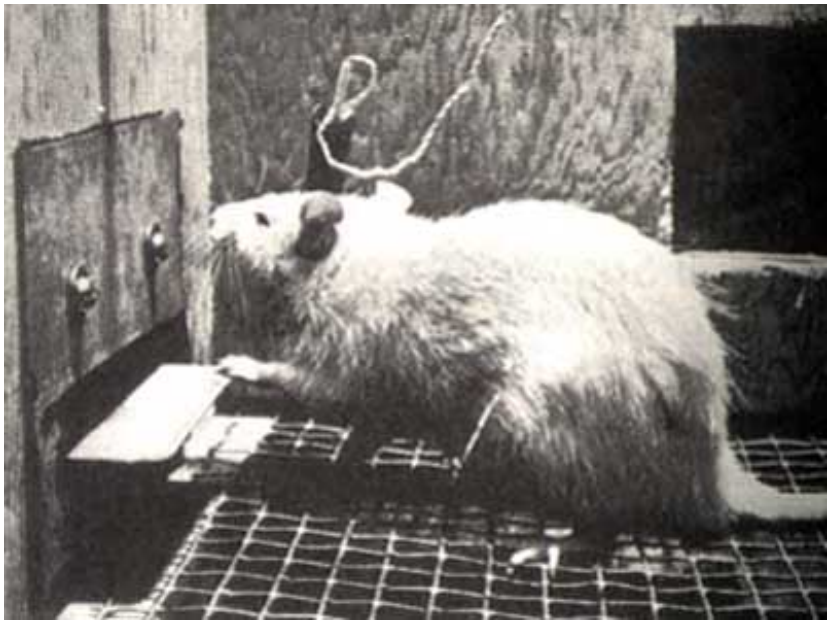
Introduction to deep reinforcement learning

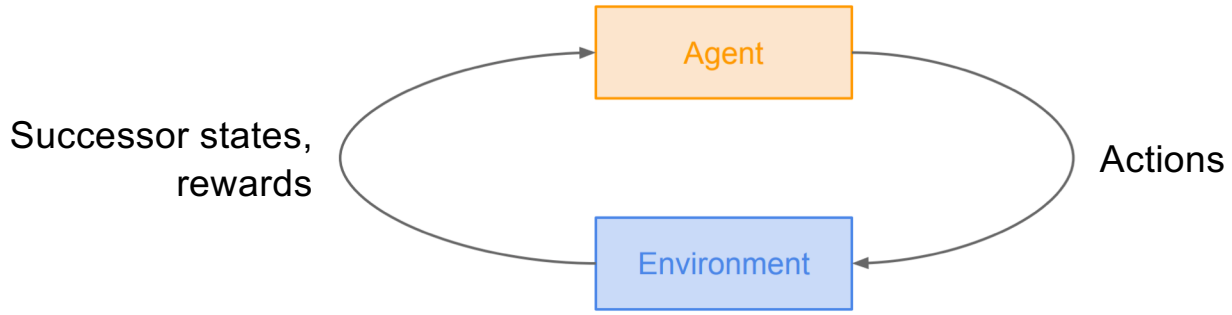


Outline

- Introduction to reinforcement learning
- Markov Decision Process (MDP) formalism
- The Bellman equation
- Q-learning
- Deep Q networks (DQN)
- Extensions
  - Double DQN
  - Dueling DQN

Reinforcement learning (RL)

- Setting: agent that can take *actions* affecting the *state* of the environment and observe occasional *rewards* that depend on the state
- Goal: learn a *policy* (mapping from states to actions) to maximize expected reward over time



RL vs. supervised learning

- **Reinforcement learning loop**
  - From state s, take action a determined by *policy* $\pi(s)$
  - Environment selects next state s' based on *transition model* $P(s'|s,a)$
  - Observe s' and reward $r(s')$, update policy

- **Supervised learning loop**
  - Get input x_i sampled i.i.d. from data distribution
  - Use model with parameters w to predict output y
  - Observe target output y_i and loss $l(w, x_i, y_i)$
  - Update w to reduce loss: $w \leftarrow w - \eta \nabla l(w, x_i, y_i)$

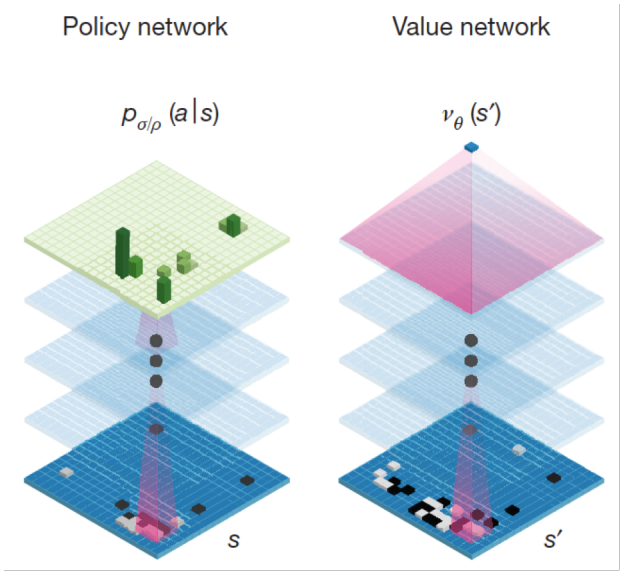
RL vs. supervised learning

- **Reinforcement learning**
  - Agent's actions affect the environment and help to determine next observation
  - Rewards may be sparse
  - Rewards are not differentiable w.r.t. model parameters

- **Supervised learning**
  - Next input does not depend on previous inputs or agent's predictions
  - There is a supervision signal at every step
  - Loss is differentiable w.r.t. model parameters

Example applications of deep RL

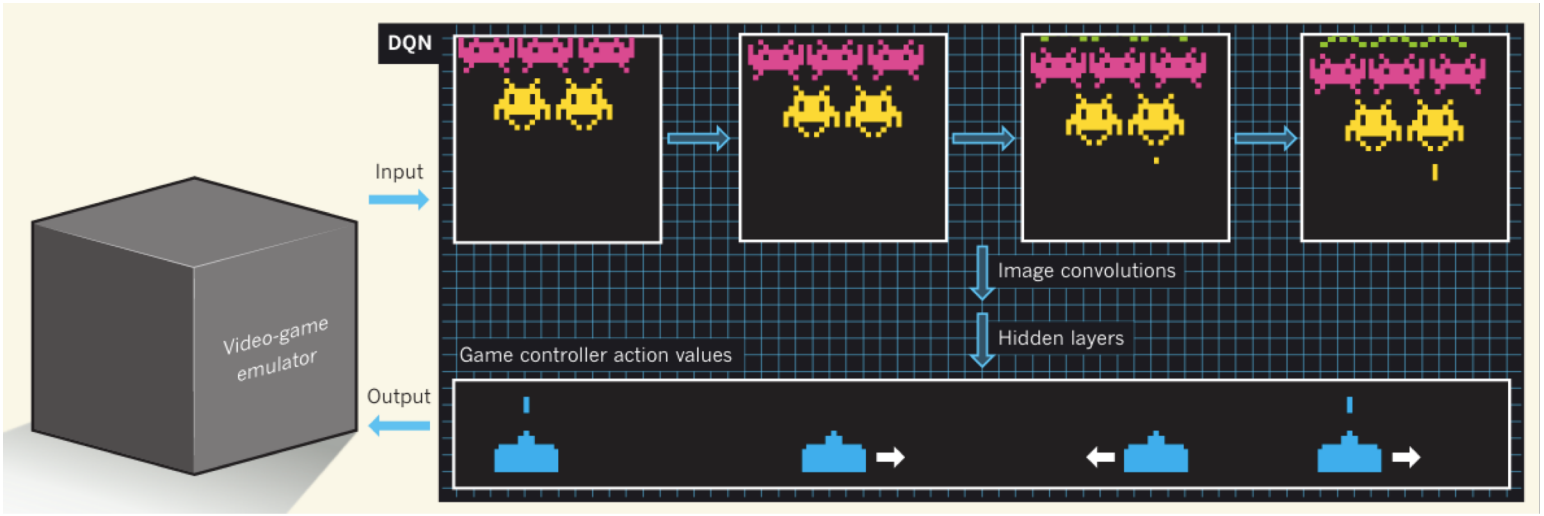
- AlphaGo and AlphaZero



https://deepmind.com/research/alphago/

Example applications of deep RL

- Playing video games



Video

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Human-level control through deep reinforcement learning, *Nature* 2015

Example applications of deep RL

- Sensorimotor learning

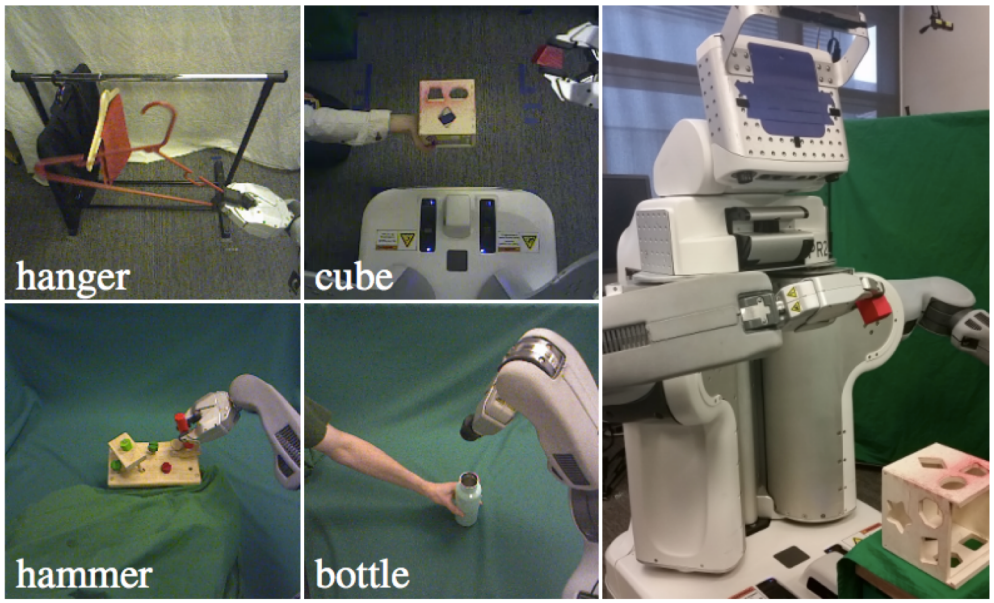


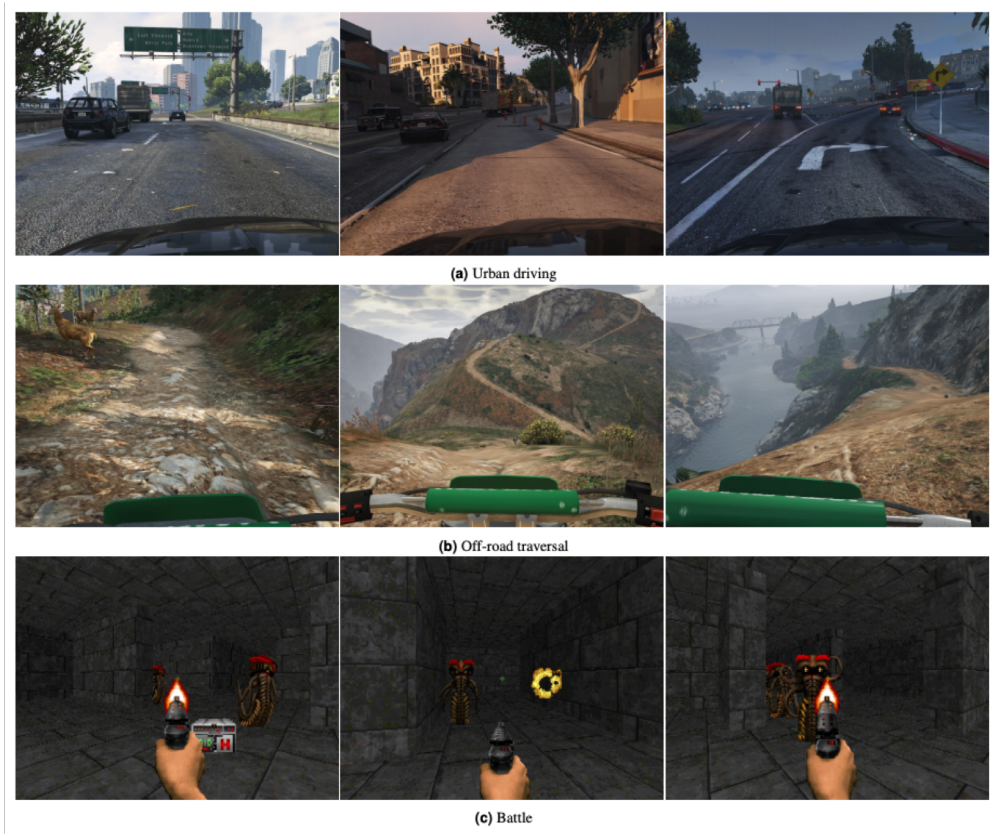
Fig. 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).

Video

S. Levine, C. Finn, T. Darrell and P. Abbeel, End-to-End Training of Deep Visuomotor Policies, JMLR 2016

Example applications of deep RL

- Sensorimotor learning



(a) Urban driving

(b) Off-road traversal

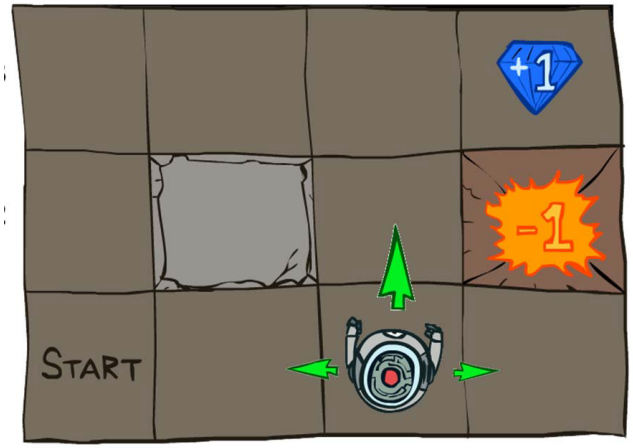
(c) Battle

B. Zhou, P. Krähenbühl, and V. Koltun, Does Computer Vision Matter for Action? Science Robotics, 4(30), 2019

Formalism: Markov Decision Processes

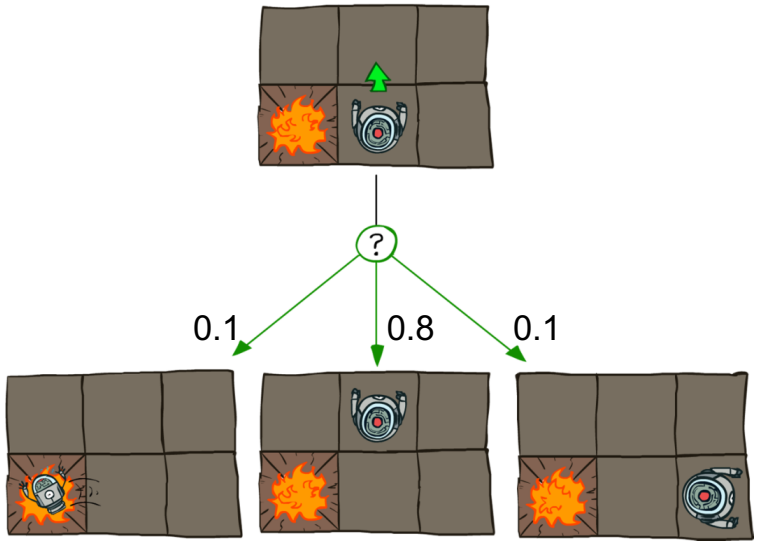
- Components:
  - **States** s, beginning with initial state s_0
  - **Actions** a
  - **Transition model** $P(s' \mid s, a)$
    - *Markov assumption*: the probability of going to s' from s depends only on s and a and not on any other past actions or states
  - **Reward function** $r(s)$
- **Policy** $\pi(s)$: the action that an agent takes in any given state
  - The "solution" to an MDP

Example MDP: Grid world



$r(s) = -0.04$ for every non-terminal state

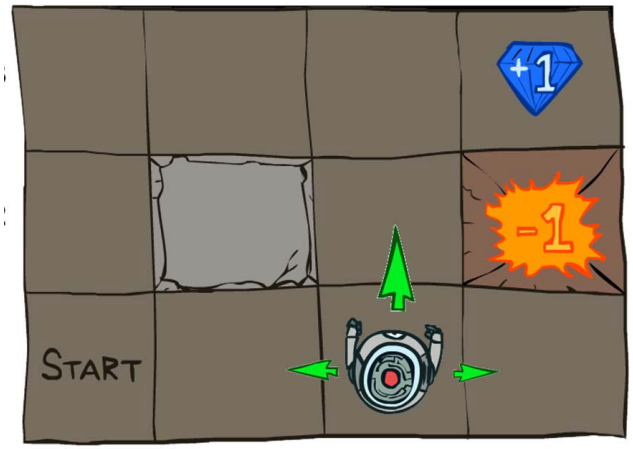
Transition model:

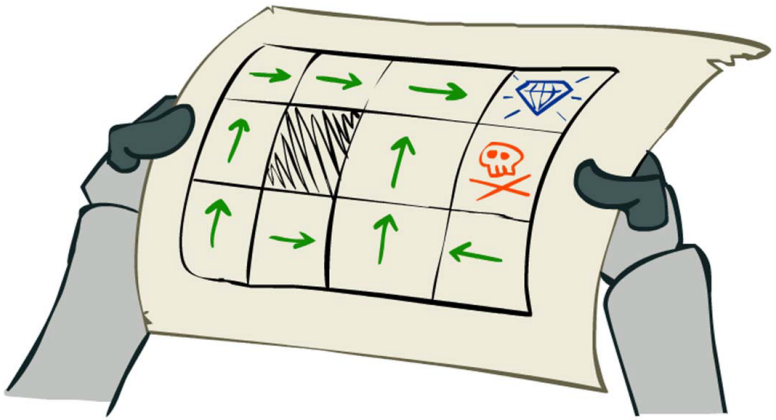


Source: P. Abbeel and D. Klein

Example MDP: Grid world

- Goal: find the best policy

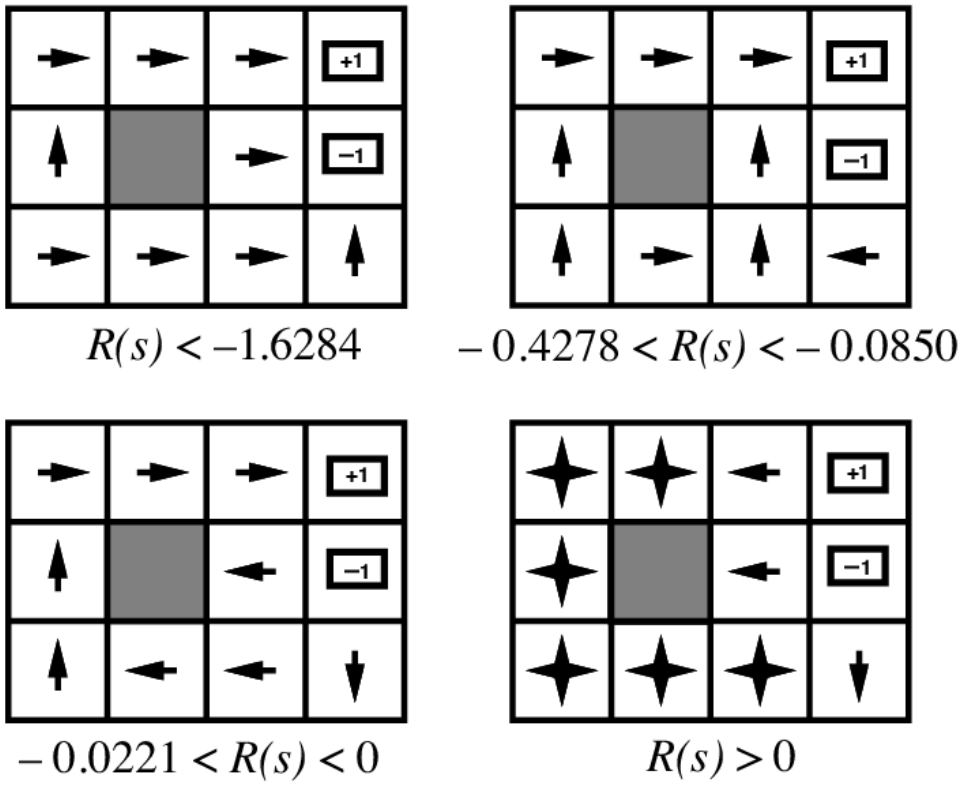




Source: P. Abbeel and D. Klein

Example MDP: Grid world

- Optimal policies for various values of $r(s)$:



Rewards of state sequences

- Suppose that following policy π starting in state s_0 leads to a sequence $s_0, s_1, s_2, \ldots$
- The *cumulative reward* of the sequence is $\sum_{t \geq 0} r(s_t)$
- **Problem:** state sequences can vary in length or even be infinite
- **Solution:** redefine cumulative reward as sum of rewards *discounted* by a factor γ

Image source: P. Abbeel and D. Klein

Discounting

- Discounted cumulative reward:

$$r(s_0) + \gamma\, r(s_1) + \gamma^2 r(s_2) + \gamma^3 r(s_3) + \cdots$$
$$= \sum_{t \geq 0} \gamma^t\, r(s_t)\,, \qquad 0 < \gamma \leq 1$$

- Sum is bounded by $\frac{r_{\max}}{1-\gamma}$
- Helps algorithms converge

Value function

- The *value function* $V^{\pi}(s)$ of a state s w.r.t. policy π is the expected cumulative reward of following that policy starting in s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t \, r(s_t) \mid s_0 = s, \pi\right]$$

with $a_t = \pi(s_t), s_{t+1} \sim P\left(\cdot \mid s_t, a_t\right)$

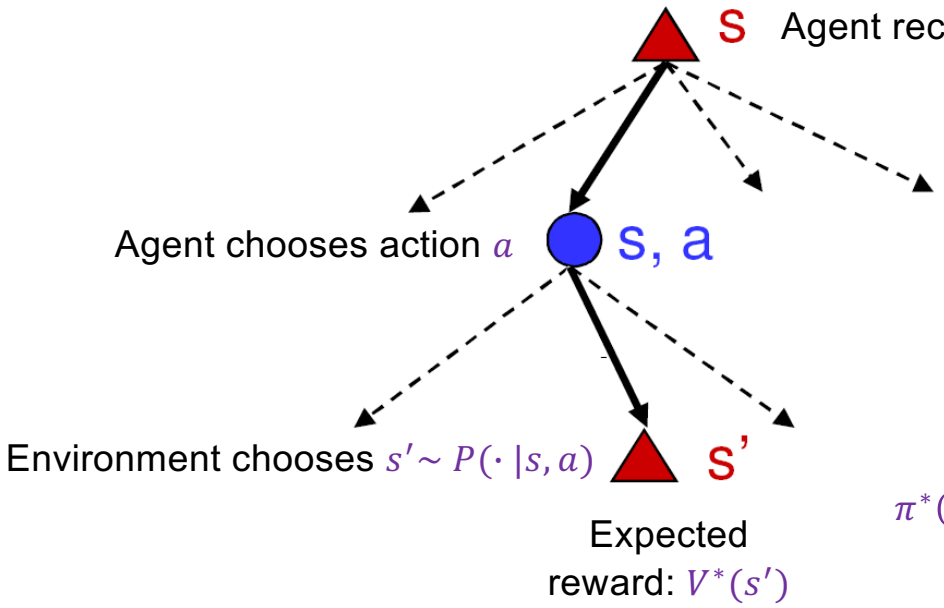
- The *optimal value* of a state is the value achievable by following the best possible policy:

$$V^{*}(s) = \max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t \, r(s_t) \mid s_0 = s, \pi\right]$$

The Bellman equation

- Recursive relationship between optimal values of successive states:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V^*(s')$$



Expected value for action a:

$$\sum_{s'} P(s'|s,a)V^*(s')$$

Optimal policy:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^*(s')$$

The Bellman equation

- Recursive relationship between optimal values of successive states:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$$

Reward in current state

Discounted expected future reward assuming agent follows the optimal policy

Outline

- Introduction to reinforcement learning
- Markov Decision Process (MDP) formalism
- The Bellman equation
- Q-learning

Q-learning

- Optimal policy in terms of the state value function:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^*(s')$$

  - To use this in practice, we need to know the transition model

- It is more convenient to define the value of a *state-action pair*:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t\, r(s_t)\,|s_0 = s, a_0 = a, \pi\right]$$

Q-value function

- The *optimal* Q-value:

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t\, r(s_t) \mid s_0 = s, a_0 = a, \pi\right]$$

- What is the relationship between $V^*(s)$ and $Q^*(s,a)$?

$$V^*(s) = \max_a Q^*(s,a)$$

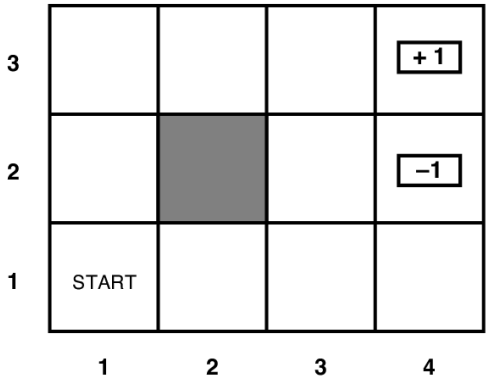
- What is the optimal policy?

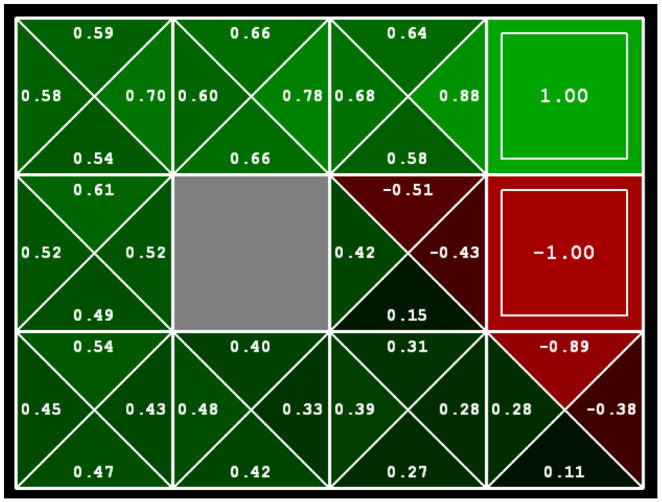
$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

Q-value function

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t\, r(s_t) \mid s_0 = s, a_0 = a, \pi\right]$$

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$





Bellman equation for Q-values

$$V^*(s) = \max_a Q^*(s,a)$$

- Regular Bellman equation:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$$

- Bellman equation for Q-values:

$$Q^*(s,a) = r(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

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

Finding the optimal policy

- The Bellman equation is a constraint on Q-values of successive states:

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

- We could think of $Q^*(s,a)$ as a table indexed by states and actions, and try to solve the system of Bellman equations to fill in the unknown values of the table
- **Problem:** state spaces for interesting problems are huge
- **Solution:** approximate Q-values using a parametric function:

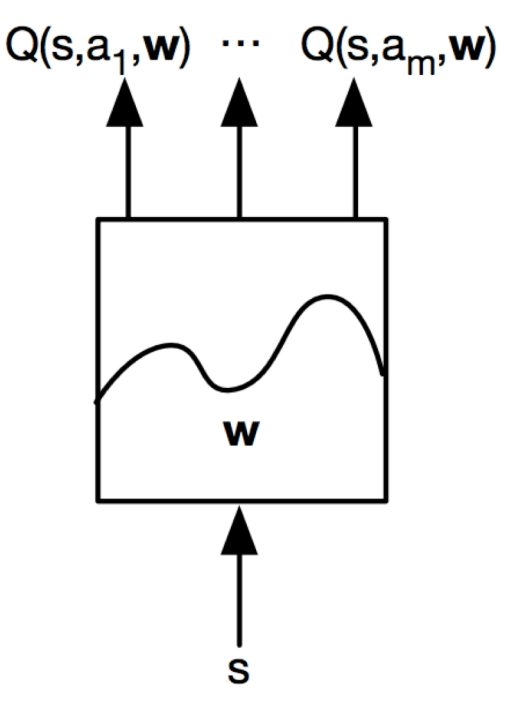
$$Q^*(s,a) \approx Q_w(s,a)$$

Outline

- Introduction to reinforcement learning
- Markov Decision Process (MDP) formalism and classical Bellman equation
- Q-learning
- Deep Q networks

Deep Q-learning

- Train a deep neural network to estimate Q-values:



Source: D. Silver

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Human-level control through deep reinforcement learning, *Nature* 2015

Deep Q-learning

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

- Idea: at each iteration i of training, update model parameters w_i to "nudge" the left-hand side toward the right-hand "target":

$$y_i(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s',a')|s,a\right]$$

- Loss function:

$$L_i(w_i) = \mathbb{E}_{s,a \sim \rho}\left[(y_i(s,a) - Q_{w_i}(s,a))^2\right]$$

where ρ is a *behavior distribution*

Deep Q-learning

- Target: $y_i(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s', a') | s, a \right]$
- Loss: $L_i(w_i) = \mathbb{E}_{s,a \sim \rho} \left[(y_i(s,a) - Q_{w_i}(s,a))^2 \right]$

- Gradient update:

$$\nabla_{w_i} L(w_i) = \mathbb{E}_{s,a \sim \rho} \left[(y_i(s,a) - Q_{w_i}(s,a)) \nabla_{w_i} Q_{w_i}(s,a) \right]$$
$$= \mathbb{E}_{s,a \sim \rho, s'} \left[(r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s', a') - Q_{w_i}(s,a)) \nabla_{w_i} Q_{w_i}(s,a) \right]$$

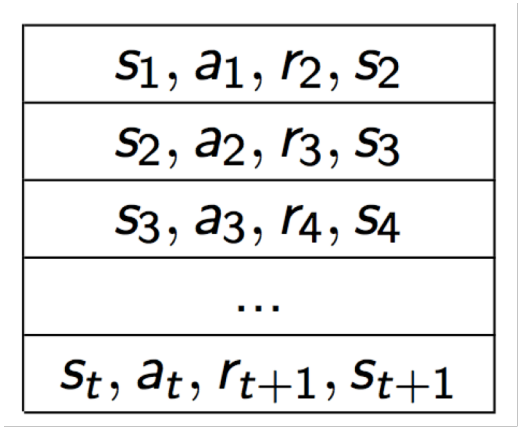
- SGD training: replace expectation by sampling *experiences* (s, a, s') using behavior distribution and transition model

Deep Q-learning in practice

- Training is prone to instability
  - Unlike in supervised learning, the targets themselves are moving!
  - Successive experiences are correlated and dependent on the policy
  - Policy may change rapidly with slight changes to parameters, leading to drastic change in data distribution
- Solutions
  - Freeze target Q network
  - Use *experience replay*

Experience replay

- At each time step:
  - Take action a_t according to *epsilon-greedy policy*
  - Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay memory buffer*
  - Randomly sample *mini-batch* of experiences from the buffer



Experience replay

- At each time step:
  - Take action a_t according to *epsilon-greedy policy*
  - Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay memory buffer*
  - Randomly sample *mini-batch* of experiences from the buffer
  - Update parameters to reduce loss:

$$L_i(w_i) = \mathbb{E}_{s,a,s'}\left[(r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s', a') - Q_{w_i}(s, a))^2\right]$$

Keep parameters of *target network* fixed, update every once in a while

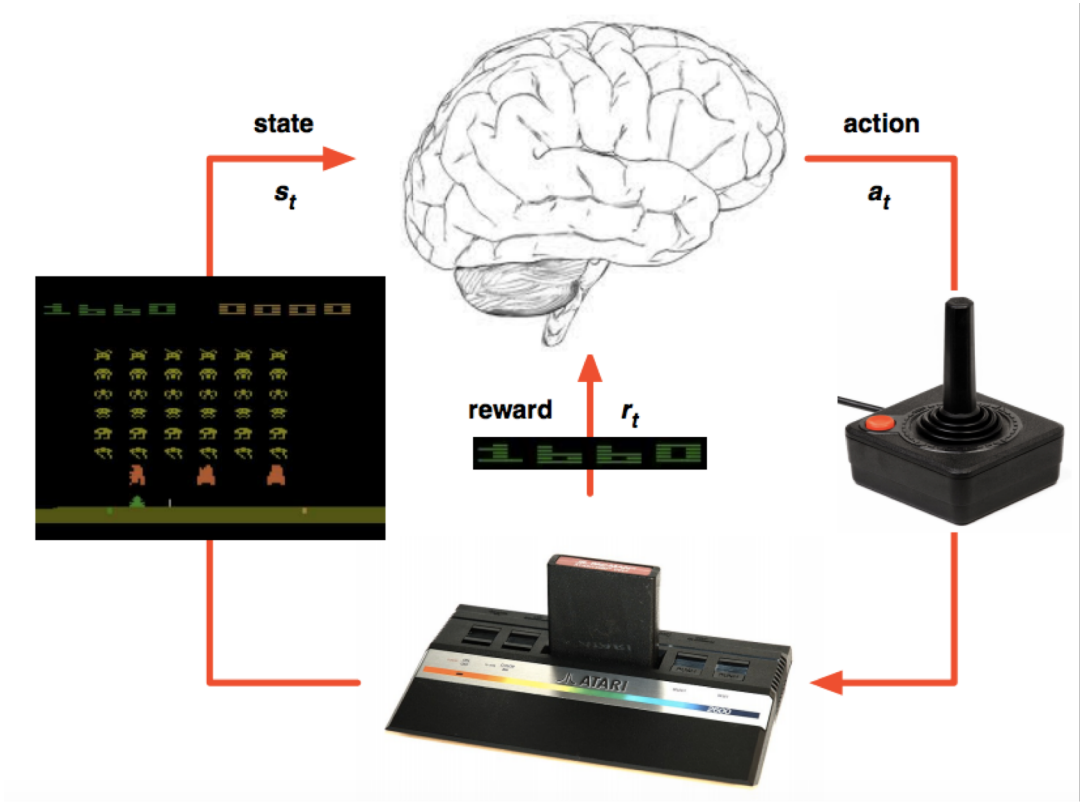
Experience replay

- At each time step:
  - Take action a_t according to *epsilon-greedy policy*
  - Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay memory buffer*
  - Randomly sample *mini-batch* of experiences from the buffer
  - Update parameters to reduce loss:

$$L_i(w_i) = \mathbb{E}_{s,a,s'}\left[(r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s', a') - Q_{w_i}(s, a))^2\right]$$

- Prioritized experience replay (Schaul et al., 2016):
  - Increase the probability of sampling transitions whose Q-values are farther from the target

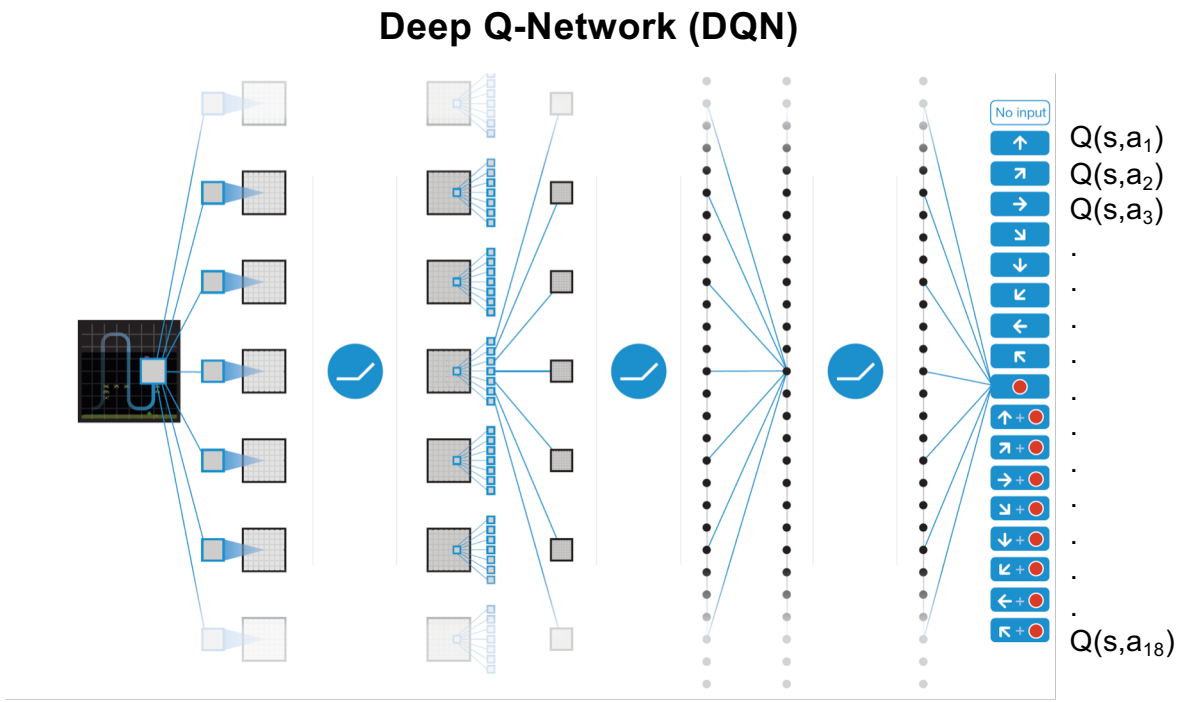
Deep Q-learning in Atari



V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller,
Human-level control through deep reinforcement learning, *Nature* 2015

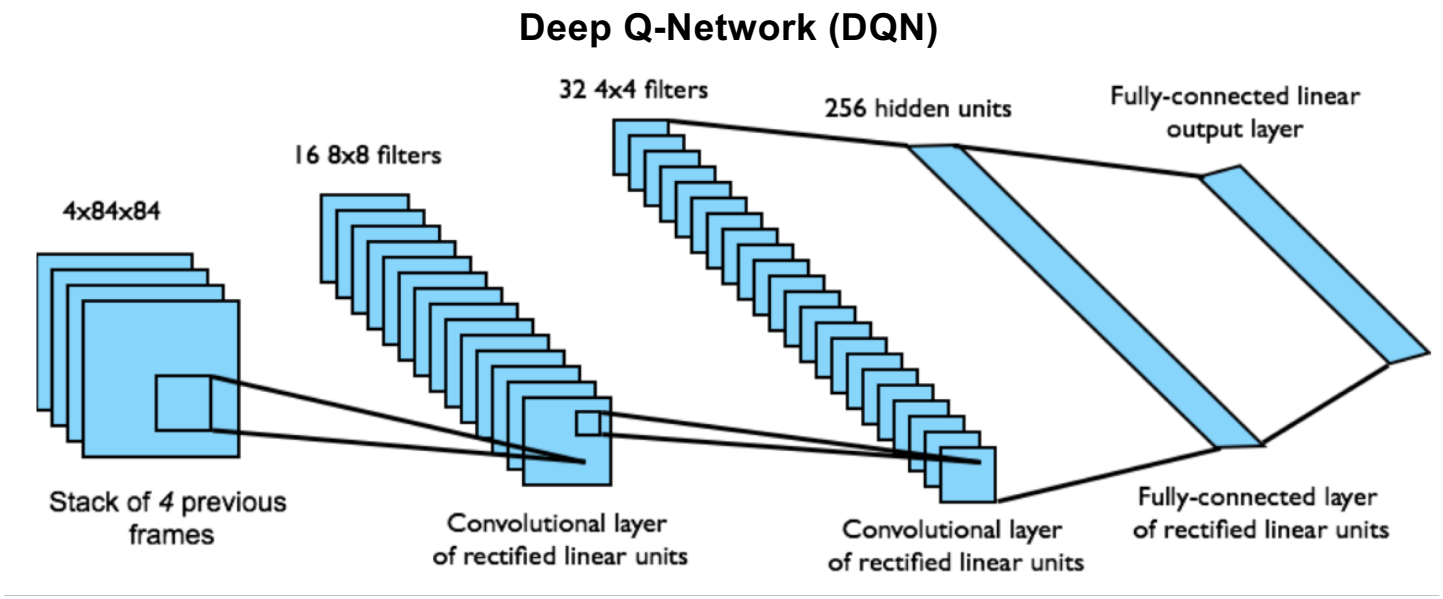
Deep Q-learning in Atari

- End-to-end learning of $Q(s,a)$ from pixels s
- Output is $Q(s,a)$ for 18 joystick/button configurations
- Reward is change in score for that step

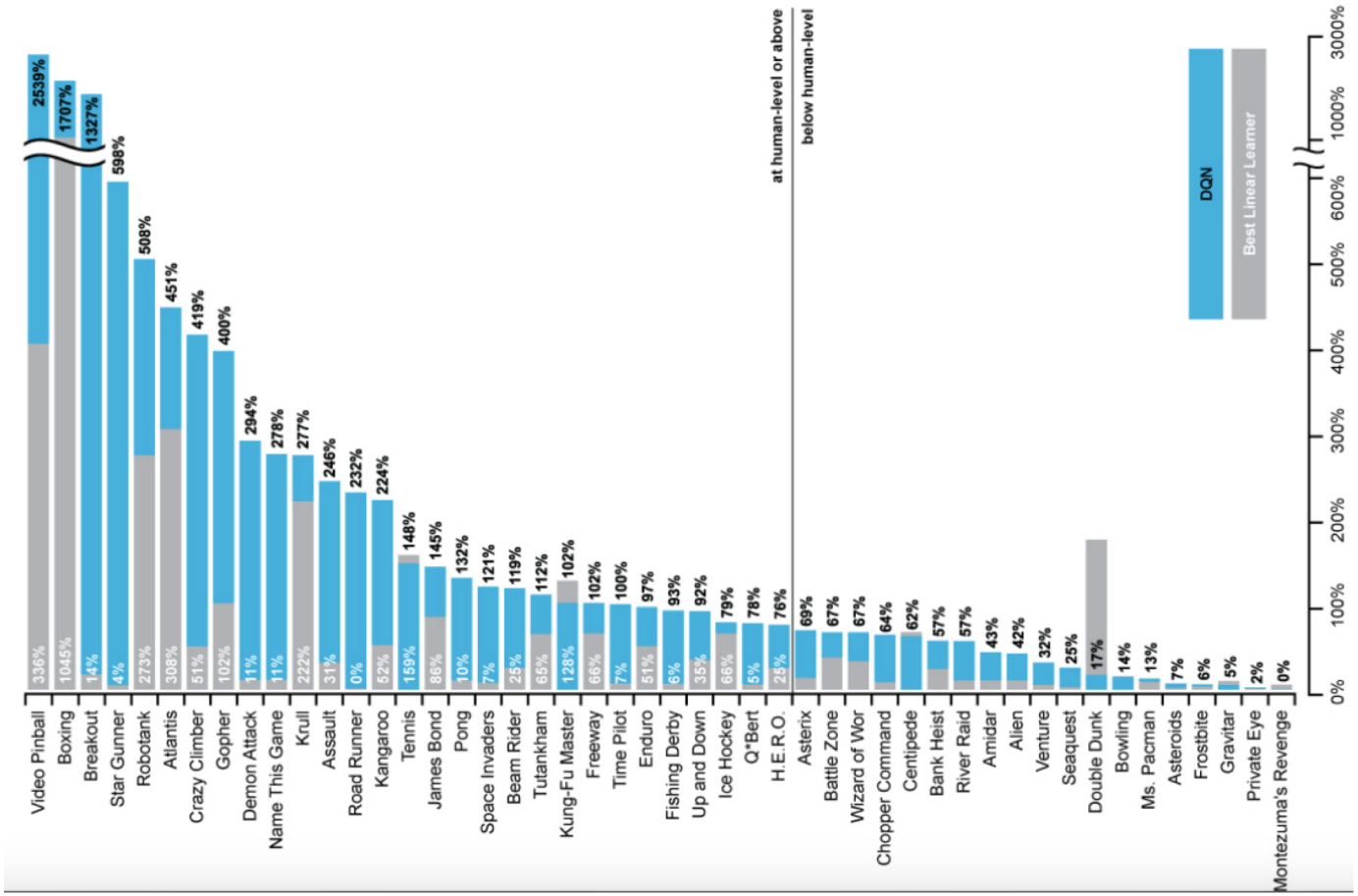


Deep Q-learning in Atari

- Input state is stack of raw pixels (grayscale) from last 4 frames
- Network architecture and hyperparameters fixed for all games



Deep Q-learning in Atari



Breakout demo

https://www.youtube.com/watch?v=TmPfTpjtdgg

Outline

- Introduction to reinforcement learning
- Markov Decision Process (MDP) formalism and classical Bellman equation
- Q-learning
- Deep Q networks
- Extensions
  - Double DQN
  - Dueling DQN

Extension: Double Q-learning

- Max operator in standard Q-learning is used both to select and evaluate an action, leading to systematic over-estimation of Q-values
- Modification: *select* action using the online (current) network, but *evaluate* Q-value using the target network
- Regular DQN target:

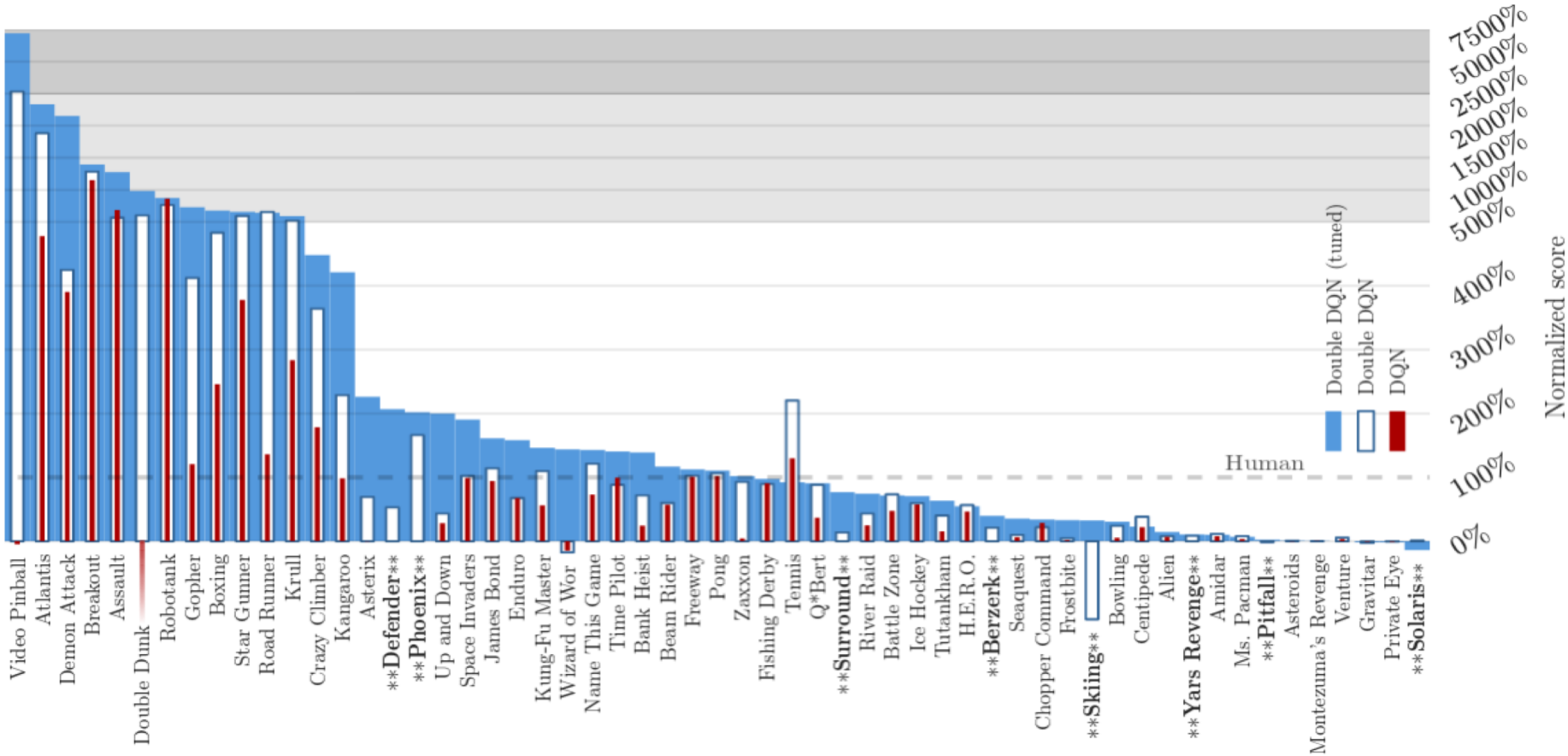
$$y_i(s,a) = r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s',a')$$

- Double DQN target:

$$y_i(s,a) = r(s) + \gamma\, Q_{w_{i-1}}\left(s', \operatorname{argmax}_{a'} Q_{w_i}(s',a')\right)$$

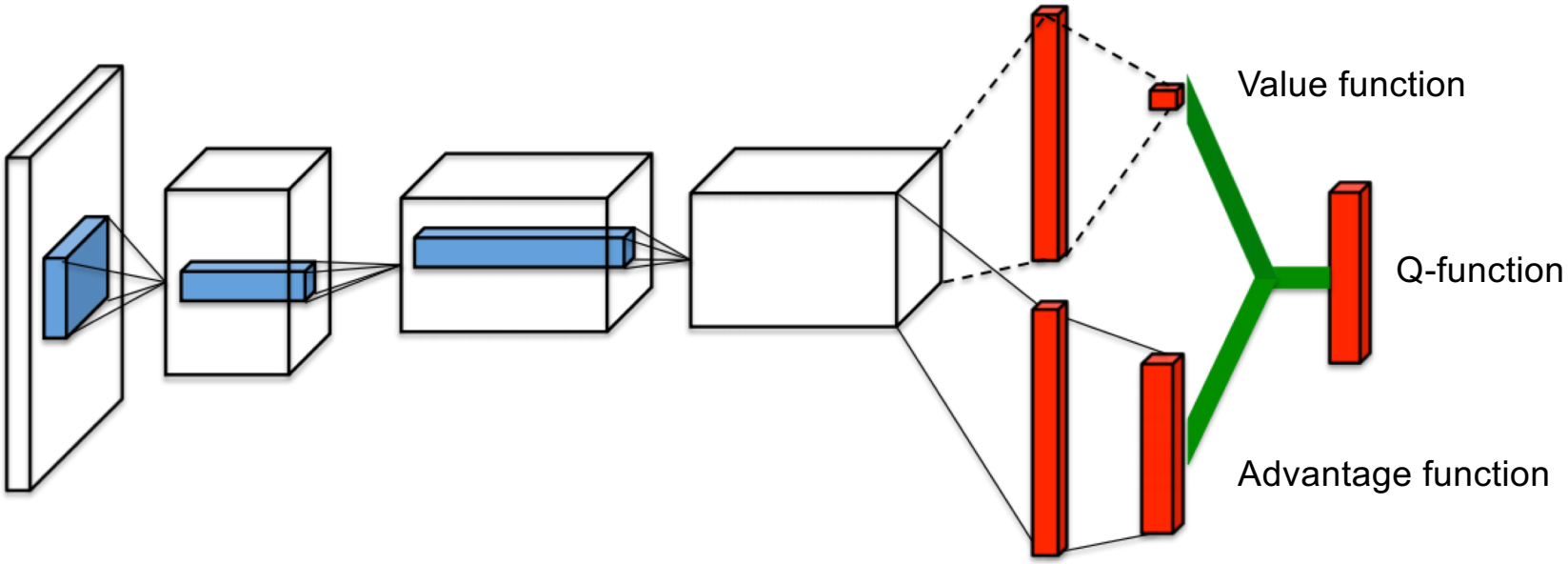
H. van Hasselt, A. Guez, D. Silver, Deep Reinforcement Learning with Double Q-learning, AAAI 2016

Double DQN results



Another extension: Dueling DQN

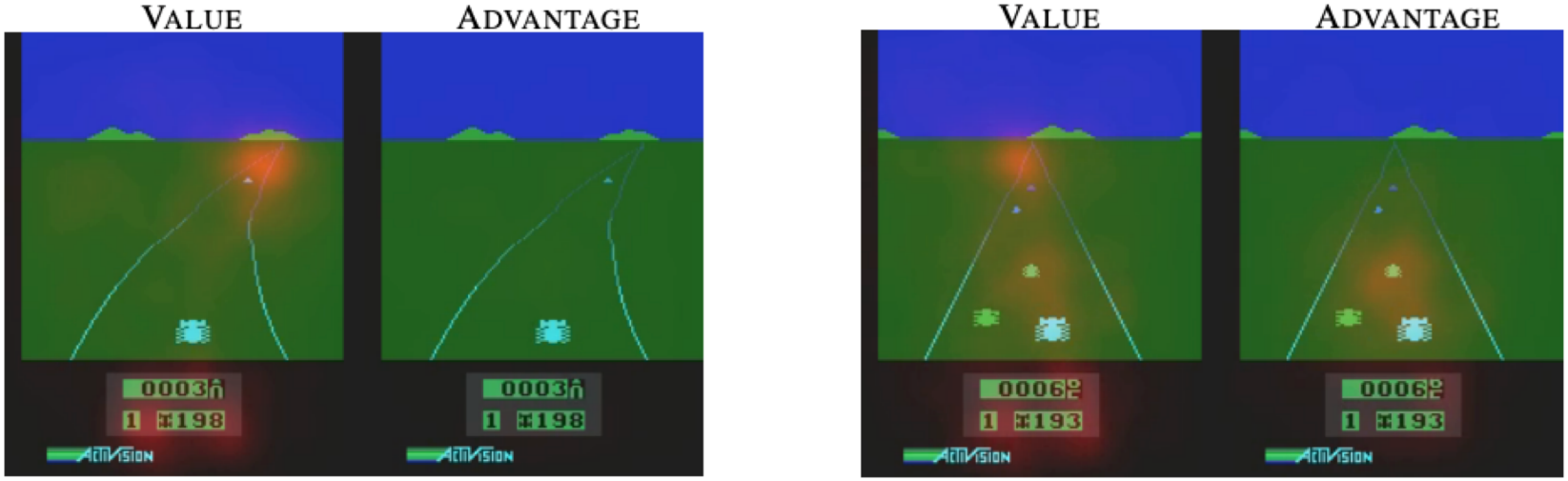
- Decompose estimation of Q-function into *value* and *advantage* functions



Z. Wang et al., Dueling Network Architectures for Deep Reinforcement Learning, ICML 2016

Dueling DQN

- Decompose estimation of Q-function into *value* and *advantage* functions
  - Motivation: in many states, actions don't meaningfully affect the environment, so it is not necessary to know the exact value of each action at each time step



Z. Wang et al., Dueling Network Architectures for Deep Reinforcement Learning, ICML 2016

Dueling DQN

- Decompose estimation of Q-function into *value* and *advantage* functions:

$$Q(s,a) = V(s) + (A(s,a) - \max_{a'} A(s,a'))$$ or

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

Z. Wang et al., Dueling Network Architectures for Deep Reinforcement Learning, ICML 2016

Dueling DQN: Results

Improvements over prioritized DDQN baseline: