Deep Reinforcement Learning

Deep Q-Networks

Carlos Morato

Used Materials

•**Disclaimer**: Much of the material and slides for this lecture were borrowed from Rich Sutton’s RL class and David Silver’s Deep RL tutorial

Components of an RL Agent

- An RL agent may include one or more of these components:
  - Policy: agent's behavior function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

- A policy is the agent's behavior
- It is a map from state to action:
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = P[a|s]$

Review: Value Function

- A value function is a prediction of future reward
  - How much reward will I get from action a in state s?
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

- Value functions decompose into a Bellman equation

$$Q^{\pi}(s, a) = \mathbb{E}_{s', a'}\left[r + \gamma Q^{\pi}(s', a') \mid s, a\right]$$

Optimal Value Function

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

- Once we have Q*, the agent can act optimally

$$\pi^*(s) = \operatorname*{argmax}_{a} Q^*(s,a)$$

- Formally, optimal values decompose into a Bellman equation

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

Optimal Value Function

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

- Formally, optimal values decompose into a Bellman equation

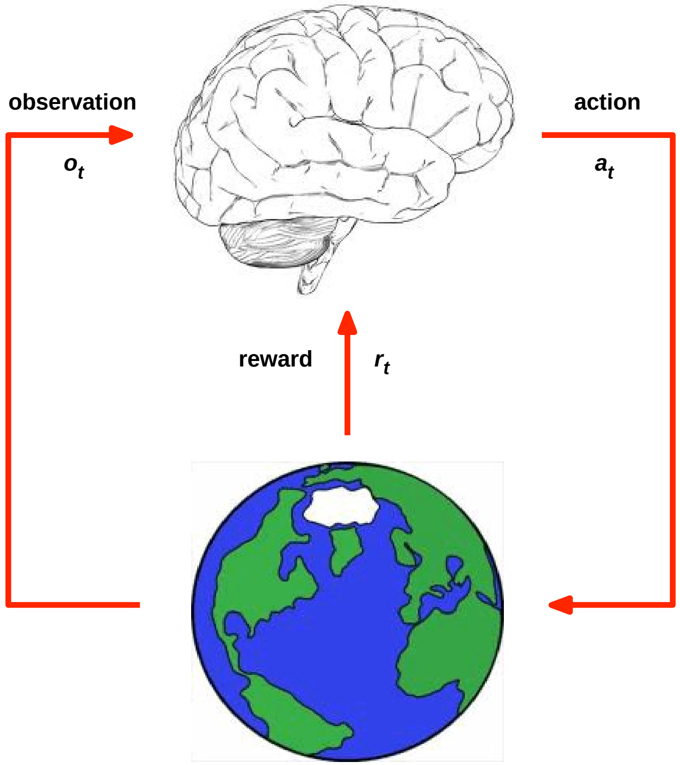
$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$

- Informally, optimal value maximizes over all decisions

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

Model

- Model is learned from experience
- Acts as proxy for environment
- Planner interacts with model, e.g. using look-ahead search



Approaches to RL

- Value-based RL (this is what we have looked at so far)
  - Estimate the optimal value function $Q^*(s,a)$
  - This is the maximum value achievable under any policy
- Policy-based RL
  - Search directly for the optimal policy π^*
  - This is the policy achieving maximum future reward
- Model-based RL
  - Build a model of the environment
  - Plan (e.g. by look-ahead) using model
- Let us revisit value-based RL.

Deep Reinforcement Learning

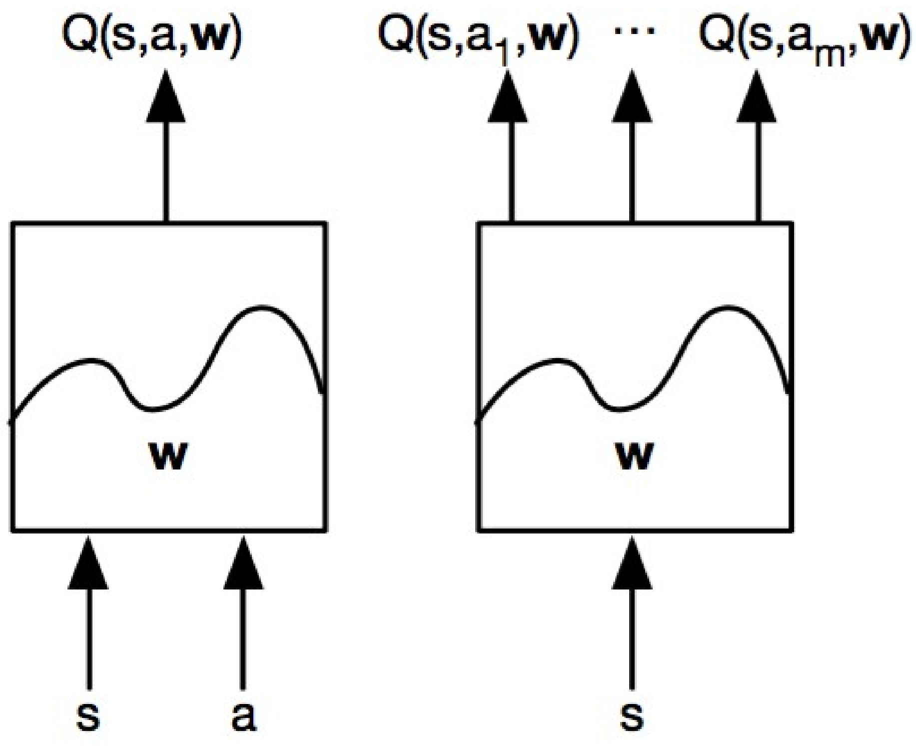
- Use deep neural networks to represent
  - Value function
  - Policy
  - Model

- Optimize loss function by stochastic gradient descent (SGD)

Deep Q-Networks (DQNs)

- Represent value function by Q-network with weights w

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



Q-Learning

- Optimal Q-values should obey Bellman equation

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

- Treat right-hand $r + \gamma \max_{a'} Q(s', a', \mathbf{w})$ as a target

- Minimize MSE loss by stochastic gradient descent

$$I = \left(r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})\right)^2$$

- Remember VFA lecture: Minimize mean-squared error between the true action-value function $q_\pi(S,A)$ and the approximate Q function:

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

Q-Learning

- Minimize MSE loss by stochastic gradient descent

$$l = \left(r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \right)^2$$

- Converges to Q* using table lookup representation

- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

DQNs: Experience Replay

- To remove correlations, build data-set from agent's own experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$$\rightarrow \quad s, a, r, s'$$

- Sample experiences from data-set and apply update

$$I = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- To deal with non-stationarity, target parameters w− are held fixed

Remember: Experience Replay

- Given experience consisting of ⟨state, value⟩, or <state, action/value> pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Repeat
  - Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

  - Apply stochastic gradient descent update

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

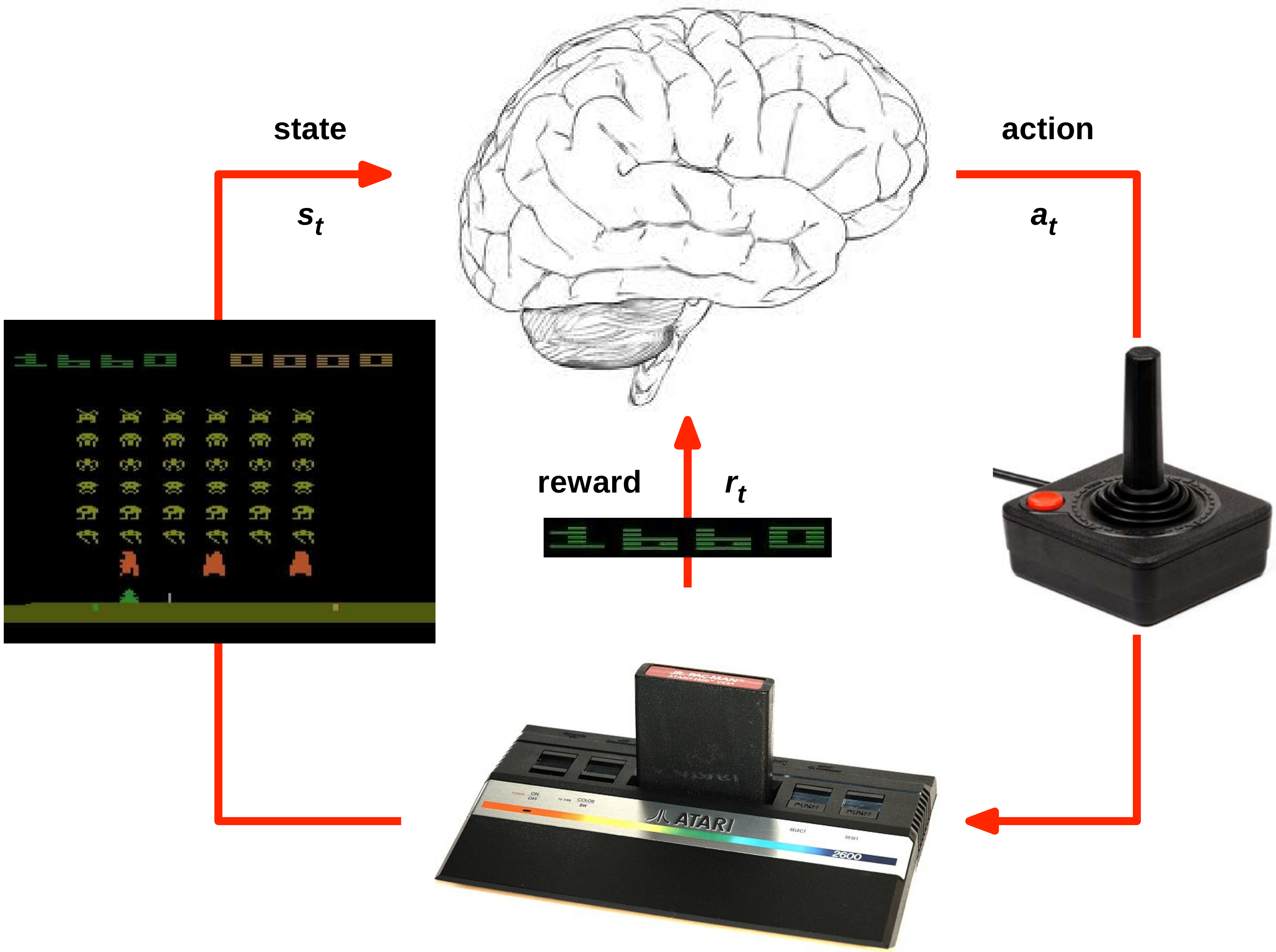
DQNs: Experience Replay

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- Sample random mini-batch of transitions (s,a,r,s′) from D
- Compute Q-learning targets w.r.t. old, fixed parameters w−
- Optimize MSE between Q-network and Q-learning targets

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

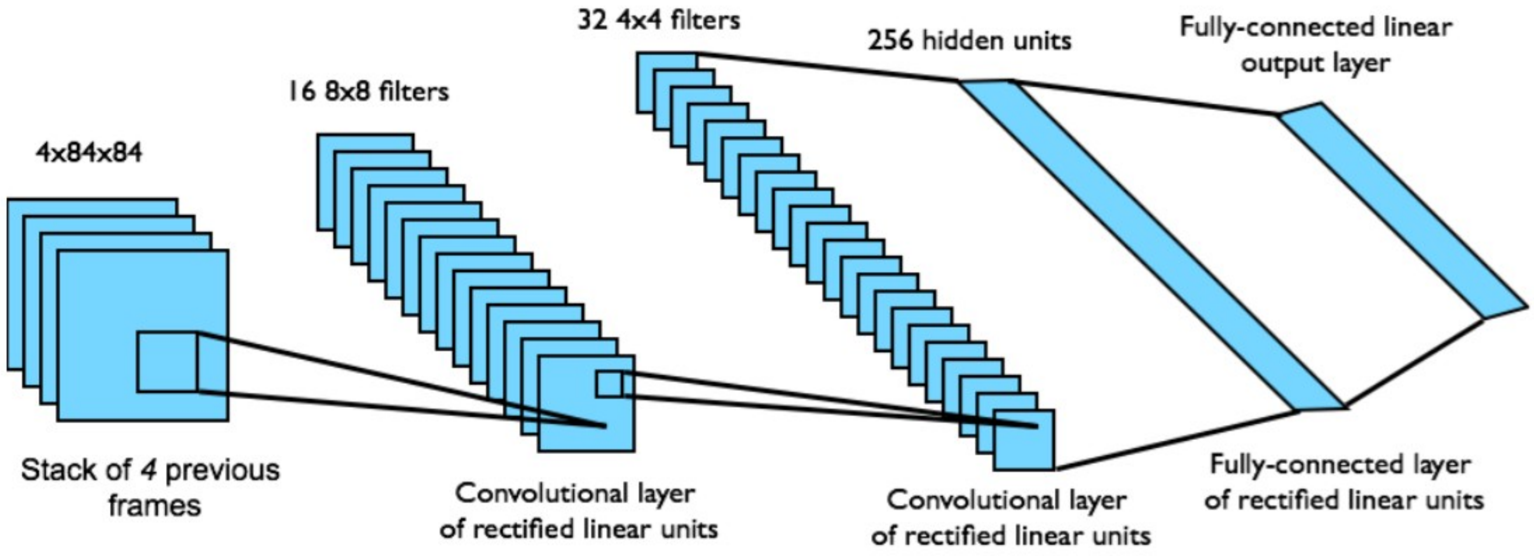
- Use stochastic gradient descent

DQNs in Atari



DQNs in 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

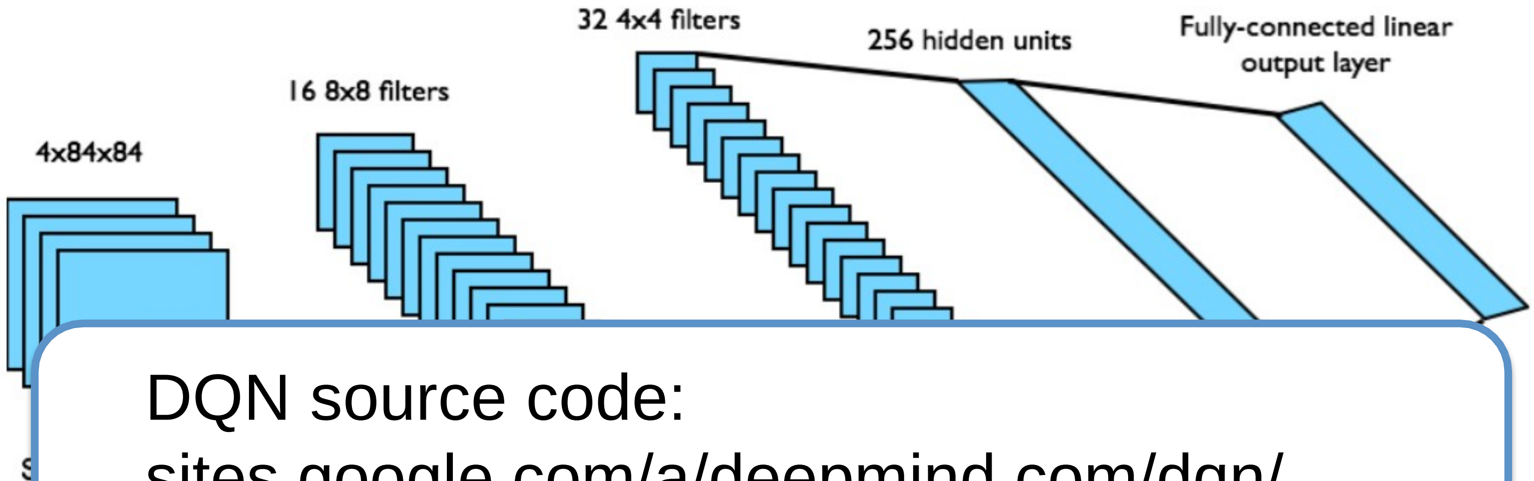


- Network architecture and hyperparameters fixed across all games

Mnih et.al., Nature, 2014

DQNs in 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 source code:
sites.google.com/a/deepmind.com/dqn/

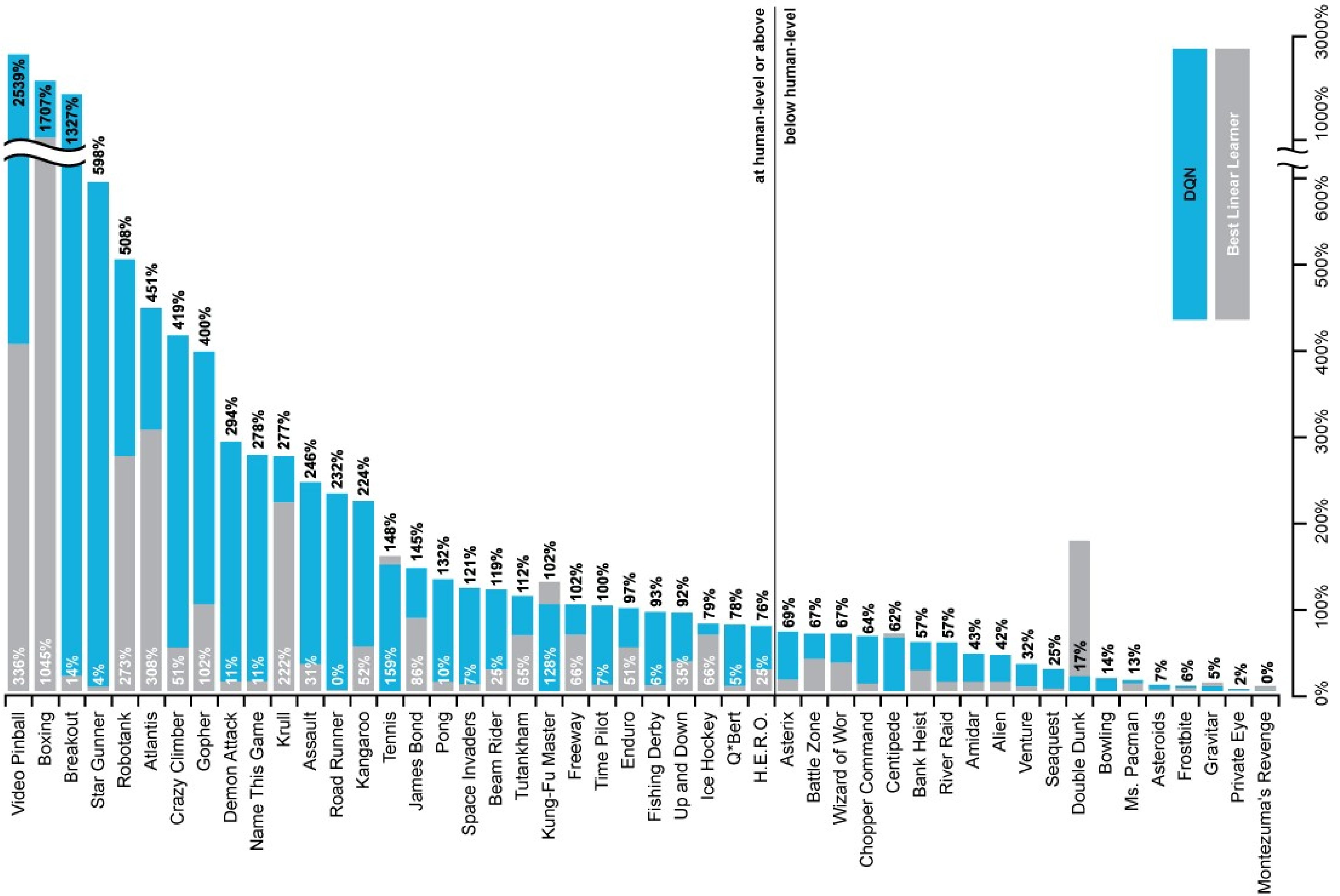
- Network architecture and hyperparameters fixed across all games

Mnih et.al., Nature, 2014

Demo

Mnih et.al., Nature, 2014

DQN Results in Atari



Double Q-Learning

- Train 2 action-value functions, Q_1 and Q_2
- Do Q-learning on both, but
  - never on the same time steps (Q_1 and Q_2 are independent)
  - pick Q_1 or Q_2 at random to be updated on each step

- If updating Q_1, use Q_2 for the value of the next state:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) +$$
$$+ \alpha \Big(R_{t+1} + Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big)$$

- Action selections are ε-greedy with respect to the sum of Q_1 and Q_2

Double Q-Learning

Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
    Take action A, observe R, S'
    With 0.5 probabilility:
$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$
    else:
$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$
    $S \leftarrow S'$;
  until S is terminal

Hado van Hasselt 2010

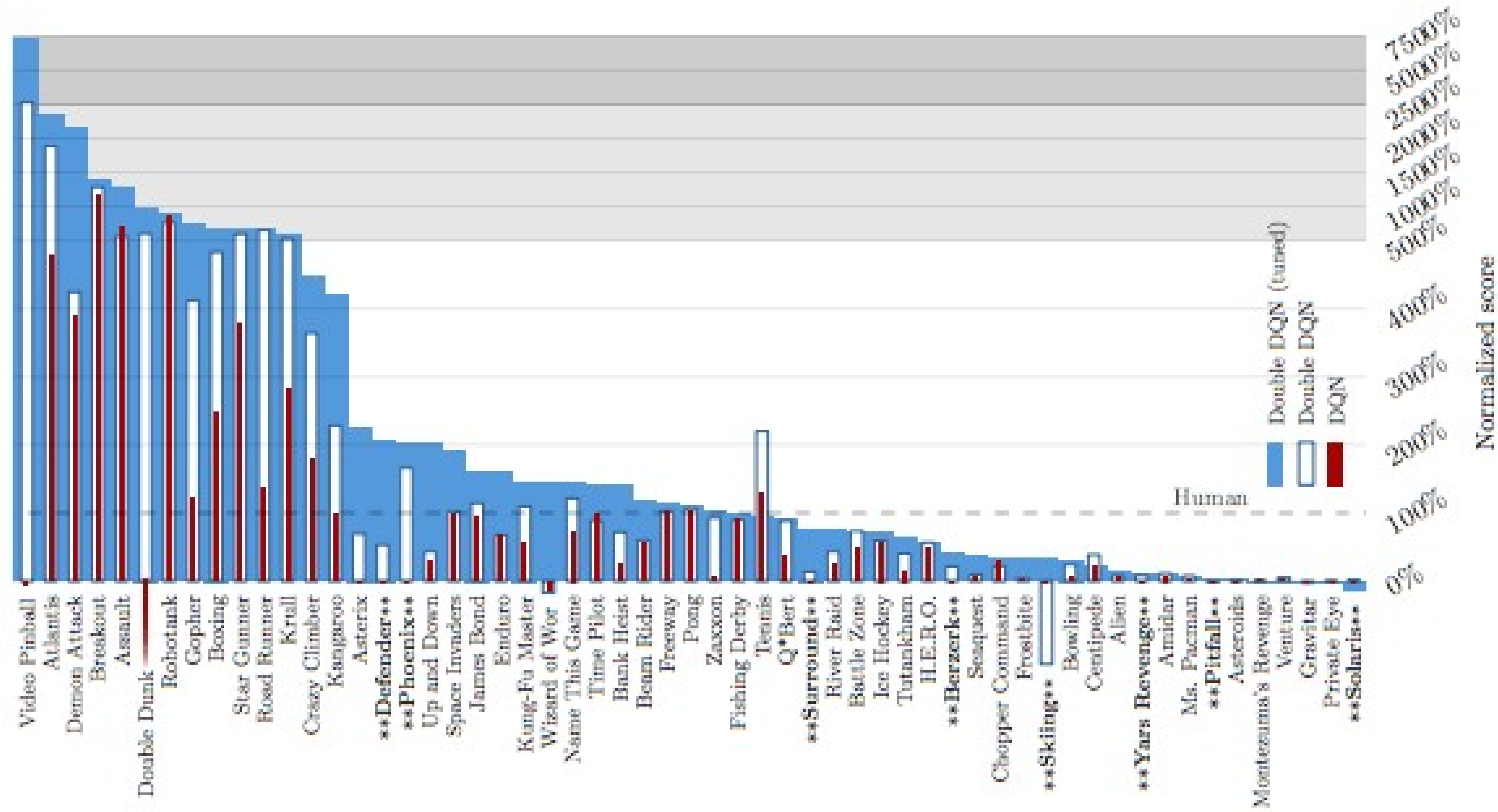
Double DQN

- Current Q-network w is used to select actions
- Older Q-network w− is used to evaluate actions

$$l = \Big(r + \gamma \overbrace{Q(s', \underbrace{\operatorname*{argmax}_{a'} Q(s', a', \mathbf{w})}_{\text{Action selection: w}}, \mathbf{w}^-)}^{\text{Action evaluation: w}-} - Q(s, a, \mathbf{w})\Big)^2$$

van Hasselt, Guez, Silver, 2015

Double DQN



van Hasselt, Guez, Silver, 2015

Prioritized Replay

- Weight experience according to surprise
- Store experience in priority queue according to DQN error

$$\underbrace{\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|}$$

- Stochastic Prioritization

p_i is proportional to DQN error

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

- α determines how much prioritization is used, with α = 0 corresponding to the uniform case.

Schaul, Quan, Antonoglou, Silver, ICLR 2016

Dueling Networks

- Split Q-network into two channels
- Action-independent value function V(s,v)
- Action-dependent advantage function A(s, a, w)

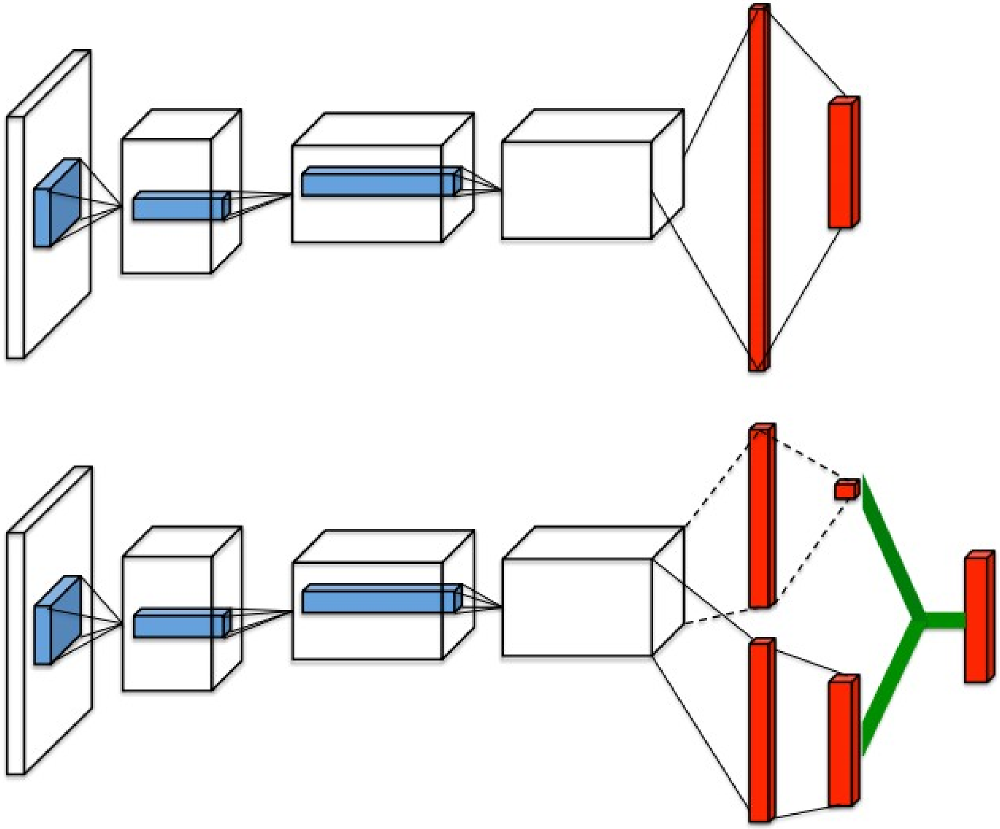
$$Q(s,a) = V(s,v) + A(s,a,\mathbf{w})$$

- Advantage function is defined as:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s).$$

Wang et.al., ICML, 2016

Dueling Networks vs. DQNs



DQN

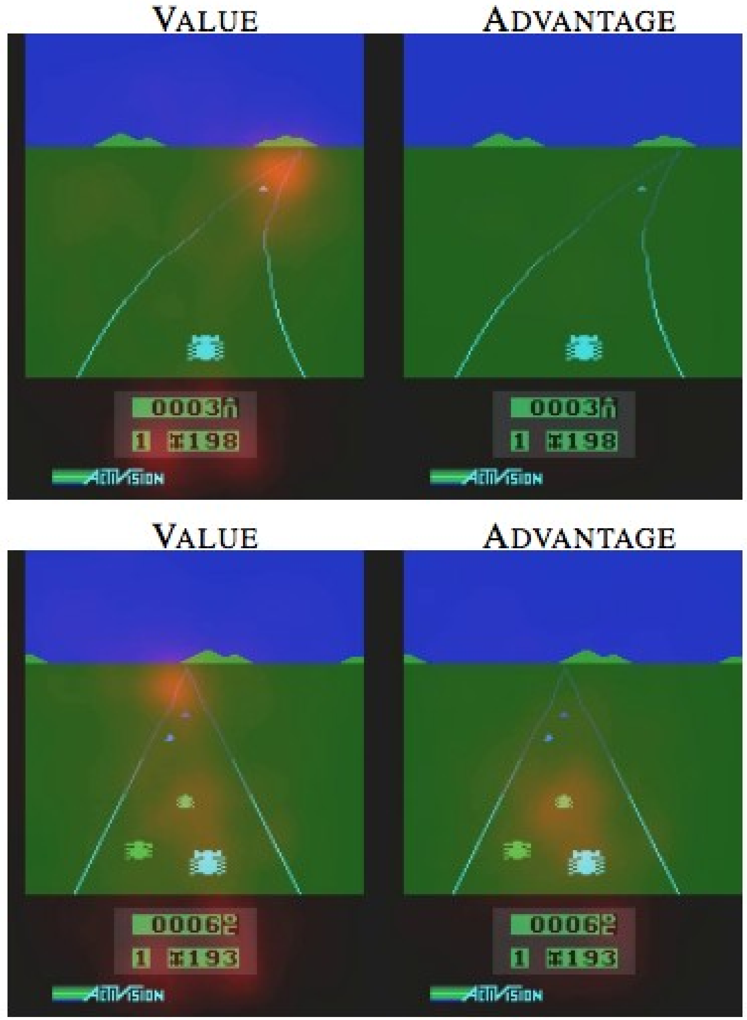
Dueling Networks

$$Q(s, a) = V(s, v) + A(s, a, \mathbf{w})$$

Wang et.al., ICML, 2016

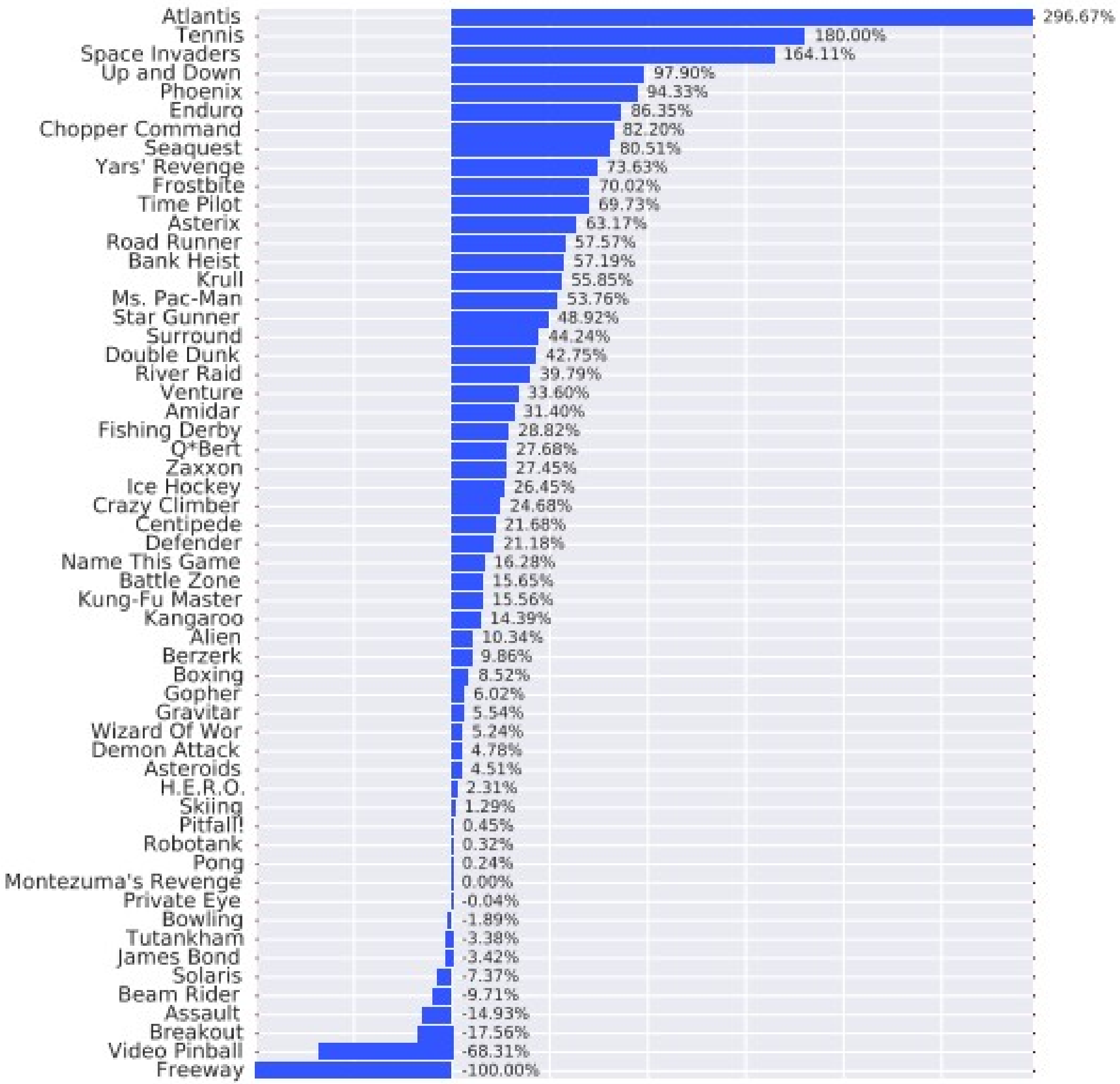
Dueling Networks

- The value stream learns to pay attention to the road
- The advantage stream: pay attention only when there are cars immediately in front, so as to avoid collisions



Wang et.al., ICML, 2016

Dueling Networks



Wang et.al., ICML, 2016

Mulitask DQNs

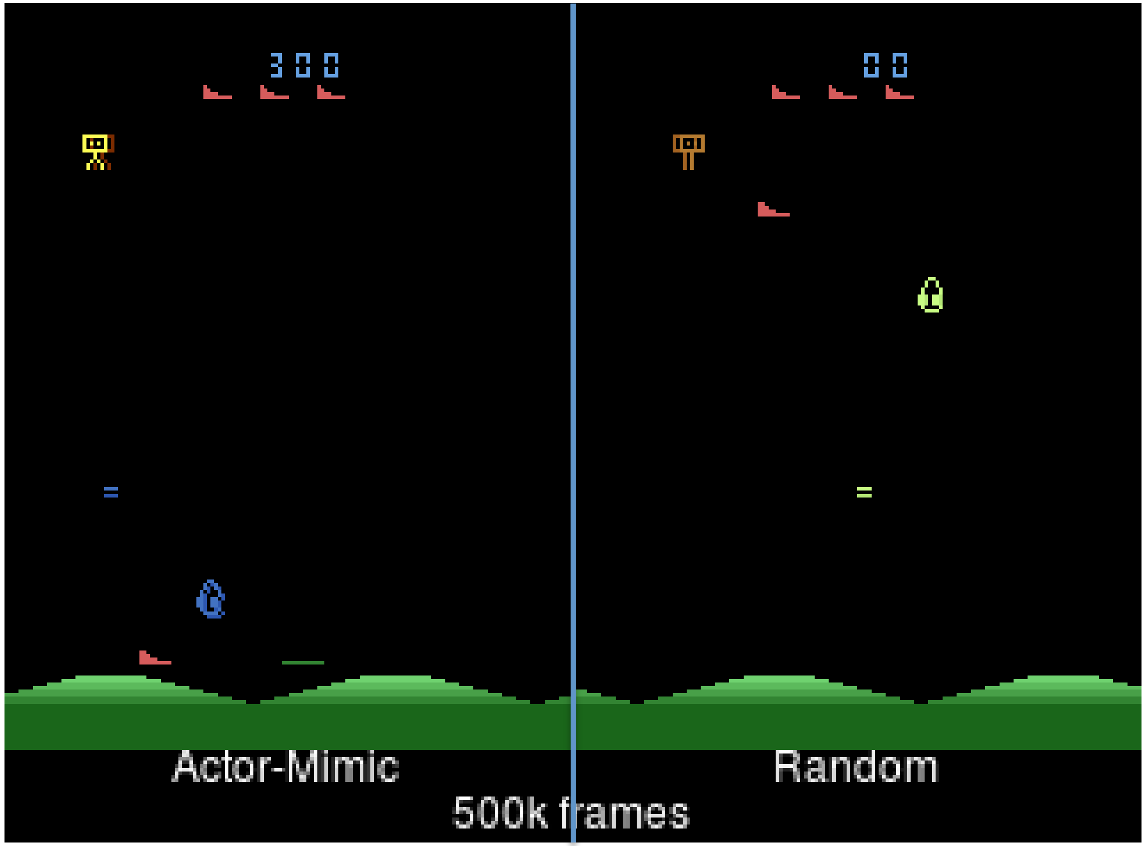
•Can we train a single DQN to play multiple games at the same time

(Parisotto, Ba, Salakhutdinov, ICLR 2016)

Transfer Learning

•Can the network learn new games faster by leveraging knowledge about the previous games it learned.

Transfer | No Transfer



Star Gunner