Reinforcement Learning: Basics and DQN

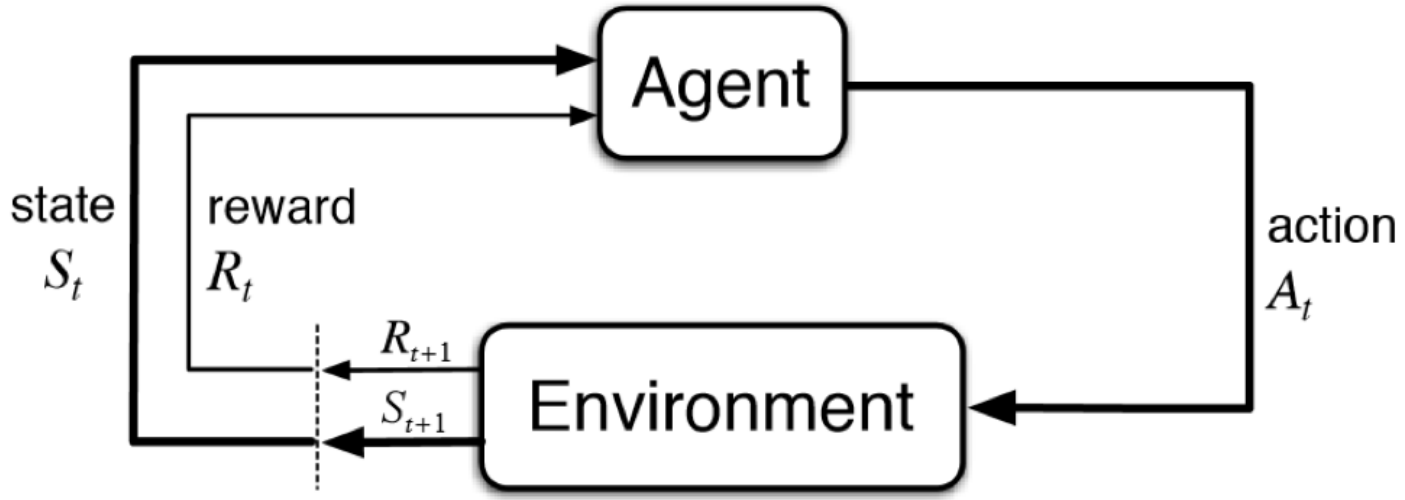
Vishnu Lokhande

Department of Computer Science and Engineering
University at Buffalo, SUNY
vishnulo@buffalo.edu

April 21, 2025

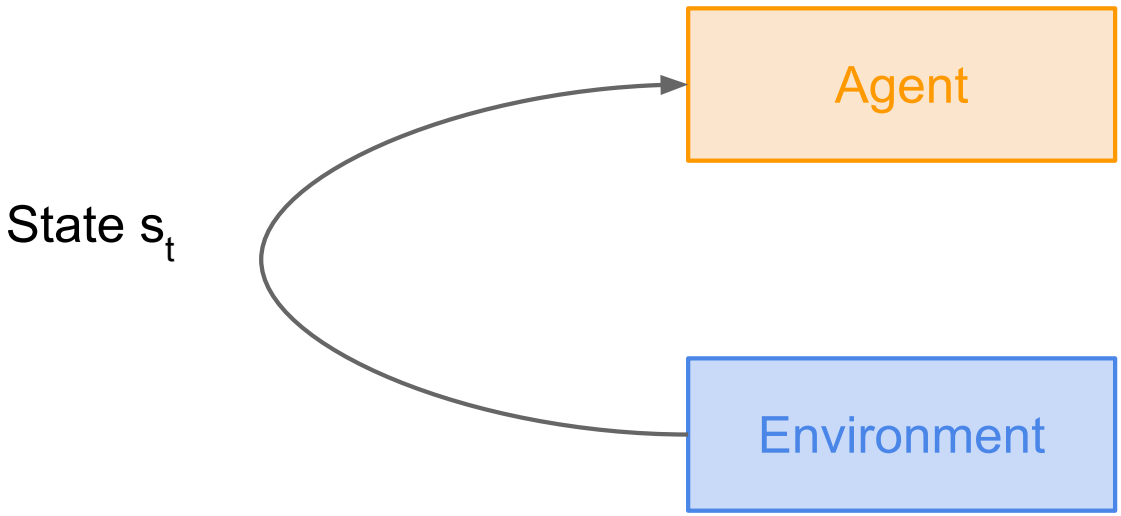
Reinforcement Learning

- Problems involving an agent interacting with an environment, which provides numeric reward signals.
- **Goal**: Learn how to take actions in order to maximize reward.

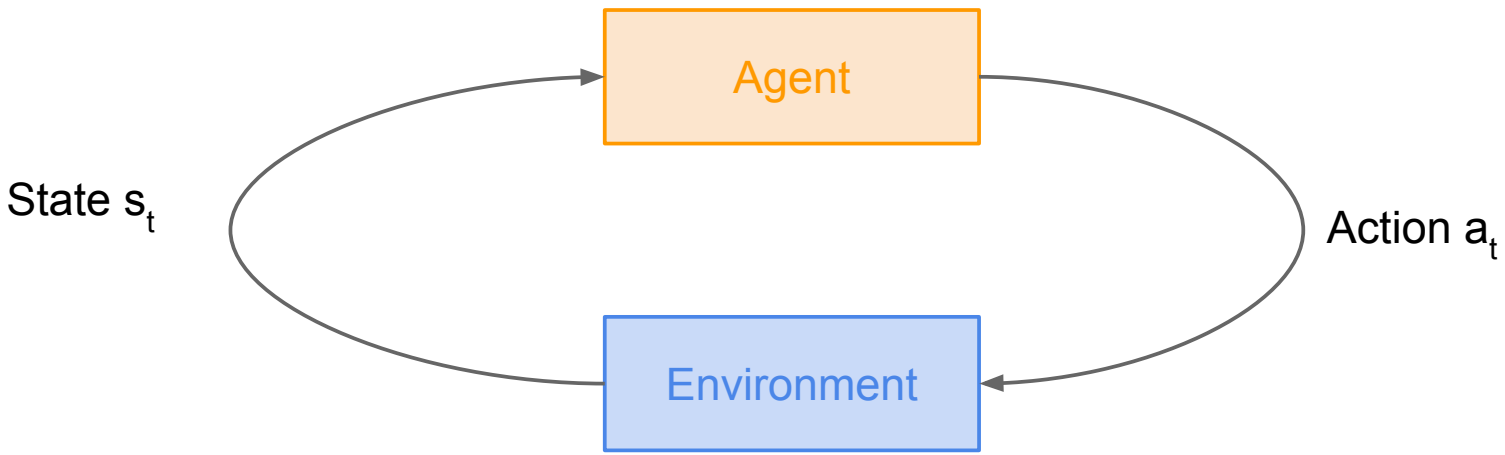


Reinforcement Learning

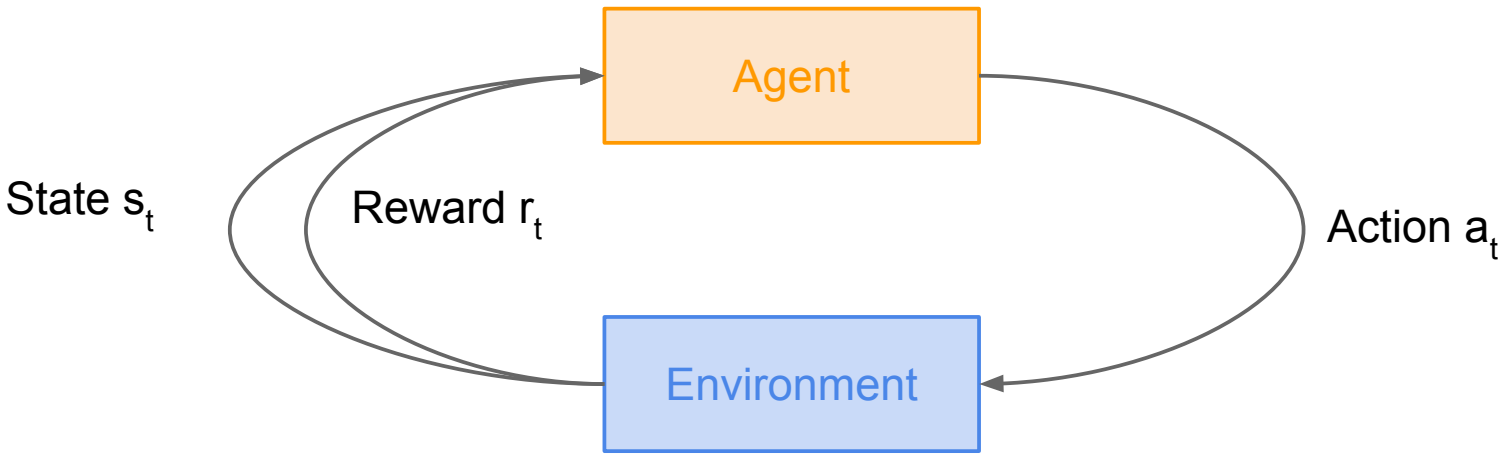
Reinforcement Learning



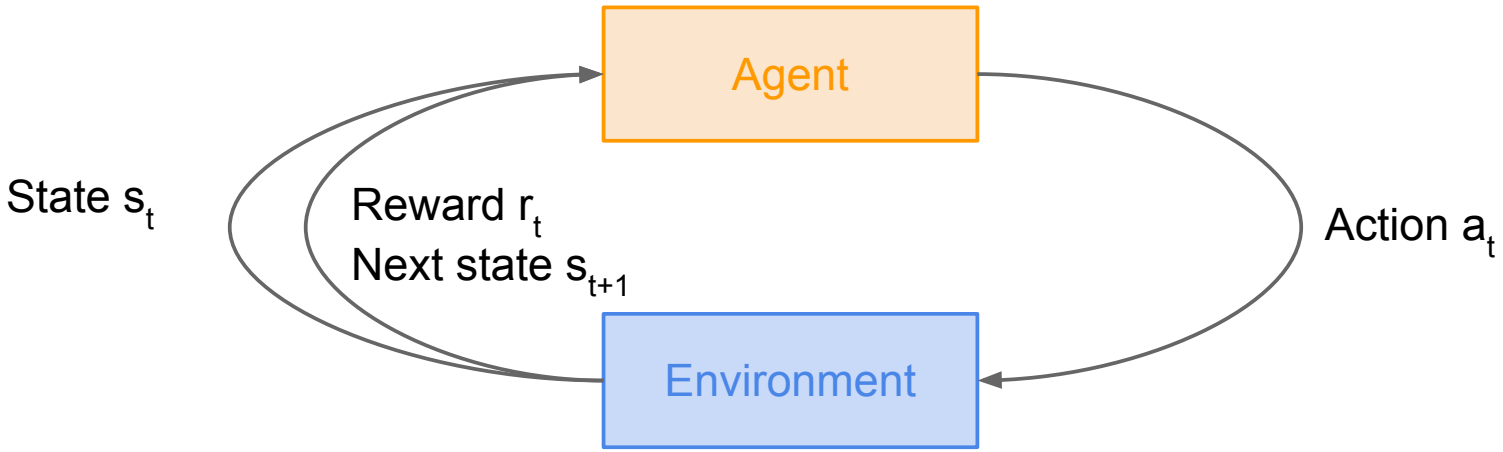
Reinforcement Learning



Reinforcement Learning

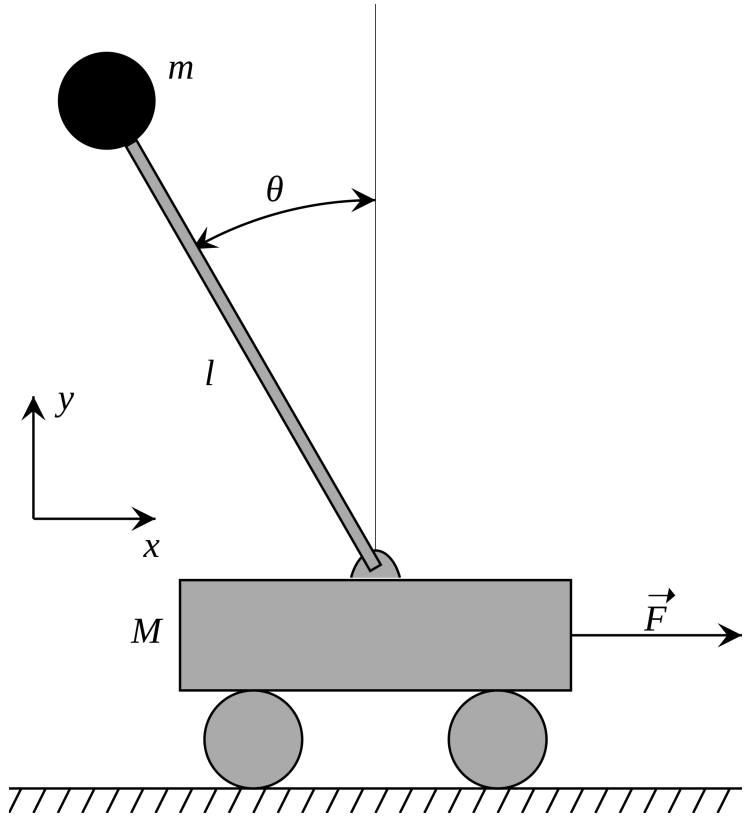


Reinforcement Learning



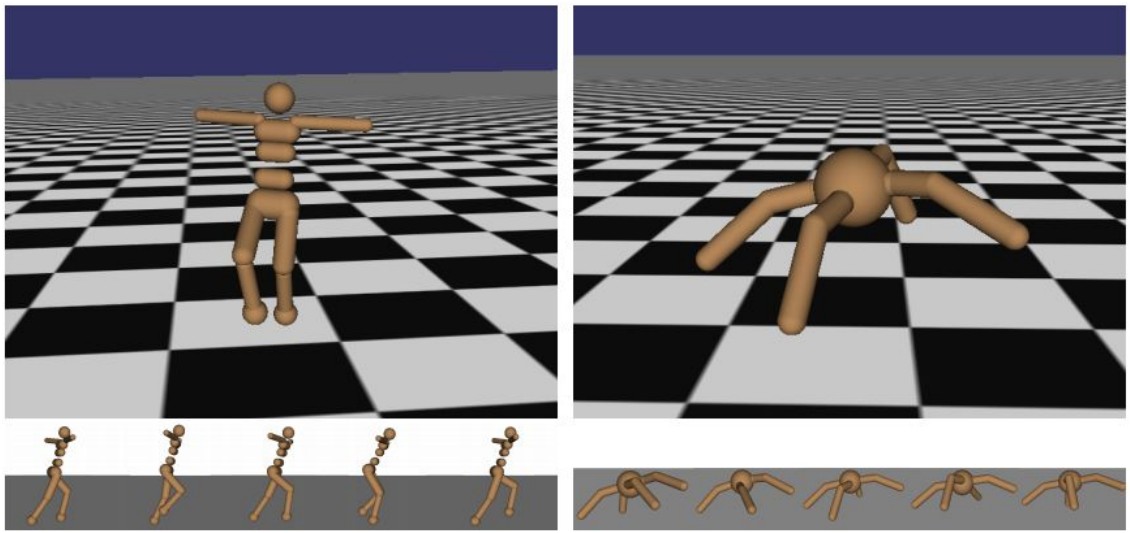
Cart-Pole Problem

- **Objective**: Balance a pole on top of a movable cart.
- **State**: angle, angular speed, position, horizontal velocity.
- **Action**: horizontal force applied on the cart.
- **Reward**: 1 at each time step if the pole is upright.



Robot Locomotion

- **Objective**: Make the robot move forward.
- **State**: Angle and position of the joints.
- **Action**: Torques applied on joints.
- **Reward**: 1 at each time step upright + forward movement.

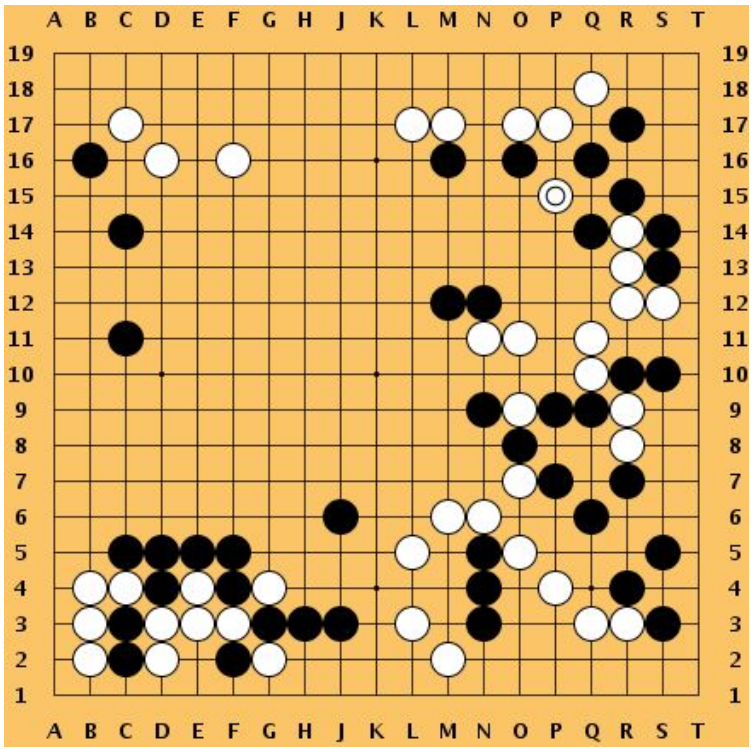


Atari Games

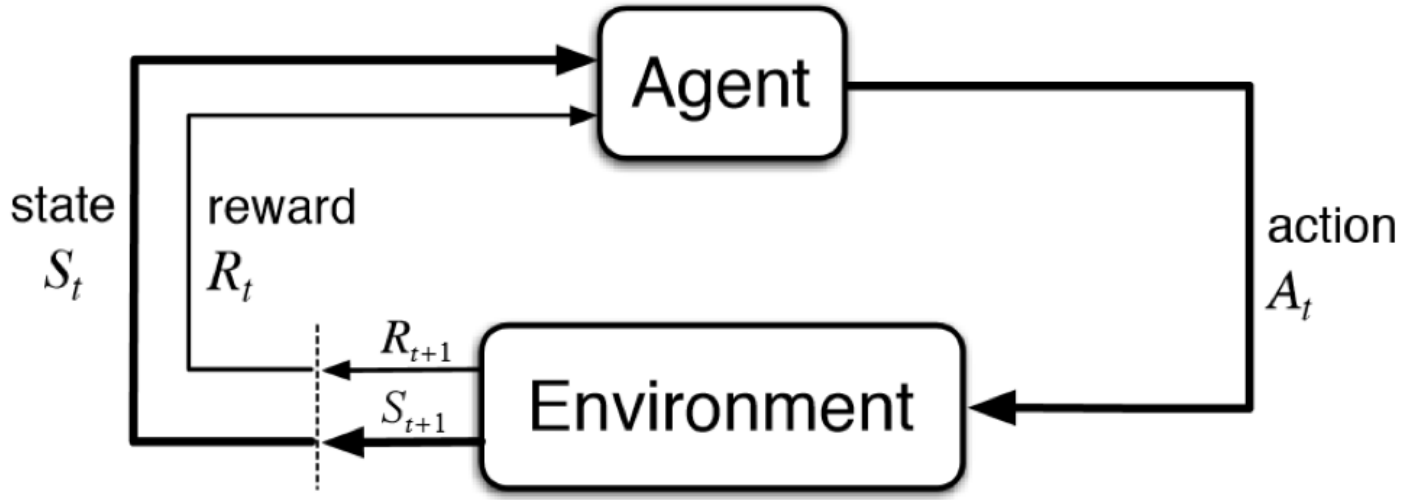
- **Objective**: Complete the game with the highest score.
- **State**: Raw pixel inputs of the game state.
- **Action**: Game controls e.g. Left, Right, Up, Down.
- **Reward**: Score increase/decrease at each time step.

Go

- **Objective**: Win the game.
- **State**: Position of all pieces.
- **Action**: Where to put the next piece down.
- **Reward**: 1 if win at the end of the game, 0 otherwise.



How to Mathematically Formalize the RL Problem?



Markov Decision Process

- Mathematical formulation of the RL problem.
- **Markov property**: Current state completely characterizes the state of the world.

$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

- $\mathcal{S}$: set of possible states.
- $\mathcal{A}$: set of possible actions.
- $\mathcal{R}$: distribution of reward given (state, action) pairs.
- $\mathbb{P}$: transition probability, *i.e.*, distribution over next state given a (state, action) pair.
- γ: discount factor.

Markov Decision Process

1. At time step $t = 0$, environment samples initial state $s_0 \sim p(s_0)$.
2. Then, for $t = 0$ until done:
   - Agent selects an action a_t.
   - Environment samples reward $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$.
   - Environment samples next state $s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)$.
   - Agent receives reward r_t and next state s_{t+1}.
3. A policy π is a function from $\mathcal{S}$ to $\mathcal{A}$ that specifies what action to take in each state:
   - usually it is modeled as a conditional distribution of action given state.
4. **Objective**: find policy π^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$.

Markov Decision Process

1. At time step $t = 0$, environment samples initial state $s_0 \sim p(s_0)$.
2. Then, for $t = 0$ until done:
   - Agent selects an action a_t.
   - Environment samples reward $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$.
   - Environment samples next state $s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)$.
   - Agent receives reward r_t and next state s_{t+1}.
3. A policy π is a function from $\mathcal{S}$ to $\mathcal{A}$ that specifies what action to take in each state:
   - usually it is modeled as a conditional distribution of action given state.
4. **Objective**: find policy π^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$.

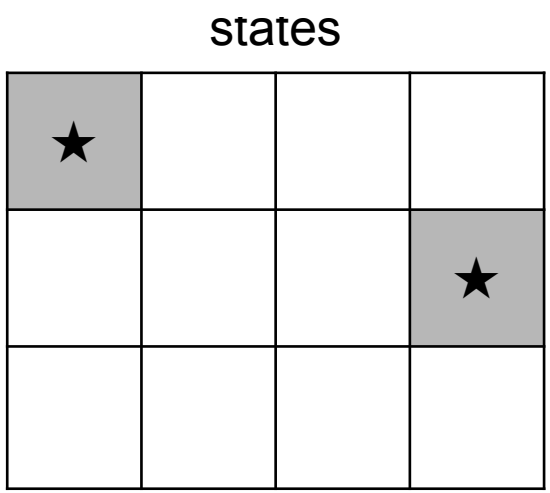
We need to learn the policy π^* and sometimes (parts of) the MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$.

A Simple MDP: Policy for Grid World

actions = {

1. right
2. left
3. up
4. down

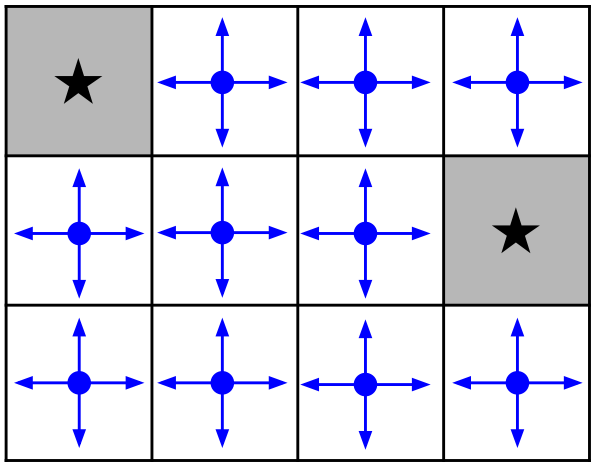
}



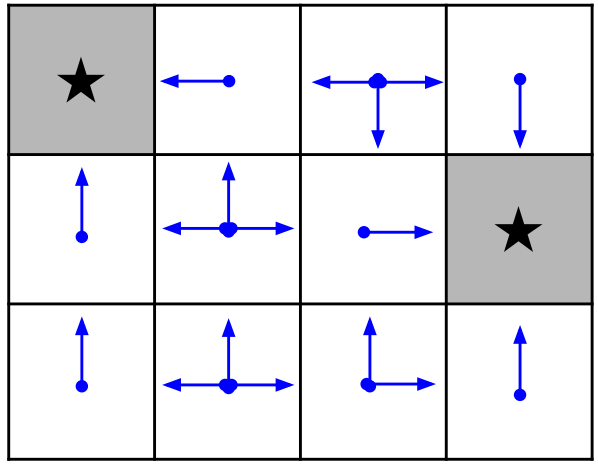
Set a negative "reward" for each transition (e.g. r = -1)

Objective: reach one of terminal states (greyed out) in least number of actions

A Simple MDP: Policy for Grid World



Random Policy



Optimal Policy

The Optimal Policy

1. We want to find optimal policy π^* that maximizes the sum of rewards $\sum_{t \geq 0} \gamma^t r_t$.
2. Directly summing over rewards endows randomness, *e.g.*, initial state, transition probability, reward probability.
3. We should maximize the *expected total reward*:

$$\pi^* = \arg\max_{\pi} \sum_{t=1}^{\infty} \mathbb{E}_{s_t \sim \mathbb{P}(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi(\cdot|s_t)} \left[\gamma^t r_t(s_t, a_t) \right] ,$$

$$\text{with } s_0 \sim p(s_0)$$

The Optimal Policy

1. We want to find optimal policy π^* that maximizes the sum of rewards $\sum_{t \geq 0} \gamma^t r_t$.
2. Directly summing over rewards endows randomness, *e.g.*, initial state, transition probability, reward probability.
3. We should maximize the *expected total reward*:

$$\pi^* = \arg\max_{\pi} \sum_{t=1}^{\infty} \mathbb{E}_{s_t \sim \mathbb{P}(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi(\cdot|s_t)} \left[\gamma^t r_t(s_t, a_t) \right] ,$$

$$\text{with } s_0 \sim p(s_0)$$

How to optimize this objective with an infinite sum, expectation with unknown distributions?

Exact Methods and Monte Carlo Approximation[1]

[1] Adapted from: https://drive.google.com/file/d/0BxXI_RttTZAhVXBlMUVkQ1BVVDQ/view, https://drive.google.com/file/d/0BxXI_RttTZAhREJKRGhDT25OOTA/view

Exact Methods

Optimal control

- Given an MDP: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$, find the optimal policy π^*.

Exact methods

- Value iteration.
- Policy iteration.

Exact Methods

Optimal control

- Given an MDP: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$, find the optimal policy π^*.

Exact methods

- Value iteration.
- Policy iteration.

Optimal Value Function

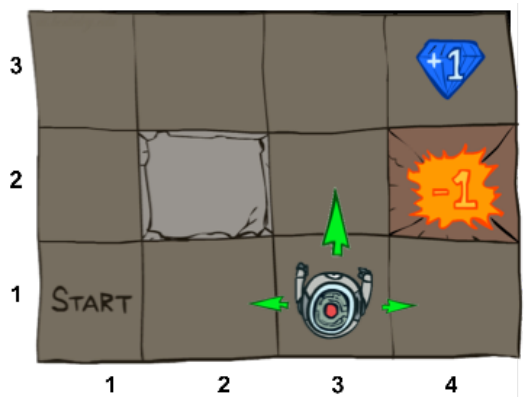
$$\pi^* = \arg\max_{\pi} \sum_{t=1}^{\infty} \mathbb{E}_{s_t \sim \mathbb{P}(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi(\cdot|s_t)} \left[\gamma^t R(s_t, a_t, s_{t+1})\right]$$

- Define the optimal value function $V^*(s)$ as the sum of discounted rewards when starting from state s and acting optimally:

$$V^*(s) \triangleq \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) | \pi, s_0 = s\right]$$

$$= \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) | \pi, s_0 = s, H \to \infty\right]$$

- What are the values $V^*(1,1)$, $V^*(1,2)$, $\cdots$, $V^*(3,4)$?



Value Iteration: Dynamic Programming to Deal with Infinite Horizon $H \to \infty$

- $V_0^*(s)$: optimal value for state s when $H = 0$:

$$V_0^*(s) = 0, \forall s$$

- $V_1^*(s)$: optimal value for state s when $H = 1$:

$$V_1^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_0^*(s'))$$

- $V_2^*(s)$: optimal value for state s when $H = 2$:

$$V_2^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_1^*(s'))$$

- $V_k^*(s)$: optimal value for state s when $H = k$:

$$V_k^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_{k-1}^*(s'))$$

Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For k = 1, … , H:

For all states s in S:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

$$\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

This is called a value update or Bellman update/back-up

Value Iteration

Theorem

Value iteration converges. At convergence, we have found the optimal value function V^* for the discounted infinite horizon problem, which satisfies the Bellman equations:

$$\forall s, V^*(s) = \max_a \sum_{s'} P(s'|s) \left[R(s, a, s') + \gamma V^*(s') \right]$$

Summary: Now we know how to act for infinite horizon with discounted rewards:

- Run value iteration until convergence.
- This produces V^*, which tells us the optimal policy:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s) \left[R(s, a, s') + \gamma V^*(s') \right]$$

Q-Values

- $Q^*(s, a)$: expected reward starting in s, taking action a, and (thereafter) acting optimally.
- Bellman equation (at convergence):

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

- Q-value iteration:

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right)$$

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right)$$

$$= \arg\max_a Q_{k+1}(s,a)$$

Exact Methods

Optimal control

- Given an MDP: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$, find the optimal policy π^*.

Exact methods

- Value iteration.
- Policy iteration.

Policy Evaluation (Optional)

- Recall value iteration:

$$V_k^*(s) = \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

- Policy evaluation for a given π:

Deterministic π: $V_i^\pi(s) = \sum_{s'} P(s'|s,\pi(s)) \left(R(s,\pi(s),s') + \gamma V_{i-1}^*(s')\right)$

Stochastic π: $V_i^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)} \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{i-1}^*(s')\right)$

- At convergence:

$$\forall s,\ V^\pi(s) = \sum_{s'} P(s'|s,\pi(s)) \left(R(s,\pi(s),s') + \gamma V^*(s')\right)$$

Policy Iteration (Optional)

One iteration of policy iteration:

- Policy evaluation for current policy π_k :
  - Iterate until convergence

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s, \pi_k(s)) \left[R(s, \pi(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

- Policy improvement: find the best action according to one-step look-ahead

$$\pi_{k+1}(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

Policy Iteration (Optional)

Theorem

Policy iteration is guaranteed to converge. At convergence, the current policy and its value function are the optimal policy and the optimal value function.

Limitations:

- Iteration over / storage for all states and actions: requires small, discrete state-action space:
  - sampling-based approximations
- Update equations require access to dynamics mode.

Policy Iteration (Optional)

Theorem

Policy iteration is guaranteed to converge. At convergence, the current policy and its value function are the optimal policy and the optimal value function.

Limitations:

- Iteration over / storage for all states and actions: requires small, discrete state-action space:
  - sampling-based approximations
- Update equations require access to dynamics mode.

Sampling-Based Approximation

- Q-value iteration?
- Value iteration?
- Policy iteration:
  - policy evaluation?
  - policy improvement?

Sampling-Based Approximation

- Q-value iteration?
- Value iteration?
- Policy iteration:
  - policy evaluation?
  - policy improvement?

Recap: Q-Values

- $Q^*(s, a)$: expected reward starting in s, taking action a, and (thereafter) acting optimally.
- Bellman equation:

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

- Q-value iteration:

$$Q_{k+1}(s, a) = \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right)$$

(Tabular) Q-Learning

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right)$$

$$\Rightarrow Q_{k+1}(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

(Tabular) Q-Learning: replace expectation by samples:

- For an state-action pair (s,a), sample $s' \sim P(s'|s,a)$.
- Calculate the new sample estimate based on the old estimate $Q_k(s,a)$:

$$\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

- Update the new sample estimate by a running average:

$$Q_{k+1}(s,a) = (1-\alpha)Q_k(s,a) + \alpha \times \text{target}(s')$$

(Tabular) Q-Learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \left[\text{target}\right]$$

$$s \leftarrow s'$$

How to Sample Actions?

- Random actions.
- Action that maximizes $Q_k(s, a)$ (greedy).
- ϵ-Greedy: choose random actions with prob.ϵ; otherwise choose actions greedily.
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough, but not decrease it too quickly

Sampling-Based Approximation

- Q-value iteration?
- Value iteration?
- Policy iteration:
  - policy evaluation?
  - policy improvement?

Value Iteration

$$V^*_{k+1}(s) = \max_a \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma V^*_k(s') \right]$$

- V^* does not depend on actions, have to integrate it out.
- Unclear how to draw samples through max.

Sampling-Based Approximation

- Q-value iteration?
- Value iteration?
- Policy iteration:
  - policy evaluation?
  - policy improvement?

Policy Iteration

One iteration of policy iteration:

- Policy evaluation for current policy π_k :
  - Iterate until convergence

$$V_{i+1}^{\pi_k}(s) \leftarrow \mathbb{E}_{s' \sim P(s'|s,\pi_k(s))}[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')]$$

Can be approximated by samples
This is called Temporal Difference (TD) Learning

- Policy improvement: find the best action according to one-step look-ahead

$$\pi_{k+1}(s) \leftarrow \arg\max_a \mathbb{E}_{s' \sim P(s'|s,a)}[R(s, a, s') + \gamma V^{\pi_k}(s')]$$

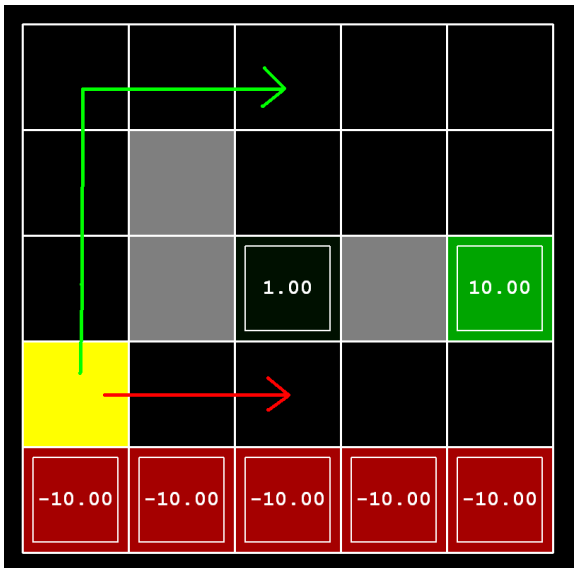
Unclear what to do with the max (for now)

Limitation

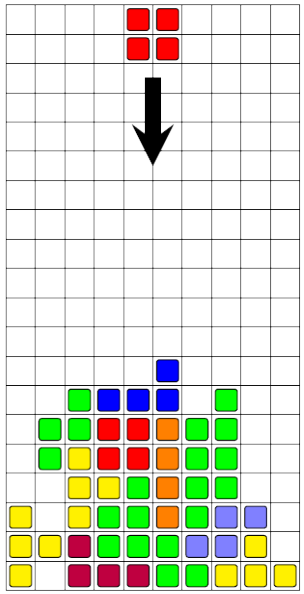
Limitations:

- Iteration over / storage for all states and actions: requires small, discrete state-action space:
  - sampling-based approximations
- Update equations require access to dynamics mode:
  - Q/V function fitting.

Can tabular methods scale?



Gridworld
10^1



Tetris
10^60

Atari
10^308 (ram) 10^16992 (pixels)

Approximate Q-Learning

- Instead of a table, we have a parametrized Q-function $Q_\theta(s, a)$:
  - Can be a linear function in features:

$$Q_\theta(s, a) = \theta_0 f_0(s, a) + \theta_1 f_1(s, a) + \cdots + \theta_n f_n(s, a)$$

  - Or a complicated neural network.
- Learning rule:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} \left(Q_\theta(s, a) - \text{target}(s') \right)^2 \right] |_{\theta=\theta_k}$$

$$\text{target}(s') \triangleq R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$$

$$Q_{k+1}(s, a) = \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right)$$

$$\approx R(s, a, s') + \gamma \max_{a'} Q_k(s', a'), \quad \text{with } s' \sim P(\cdot|s, a)$$

Connection to Tabular Q-Learning

- Suppose $\theta \in \mathbb{R}^{|S|\times|A|}, \quad Q_\theta(s,a) \equiv \theta_{sa}$

$$\nabla_{\theta_{sa}} \left[\frac{1}{2}(Q_\theta(s,a) - \text{target}(s'))^2\right]$$
$$= \nabla_{\theta_{sa}} \left[\frac{1}{2}(\theta_{sa} - \text{target}(s'))^2\right]$$
$$= \theta_{sa} - \text{target}(s')$$

- Plug into update: $\theta_{sa} \leftarrow \theta_{sa} - \alpha(\theta_{sa} - \text{target}(s'))$
$$= (1-\alpha)\theta_{sa} + \alpha[\text{target}(s')]$$

- Compare with Tabular Q-Learning update:

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\,[\text{target}(s')]$$

Convergence of Approximate Q-Learning

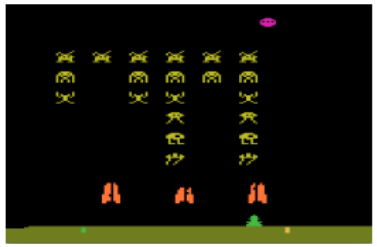
- It is not guaranteed to converge, even if the function approximation is expressive enough to represent the true Q-function:
  - An intuitive explanation: In the tabular setting, we have completely isolated entries $Q(s, a)$ for all (s, a) pairs. Whenever we update our estimate for one entry, it leaves all other entries unmodified. However, with function approximation, when we update our $Q(s, a)$ estimate for one (s, a) pair, it can potentially also affect all of our other estimates for all other state-action pairs. This might make the parameter jumping around in the parameter space without convergence.

Deep Q-Learning[2]

[2] Adapted from http://www.teach.cs.toronto.edu/∼csc2542h/fall/material/csc2542f16_dqn.pdf

Artari Games





Pong **Breakout** **Space Invaders** **Seaquest** **Beam Rider**

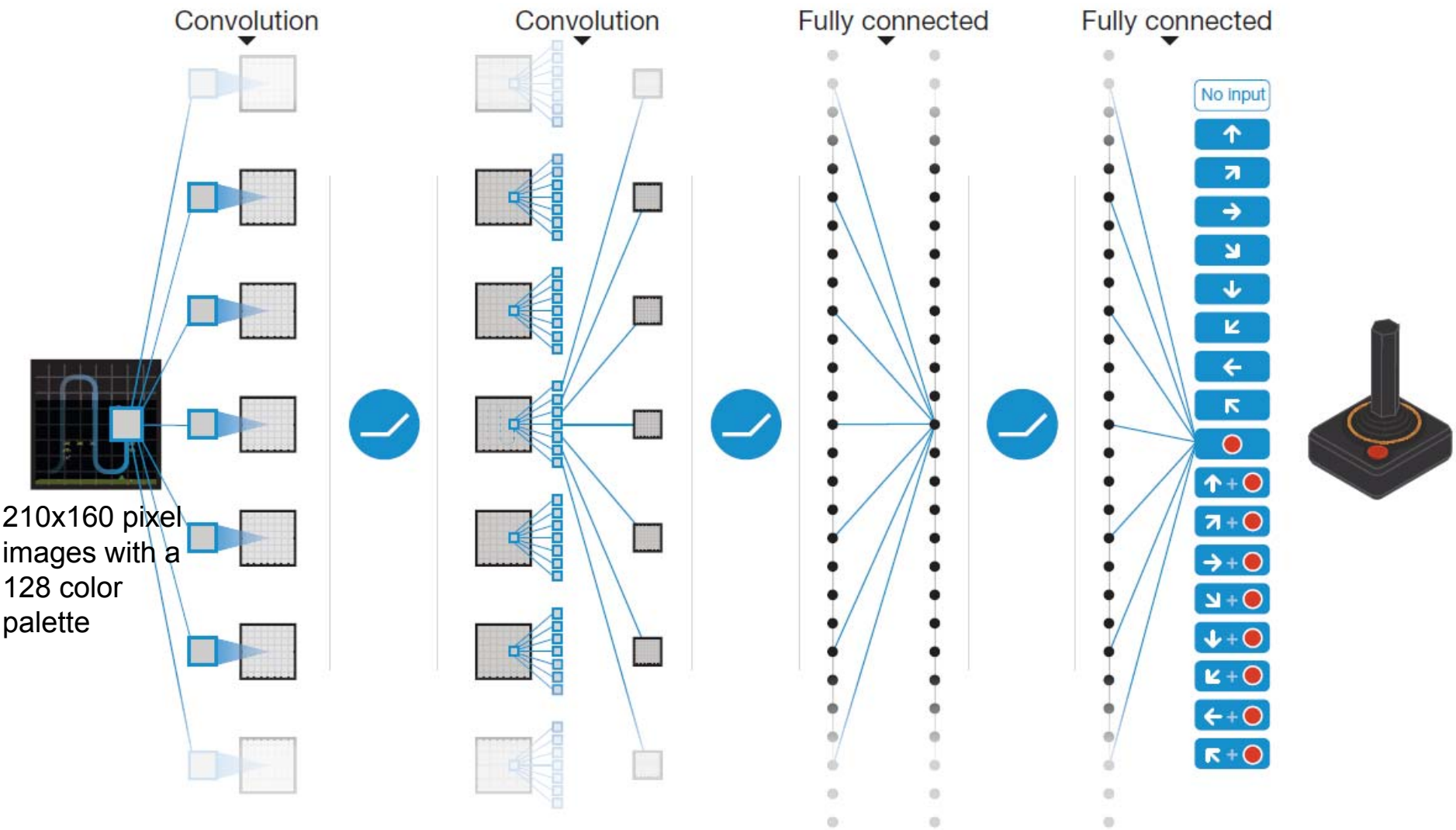
Breakout game

How to define a state?

- Location of the paddle
- Location/direction of the ball
- Presence/absence of each individual brick

Use screen pixels!

DQN in Atari


Convolution
Convolution
Fully connected
Fully connected
No Input
210x160 pixel images with a 128 color palette

Deep Q-Learning

- Naive Q-learning oscillates or diverges with neural nets:
  - Data is sequential:
    - Successive samples are correlated, non-i.i.d.
  - Policy changes rapidly with slight changes to Q-values:
    - Policy may oscillate
    - Distribution of data can swing from one extreme to another
  - Scale of rewards and Q-values is unknown:
    - Naive Q-learning gradients can be largely unstable when backpropagated

Approximate Q-Learning

- Instead of a table, we have a parametrized Q-function $Q_\theta(s, a)$:
  - Can be a linear function in features:

$$Q_\theta(s, a) = \theta_0 f_0(s, a) + \theta_1 f_1(s, a) + \cdots + \theta_n f_n(s, a)$$

  - Or a complicated neural network.
- Learning rule:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} \left(Q_\theta(s, a) - \text{target}(s') \right)^2 \right] |_{\theta=\theta_k}$$

$$\text{target}(s') \triangleq R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$$

Deep Q-Learning

- Deep Q-Network provides a stable solution to deep value-based RL:
  - Use experience replay:
    - Break correlations in data, bring us back to i.i.d. setting
    - Learn from all past policies
    - Using off-policy Q-learning
  - Freeze target Q-network:
    - Avoid oscillations
    - Break correlations between Q-network and target
  - Clip rewards or normalize network adaptively to sensible range:
    - Robust gradients

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \left[\text{target}\right]$$

$$s \leftarrow s'$$

Deep Q-Learning: Experience Replay

- To remove correlations, build data-set from agent's own experience:
  - Take action a_t according to ϵ-greedy policy
  - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$ (Huge data base to store historical samples)
  - Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
  - Optimize MSE between Q-network and Q-learning targets, *e.g.*,

$$L_k(\theta_k) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_{\theta_k}(s, a) - \text{target}(s') \right)^2 \right]$$

$$\text{target}(s') \triangleq R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$$

Deep Q-Learning: Fixed Target Q-Network

- To avoid oscillations, fix parameters used in Q-learning target:
  - Compute Q-learning targets w.r.t. old, fixed parameters θ_k^-:

$$\text{target}(s'; \theta_k^-) \triangleq R(s, a, s') + \gamma \max_{a'} Q_{\theta_k^-}(s', a')$$

  - Optimize MSE between Q-network and Q-learning targets, *e.g.*,

$$L_k(\theta_k) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_{\theta_k}(s, a) - \text{target}(s'; \theta_k^-) \right)^2 \right]$$

  - Periodically update fixed parameters $\theta_k^- = \theta_k$.

Deep Q-Learning: Reward / Value Range

- DQN clips the reward to [-1, +1].
- This prevents Q-values from becoming too large.
- Ensures gradients are well-conditioned.

Stable DQN

DQN

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Training

During Training

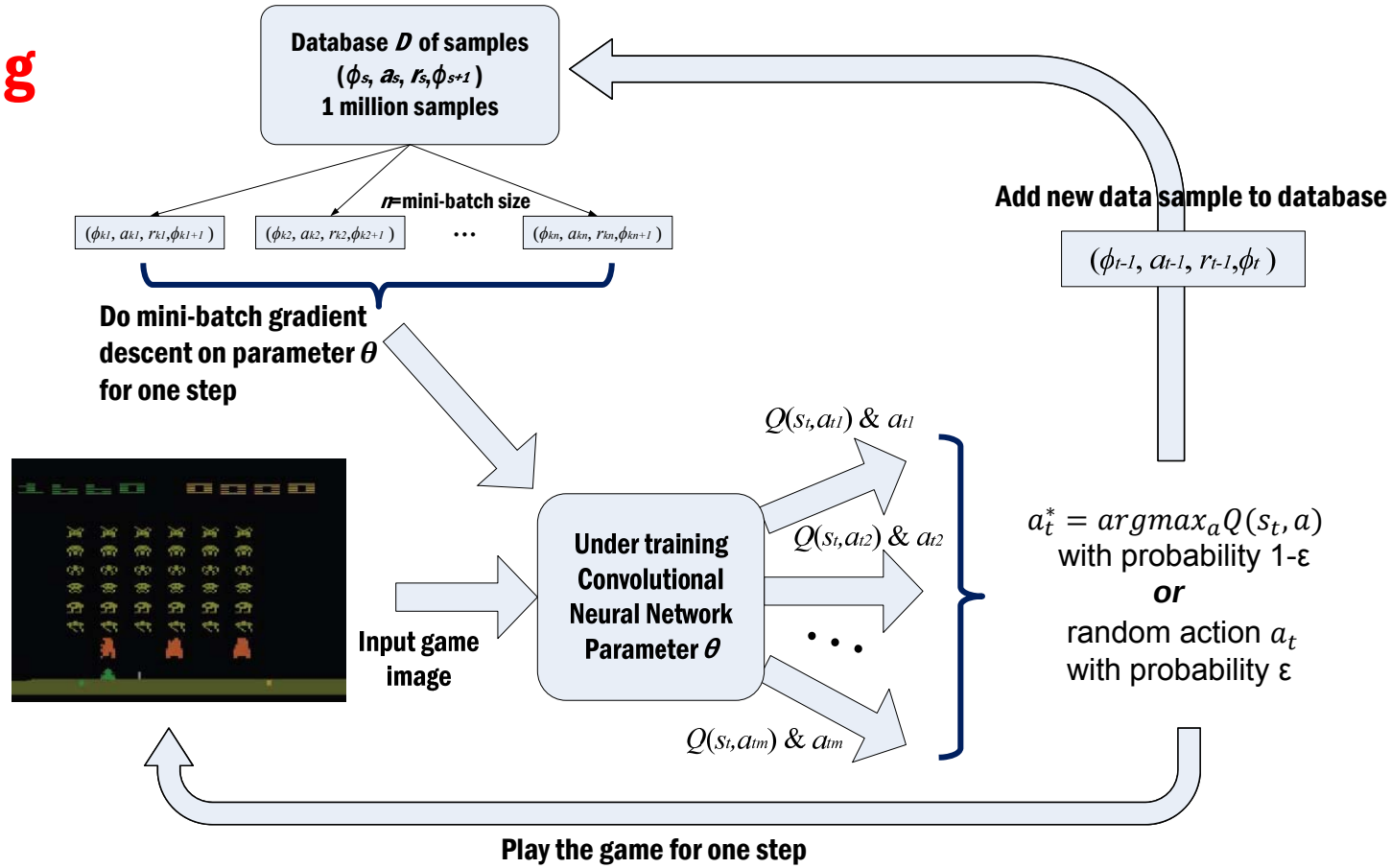


image at time t : x_t

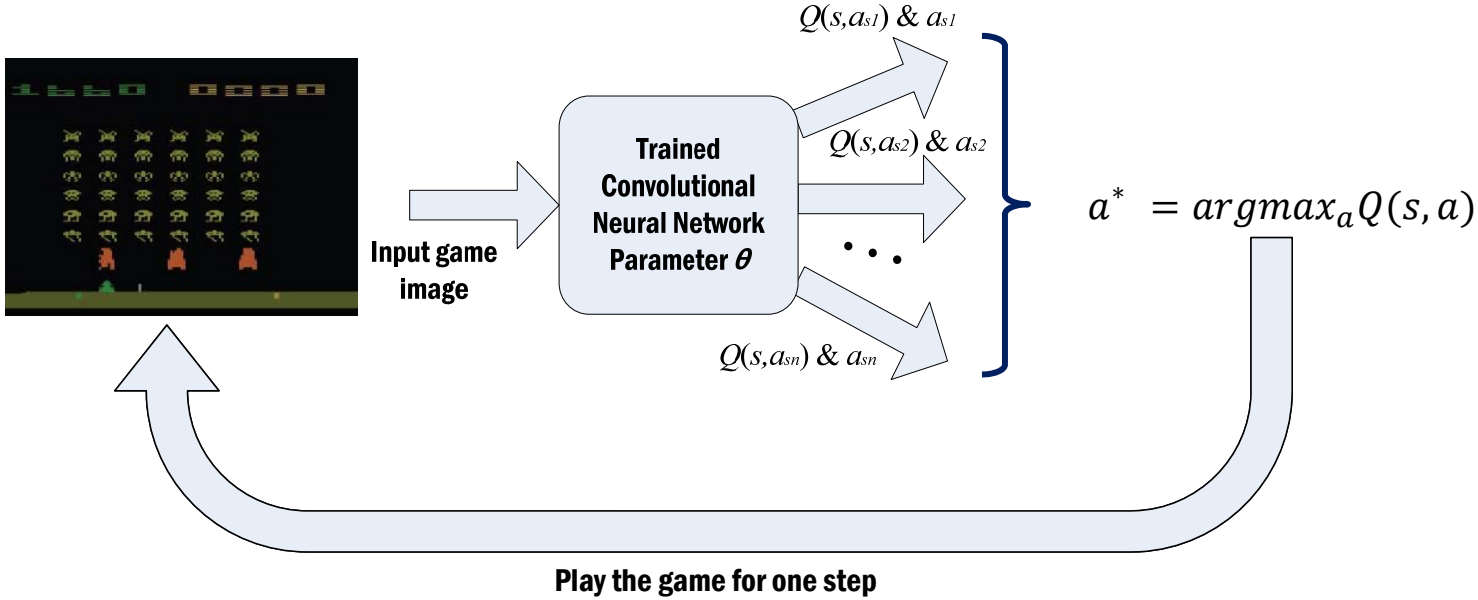
$s_t = s_{t-1}, a_{t-1}, x_t$

preprocessed sequence

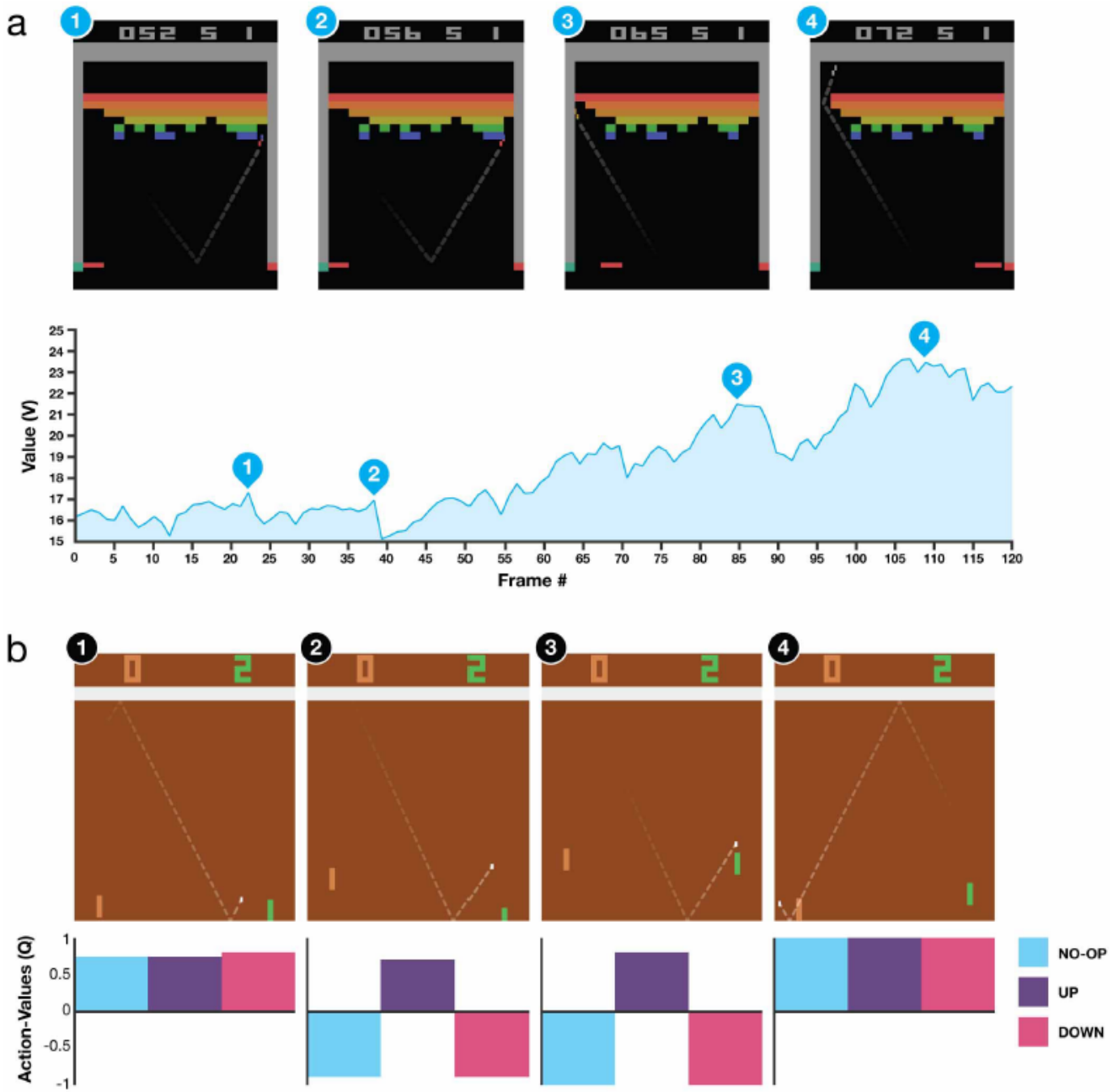
$\phi_t = \phi(s_t)$

Testing

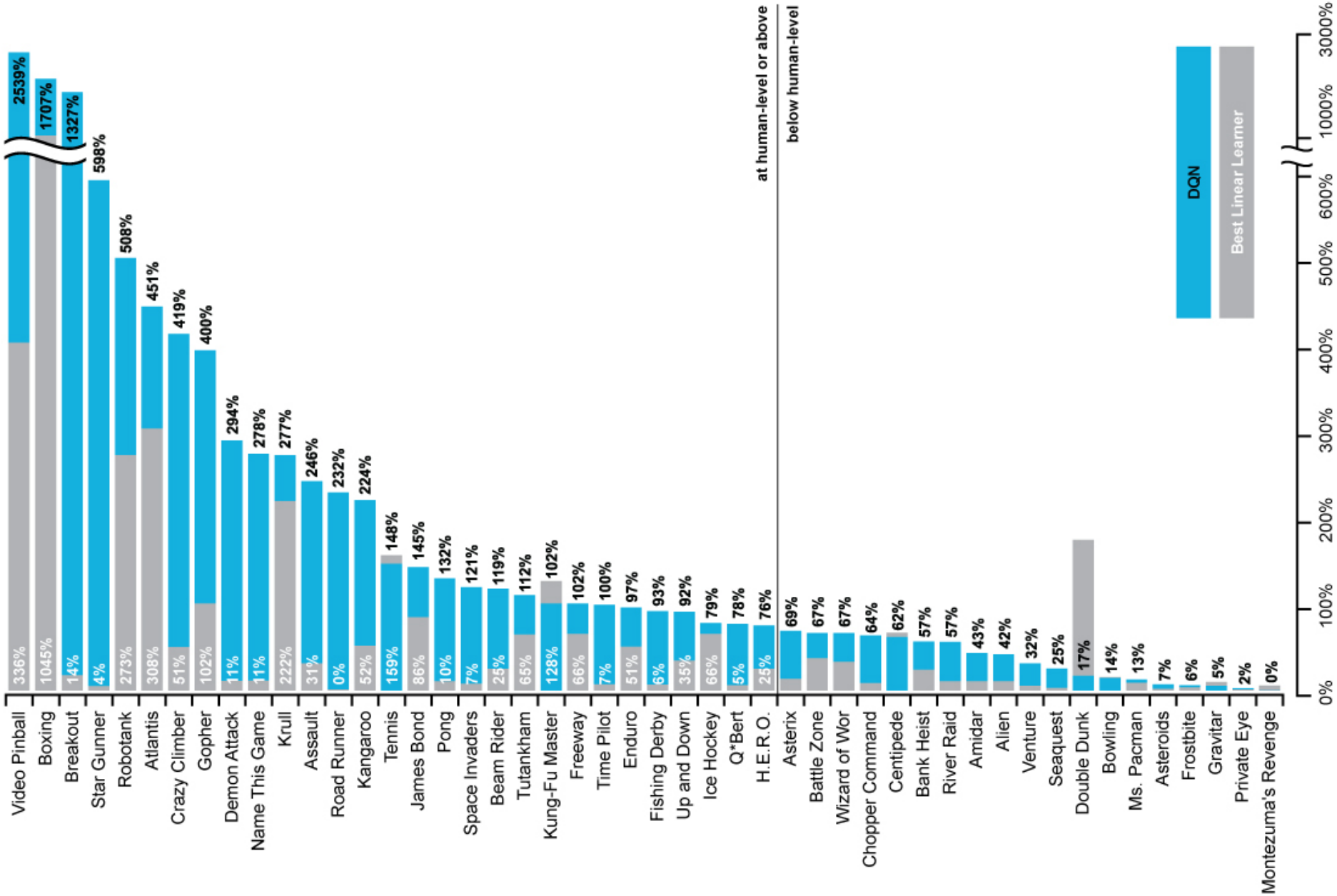
After Training



Results



Results


at human-level or above
below human-level
DQN
Best Linear Learner
0%
100%
200%
300%
400%
500%
600%
1000%
3000%
Video Pinball 336% 2539%
Boxing 1045% 1707%
Breakout 14% 1327%
Star Gunner 4% 598%
Robotank 273% 508%
Atlantis 308% 451%
Crazy Climber 51% 419%
Gopher 102% 400%
Demon Attack 11% 294%
Name This Game 11% 278%
Krull 222% 277%
Assault 31% 246%
Road Runner 0% 232%
Kangaroo 52% 224%
Tennis 159% 148%
James Bond 86% 145%
Pong 10% 132%
Space Invaders 7% 121%
Beam Rider 25% 119%
Tutankham 65% 112%
Kung-Fu Master 128% 102%
Freeway 66% 102%
Time Pilot 7% 100%
Enduro 51% 97%
Fishing Derby 6% 93%
Up and Down 35% 92%
Ice Hockey 66% 79%
Q*Bert 5% 78%
H.E.R.O. 25% 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pacman 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%

Double Q-Learning

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 to evaluate the target:

$$Q_1(s_t, a_t) = Q_1(s_t, a_t) + \alpha \Big(R_{t+1} + Q_2(s_{t+1}, \operatorname*{argmax}_a Q_1(s_{t+1}, a)) - Q_1(s_t, a_t) \Big)$$

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

Deep Q-Learning

$$\text{target}(s'; \theta_k^-) \triangleq R(s, a, s') + \gamma \max_{a'} Q_{\theta_k^-}(s', a')$$

$$L_k(\theta_k) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_{\theta_k}(s, a) - \text{target}(s'; \theta_k^-) \right)^2 \right]$$

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

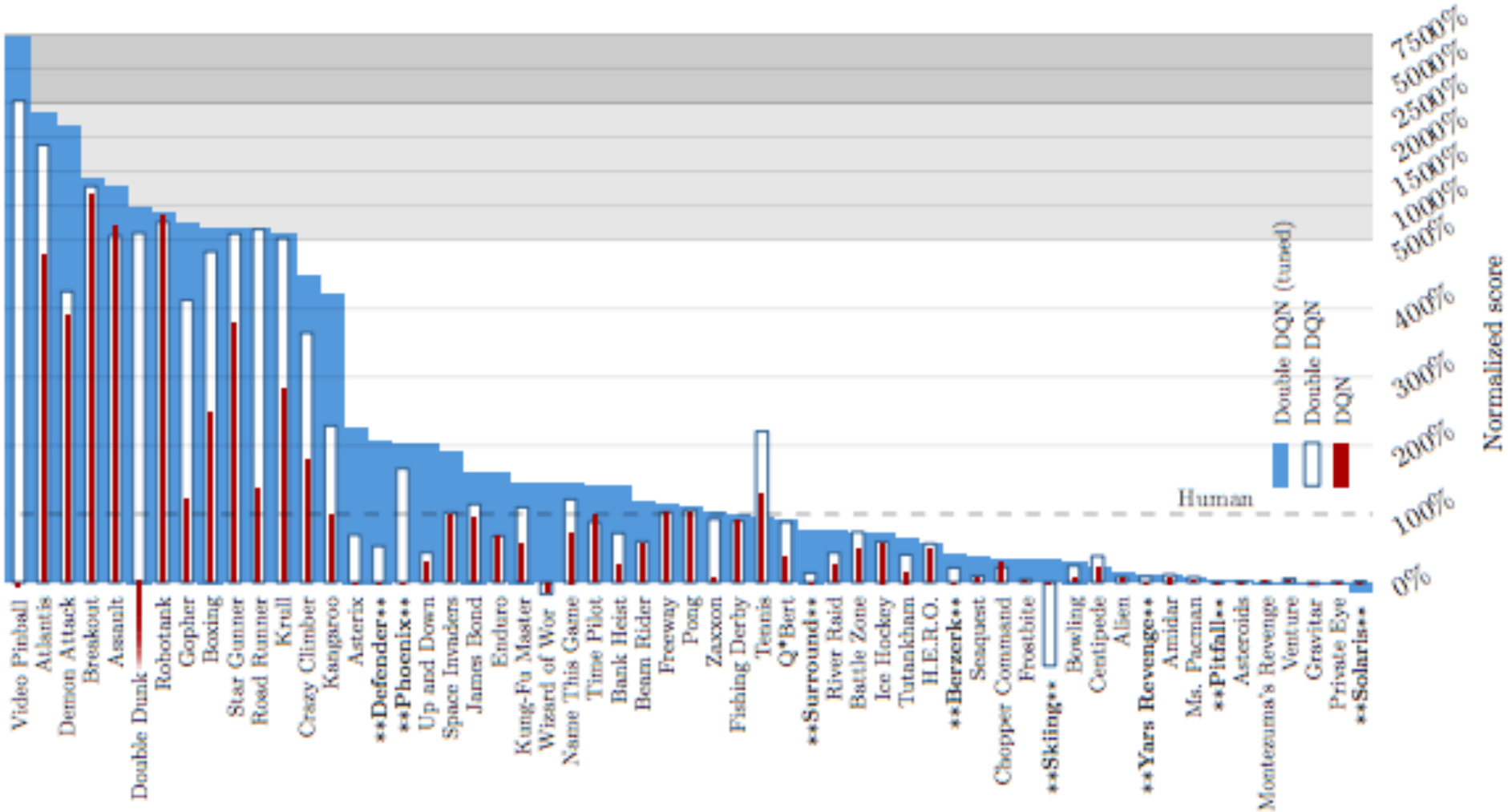
Double Q-Learning

- Current Q-network θ is used to select actions.
- Older Q-network θ^- is used to evaluate actions.

Action evaluation: θ^-

$$L = \left(r + \gamma Q(s', \underbrace{\underset{a'}{\text{argmax}}\, Q(s', a', \mathbf{w})}_{\text{Action selection: } \theta}, \theta^-) - Q(s, a, \theta) \right)^2$$

Double Q-Learning



Dueling Network Architecture

Motivation

Q function should be designed more wisely: containing an action-independent component

For many states:

- unnecessary to estimate the value of each action choice, for example, move left or right only matters when a collision is eminent.
- In most of states, the choice of action has no affect on what happens

Decompose Q

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

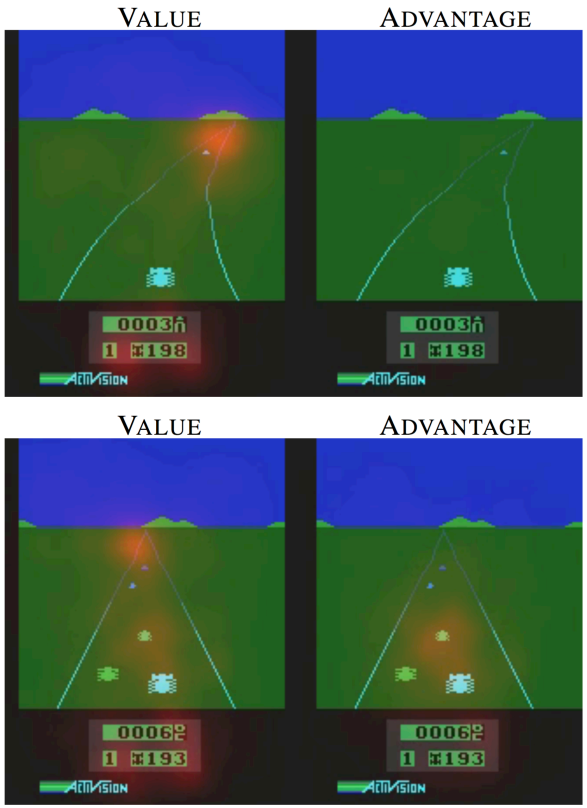
$$\Rightarrow A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) = Q^{\pi}(s,a) - \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a)]$$

(advantage function)

Saliency map on the Atari game Enduro

1. Focus on **horizon**, where new **cars appear**

2. Focus on the **score**

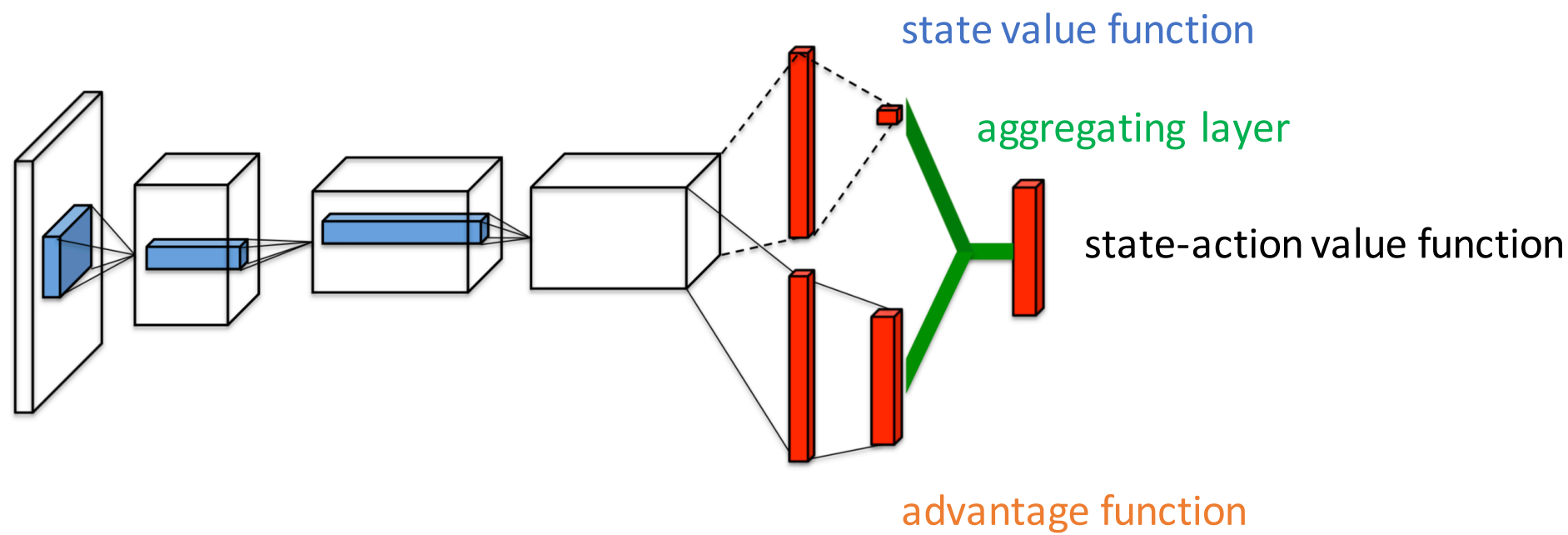


Not pay much attention when there are **no cars** in front

Attention on car immediately in front making **its choice of action very relevant**

Dueling Network

- Single Q-network with two streams.
- Produce separate estimations of state value function V and advantage function A.



- outputs a scalar value $V(s)$ and a $|\mathcal{A}|$-dimensional vector $A(s, a)$.

Aggregation Modules

- Add: $Q(s, a) = V(s) + A(s, a)$.
- Subtract max: $Q(s, a) = V(s) + (A(s, a) - \max_{a'} A(s, a'))$:
  - force A to have zero at the chosen optimal action.
  - Equivalent to shifting V.
- Subtract mean: $Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a')\right)$:
  - loses the original semantics of V and A.
  - but increases the stability of the optimization.
  - often works best.

Experiments

Table 1. Mean and median scores across all 57 Atari games, measured in percentages of human performance.

	30 no-ops		Human Starts	
	Mean	**Median**	**Mean**	**Median**
Prior. Duel Clip	**591.9%**	**172.1%**	**567.0%**	**115.3%**
Prior. Single	434.6%	123.7%	386.7%	112.9%
Duel Clip	**373.1%**	**151.5%**	**343.8%**	**117.1%**
Single Clip	341.2%	132.6%	302.8%	114.1%
Single	307.3%	117.8%	332.9%	110.9%
Nature DQN	227.9%	79.1%	219.6%	68.5%