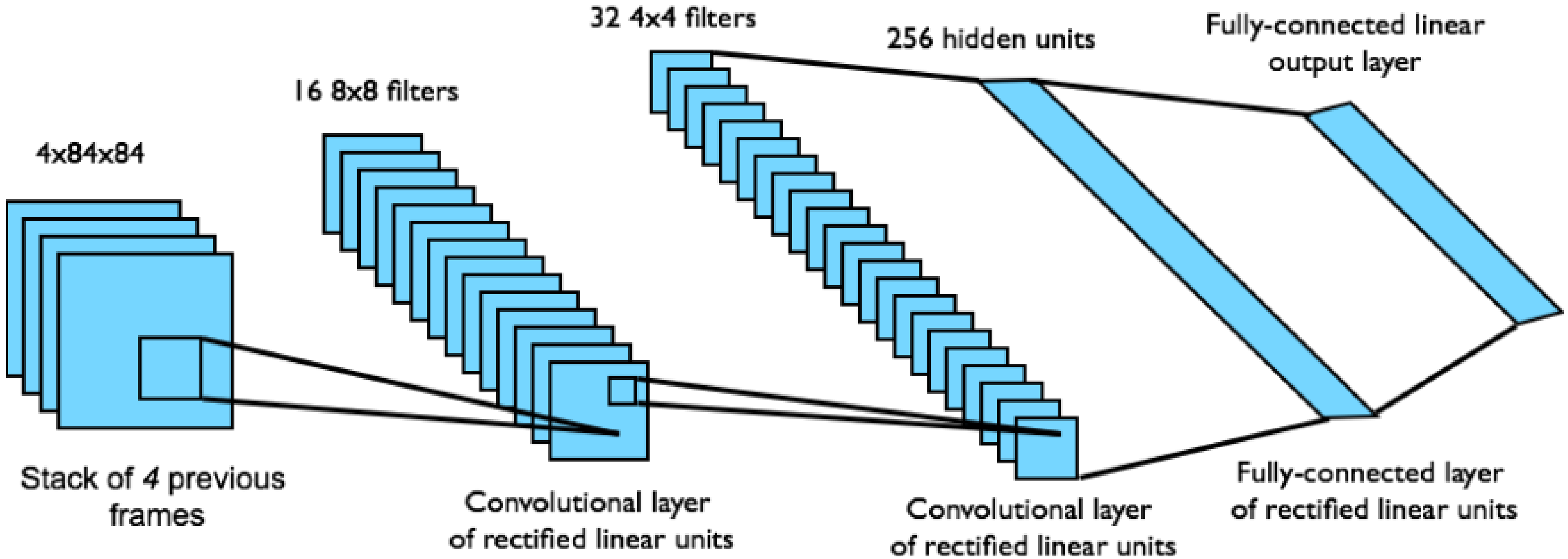
Experience replay: alternative

- Several agents with 'shared' weights plays different environments

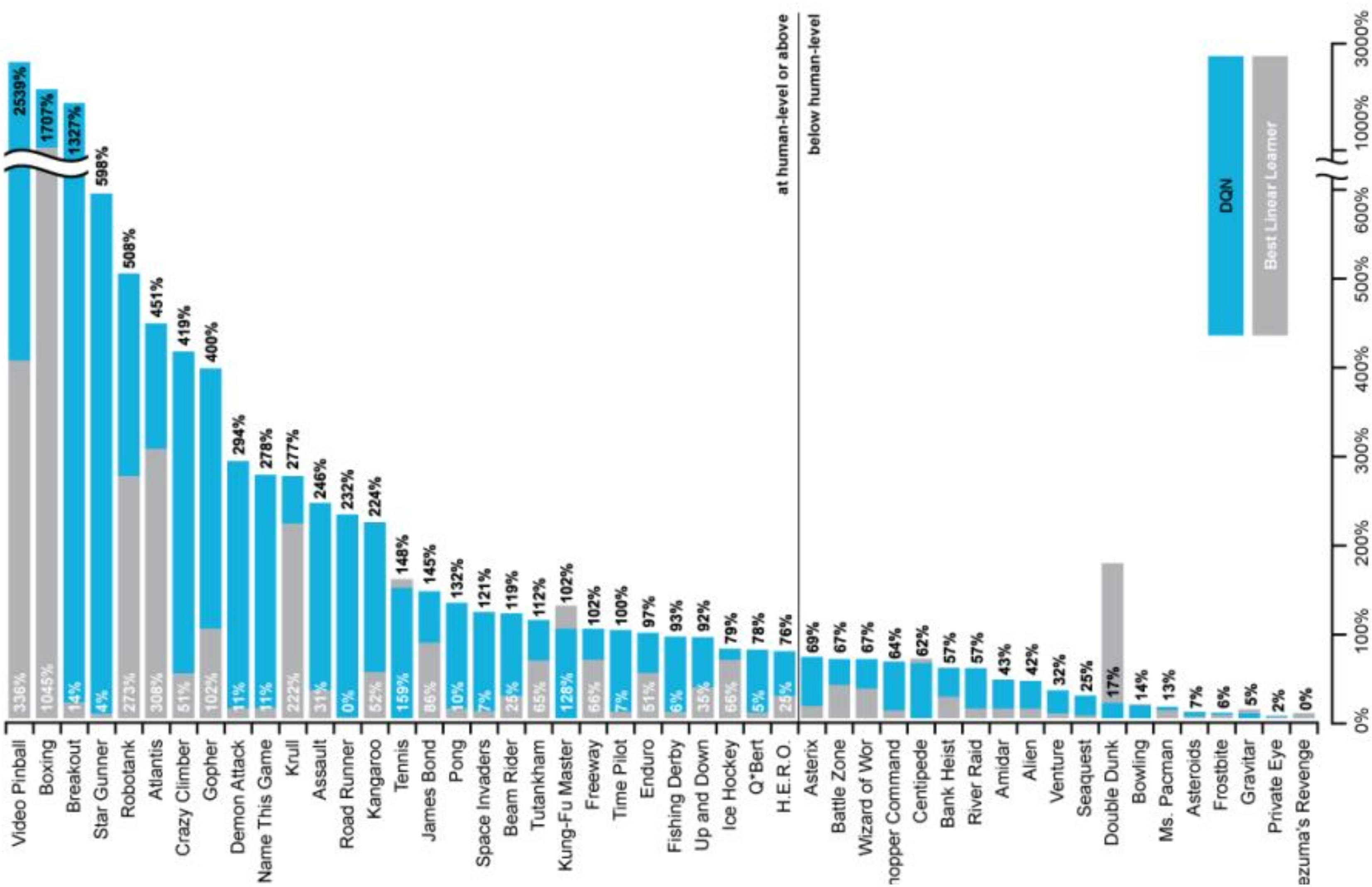


DQN: Atari

- End-to-end learning of values ***Q*(*s*, *a*)** from pixels ***s***
- Input state *s* is stack of raw pixels from last **4** frames
- Output is *Q*(*s*, *a*) for **18** joystick/button positions
- Reward is change in score for that step



DQN results on Atari



DQN: An Ablation study

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

Questions?