Section 4: Deep Reinforcement Learning

Aaron Effron, Vivekkumar Patel

Goals for today's section:

Review MDPs

Review RL

RL application: breakout

Introduce Deep Reinforcement Learning

Markov Decision Processes

A MDP is a search problem where **transitions are random** and instead of minimizing cost, we are **maximizing reward**.

Definition: Markov decision process

States: the set of states

$s_{\text{start}} \in \text{States}$: starting state

$\text{Actions}(s)$: possible actions from state s

$T(s, a, s')$: probability of s' if take action a in state s

$\text{Reward}(s, a, s')$: reward for the transition (s, a, s')

$\text{IsEnd}(s)$: whether at end of game

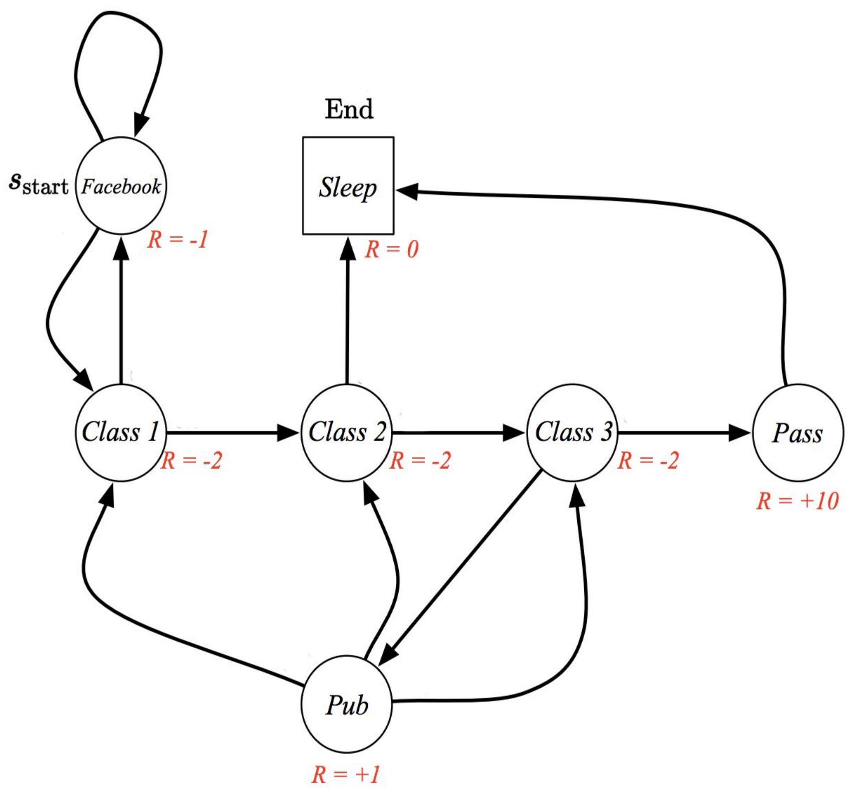
$0 \leq \gamma \leq 1$: discount factor (default: 1)

Example MDP: Life of a Student

An action is selecting an arrow.

You have a 20% chance of "slipping" and taking the non-selected arrow.

What's the
best way to navigate
This world?



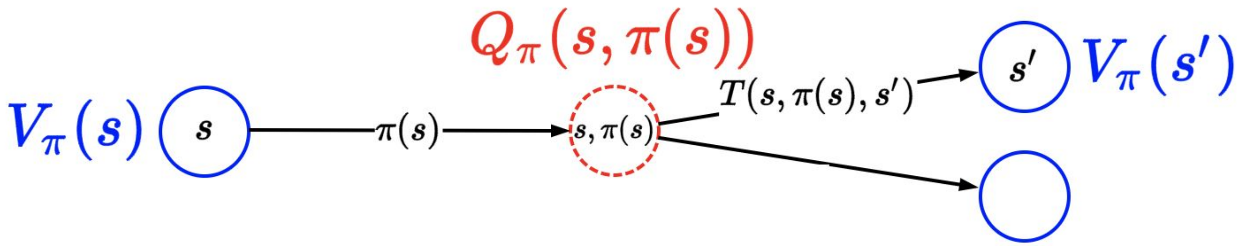
What can we do with an MDP?

(1) Given a policy π,
we can generate an *episode*

(episodes are **RANDOM**)

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$$

(2) Given a policy π, we can *evaluate it*



Utility$_\pi$

$$u_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots$$

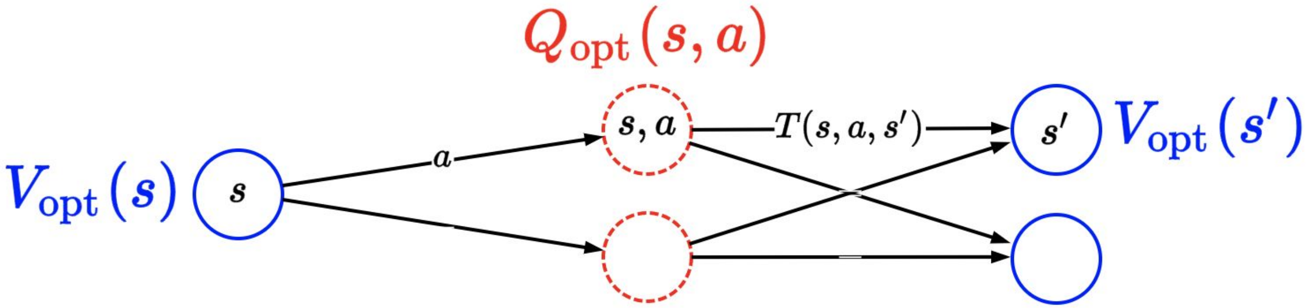
Value$_\pi$

$$V_\pi(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ Q_\pi(s, \pi(s)) & \text{otherwise.} \end{cases}$$

Q-Value$_\pi$

$$Q_\pi(s, a) = \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_\pi(s')]$$

(3) If we *don't* have π, we can compute the *optimal* policy



Value$_{\text{opt}}$

$$V_{\text{opt}}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s,a) & \text{otherwise.} \end{cases}$$

Q-Value$_{\text{opt}}$

$$Q_{\text{opt}}(s,a) = \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}(s')].$$

π_{opt}

$$\arg \max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s,a)$$

π vs opt

$$V_\pi(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ Q_\pi(s, \pi(s)) & \text{otherwise.} \end{cases}$$

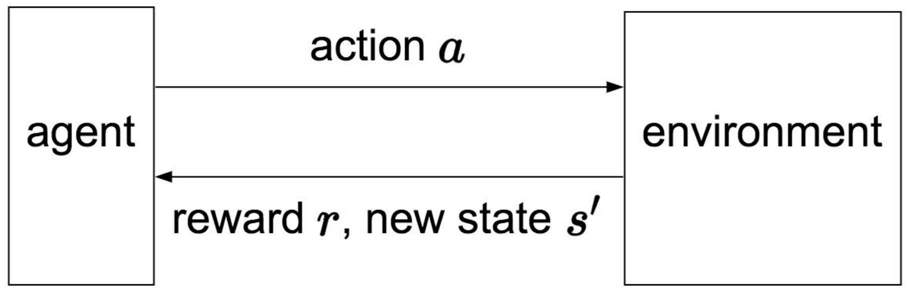
$$V_{\text{opt}}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \boxed{\max_{a \in \text{Actions}(s)}} Q_{\text{opt}}(s, a) & \text{otherwise.} \end{cases}$$

$$Q_\pi(s, a) = \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_\pi(s')]$$

$$Q_{\text{opt}}(s, a) = \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_{\text{opt}}(s')].$$

Reinforcement Learning

Reinforcement Learning



Definition: Markov decision process

States: the set of states
$s_{\text{start}} \in \text{States}$: starting state
$\text{Actions}(s)$: possible actions from state s
~~$T(s, a, s')$: probability of s' if take action a in state s~~
~~$\text{Reward}(s, a, s')$: reward for the transition (s, a, s')~~
$\text{IsEnd}(s)$: whether at end of game
$0 \leq \gamma \leq 1$: discount factor (default: 1)

What can we do with RL?

(1) Given a policy π, we can *evaluate it*

Model-Based

$$\hat{T}(s,a,s') = \frac{\#\text{ times }(s,a,s')\text{ occurs}}{\#\text{ times }(s,a)\text{ occurs}}$$

$$\widehat{\text{Reward}}(s,a,s') = \text{average of } r \text{ in } (s,a,r,s')$$

Model-Free

Monte Carlo

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta u$$

SARSA

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta[\underbrace{r}_{\text{data}} + \underbrace{\gamma\hat{Q}_\pi(s',a')}_{\text{estimate}}]$$

(2) we can compute an *Optimal* policy π_{opt}

Q-learning

Algorithm: Q-learning [Watkins/Dayan, 1992]

On each (s, a, r, s'):

$$\hat{Q}_{\text{opt}}(s,a) \leftarrow (1-\eta)\underbrace{\hat{Q}_{\text{opt}}(s,a)}_{\text{prediction}} + \eta\underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}]$$

Recall: $\hat{V}_{\text{opt}}(s') = \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s', a')$

Function approximation

Idea: make Q a *model* instead of a lookup table, and describe your environment with features

$$\min_{\mathbf{w}} \sum_{(s,a,r,s')} (\underbrace{\hat{Q}_{\text{opt}}(s,a;\mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}})^2$$

RL application: Breakout!

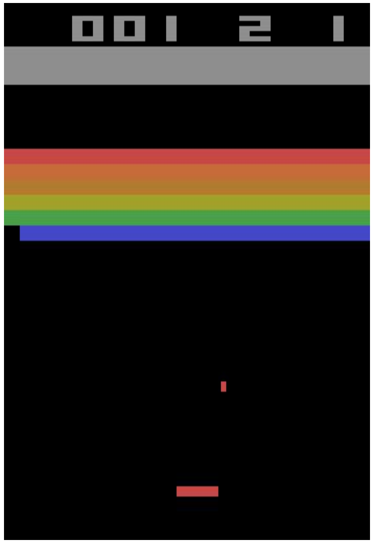
Breakout Game Description

Formally:

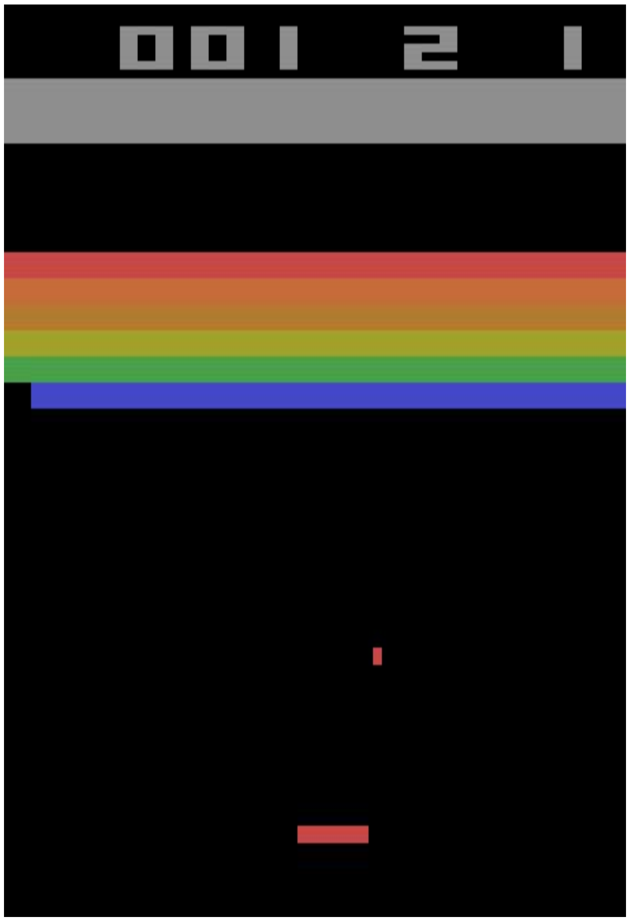
- *Actions*
  - move_paddle_left
  - move_paddle_right
  - do_not_move_paddle
- *Rewards*
  - If ball hits brick, reward = 1
  - Otherwise, reward = 0
- *End condition*
  - If ball falls off the screen, game ends

Can we learn to control an agent directly from sensory input?

In Breakout, sensory input would be a game screen frame.



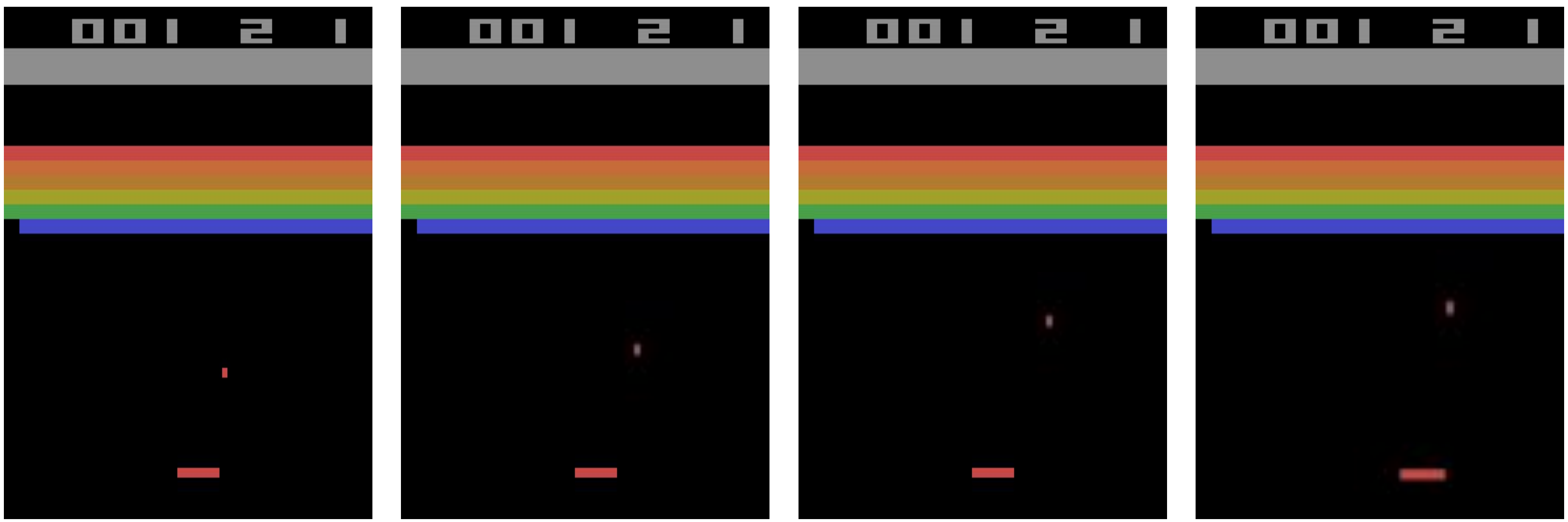
Finding a state representation



Consider this frame.

- Can you capture information like direction of the ball?
- Can you capture velocity?

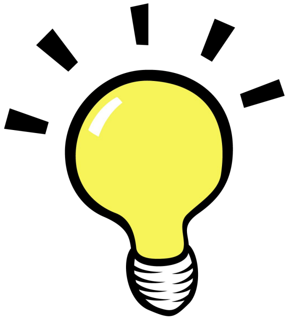
Use a small number of consecutive frames for each state.



PROBLEM!!!

$$\# \text{ states} \approx 256^{84\text{x}84\text{X}4}$$

(assume 84x84 pixels per screenshot, where each pixel can take on 256 values, and 4 screenshots per state)



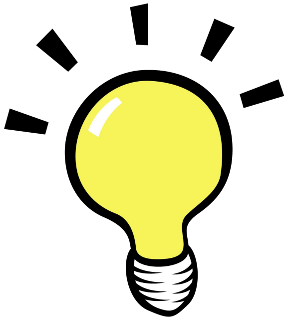
Use function approximation!

featurize our state space!

1st try: hand-designing features $\Phi(s, a)$

- Performance depends on the quality of features **Φ(s, a)**
- **Not generalized**
- Doesn’t scale well with game complexity

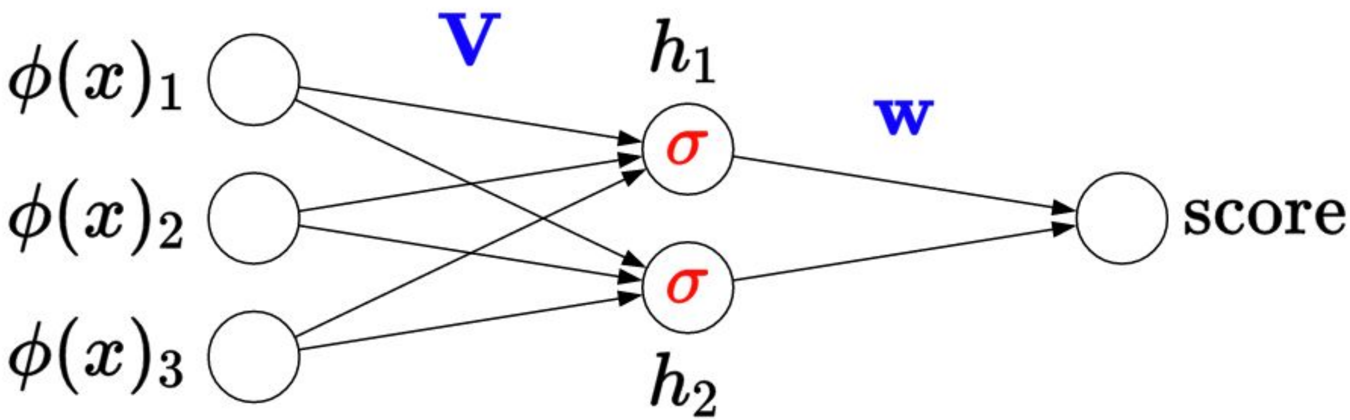
→ ***Handcrafting features is very difficult!***



What if we automatically learned features from the pixels?

Deep Neural Nets (Review)

Neural network:



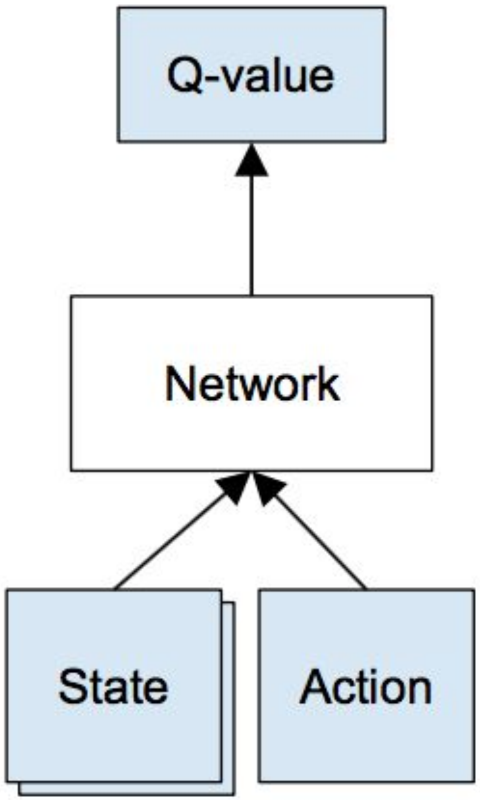
Intermediate hidden units:

$$h_j = \sigma(\mathbf{v}_j \cdot \phi(x)) \quad \sigma(z) = (1 + e^{-z})^{-1}$$

Output:

$$\text{score} = \mathbf{w} \cdot \mathbf{h}$$

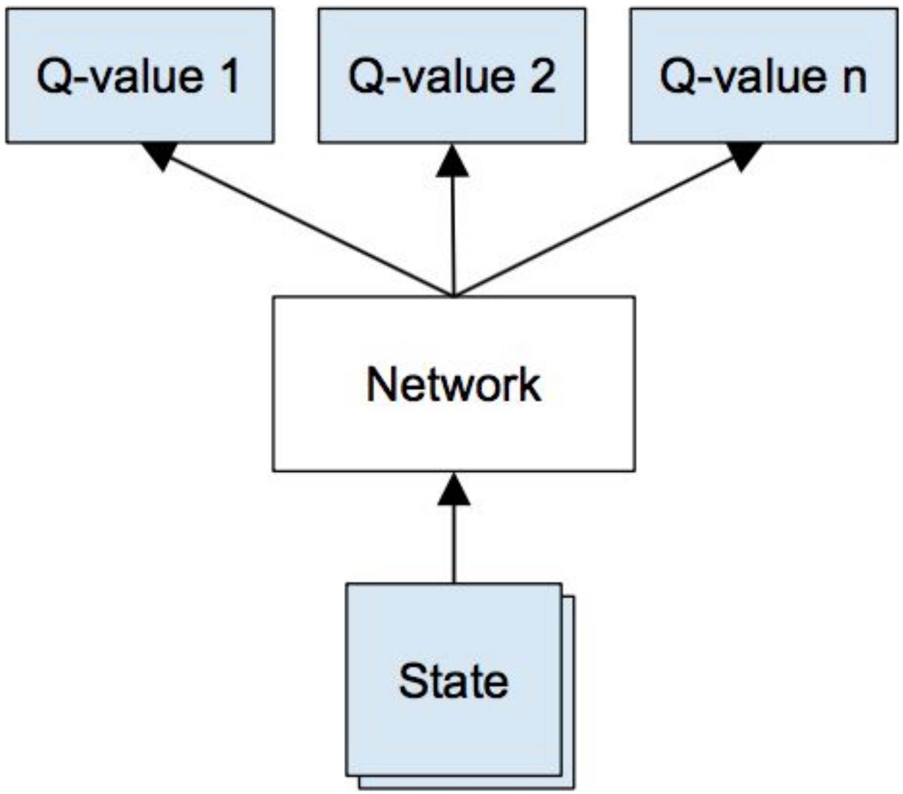
Neural Networks as Q(s, a) approximators



- Input (s,a) pair to neural network.
- Neural network predicts the Q-value for (s,a) pair

Can we make this even more efficient?

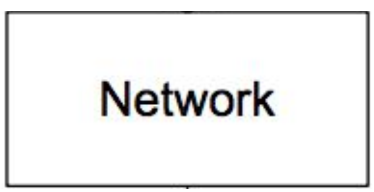
Neural Networks as Q(s, a) approximators



- State is the only input into the neural network.
- Network outputs a Q-value for every possible action.
- Action corresponding to the highest Q-value is chosen.

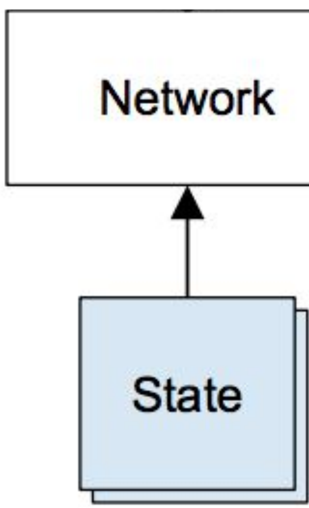
A single network to predict Q(s,a) for all possible states and actions!

Training Deep-Q-Networks (DQN)



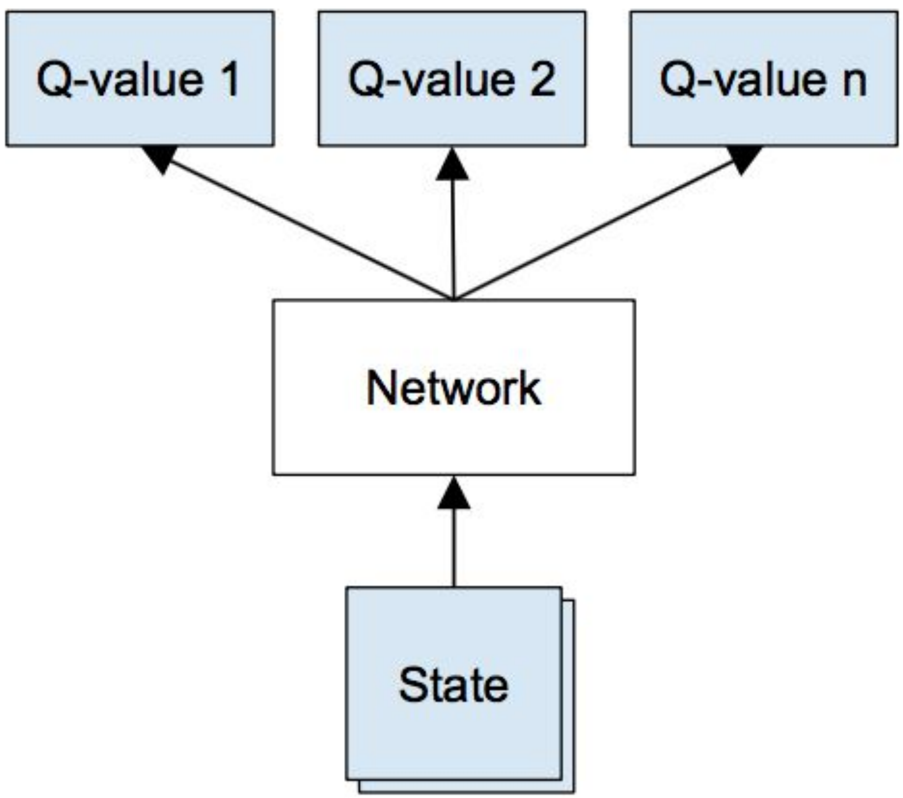
Initialize weights randomly!

Training Deep-Q-Networks (DQN)



States are passed as input.

Training Deep-Q-Networks (DQN)

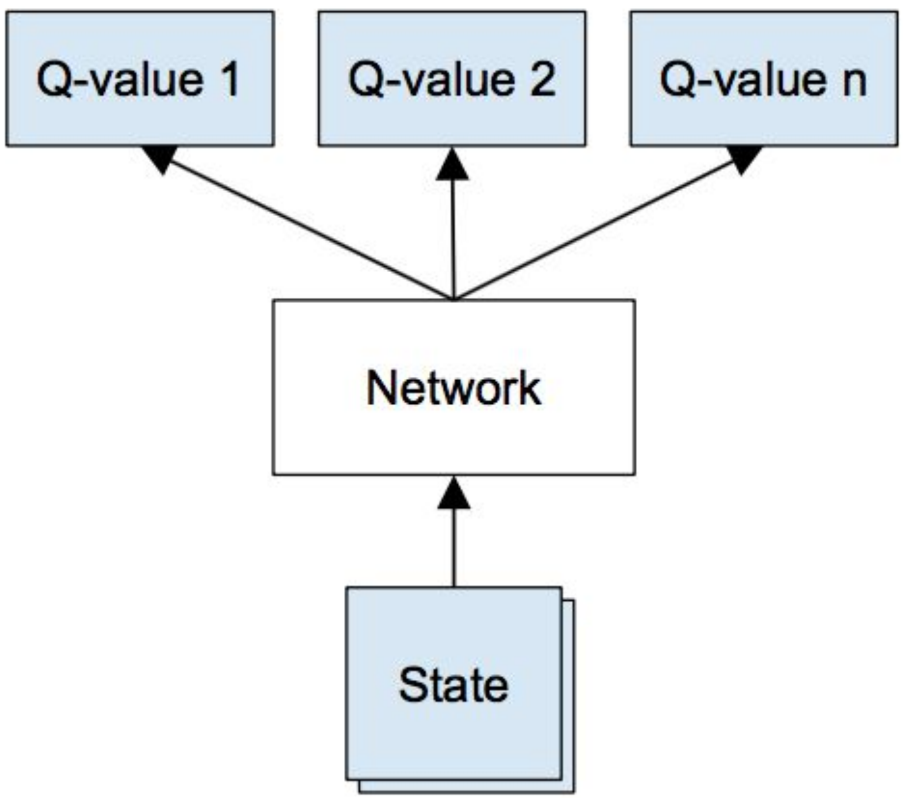


- Network outputs Q-values for each possible action.
- Action space for Breakout:

  [left, right, no-op]

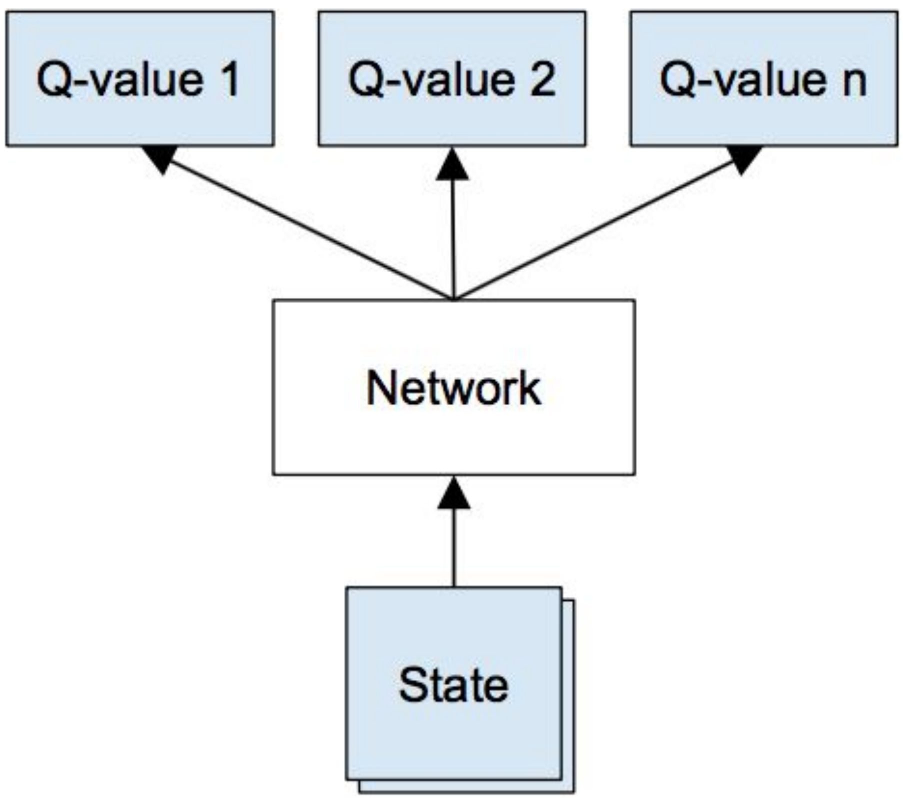
- Since initial weights are random, Q-values are random.

Training Deep-Q-Networks (DQN)



- Execute action that maximizes Q-value.
- Environment may or may not react to that action with a reward.
- Obtain next state.

Training Deep-Q-Networks (DQN)



- Run gradient descent on Q-learning loss.

Training Deep-Q-Networks

- Initialize weights randomly.
- Loop:
  - Obtain current state (s)
  - Run Neural Network on s to obtain Q-value for every action.
  - Execute action (a) that maximizes Q-value.
  - Obtain reward (r) and new state (s').
  - Perform gradient descent on Q-learning loss using (s, a, r, s')

$$\min_{\mathbf{w}} \sum_{(s,a,r,s')} (\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}})^2$$

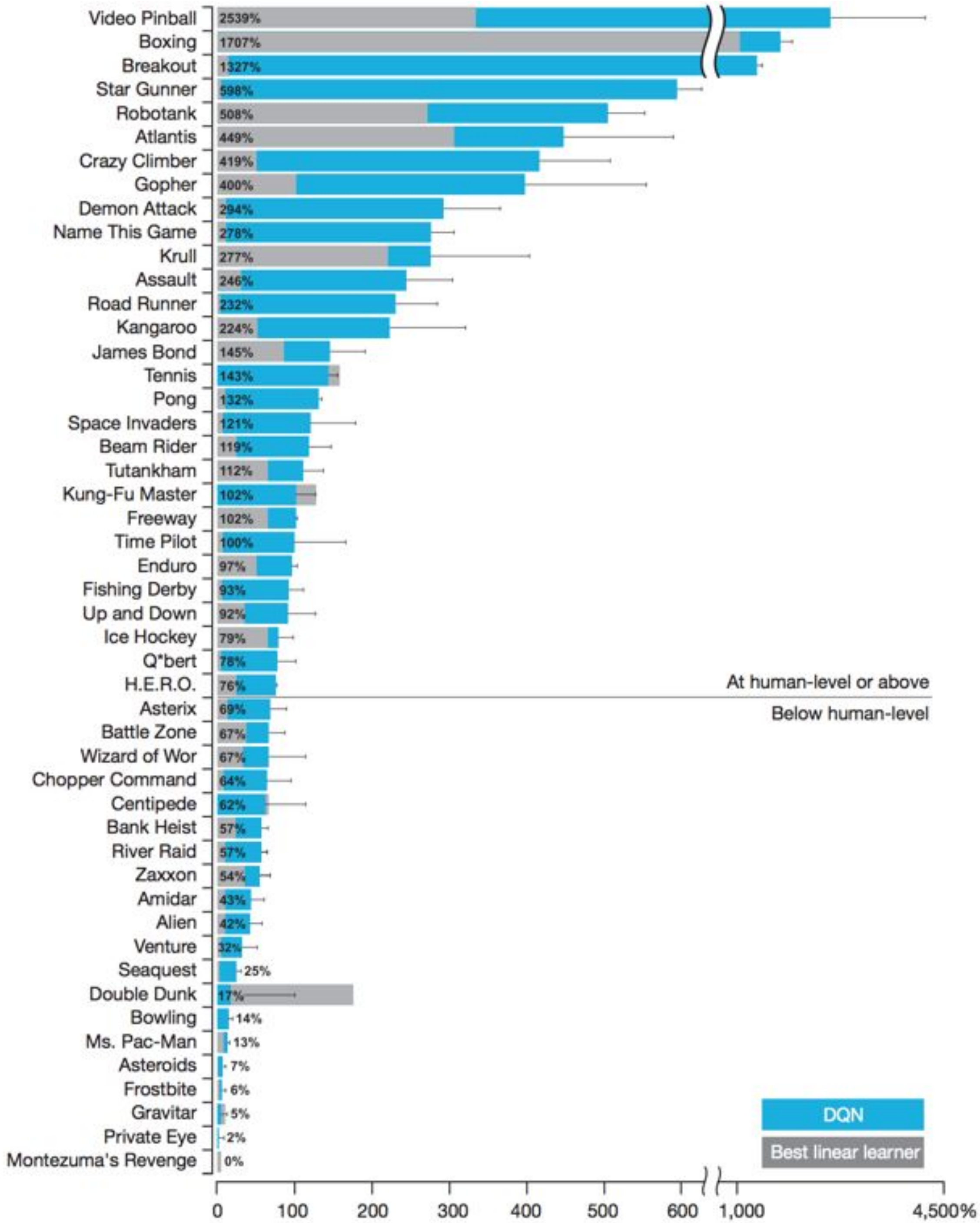
Training Deep-Q-Networks - Additional Considerations

- Initialize weights randomly
- Initialize memory (D) with capacity N
- Loop:
  - Obtain current state (s)
  - Run Neural Network on s to obtain Q-value for every action
  - With probability ε, execute random action (a)
  - Otherwise, execute action (a) that maximizes Q-value
  - Obtain reward (r) and new state (s’)
  - Store (s, a, r, s’) in D
  - Randomly sample $(s, a, r, s')_D$ from D
  - Perform gradient descent on Q-learning loss using $(s, a, r, s')_D$

Let’s watch Deep RL in action

https://www.youtube.com/watch?v=V1eYniJ0Rnk

How well does DQN work on other Atari games?



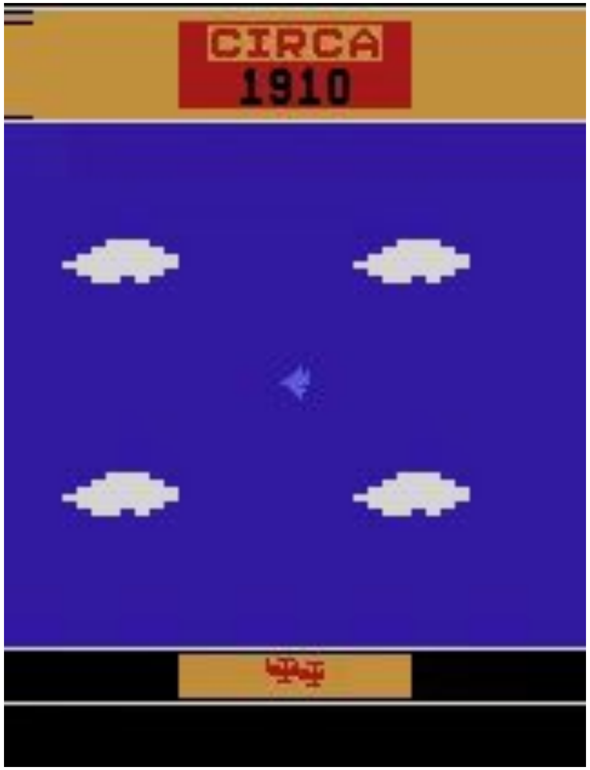
Comparison of the DQN agent with the best RL methods in the literature

The performance of DQN is normalized w.r.t. A professional human games tester (that is, 100% level) and random play (that is, 0% level).

Source: Mnih et al. (2015)

Video Pinball

(DQN does very well, 2539% of human performance)



Time Pilot

(DQN reaches human gaming performance)

Montezuma’s Revenge

(DQN does very poorly, similar to random gameplay)

Shortcomings of DQN

- Does not work well if environment has sparse, delayed rewards
- Does not work well in continuous action space
- Multi-agent co-operation
- Transfer across games

Active Area of Research: Transfer Learning

Inspiration: Humans can train on tasks A, B and apply knowledge to new task C.

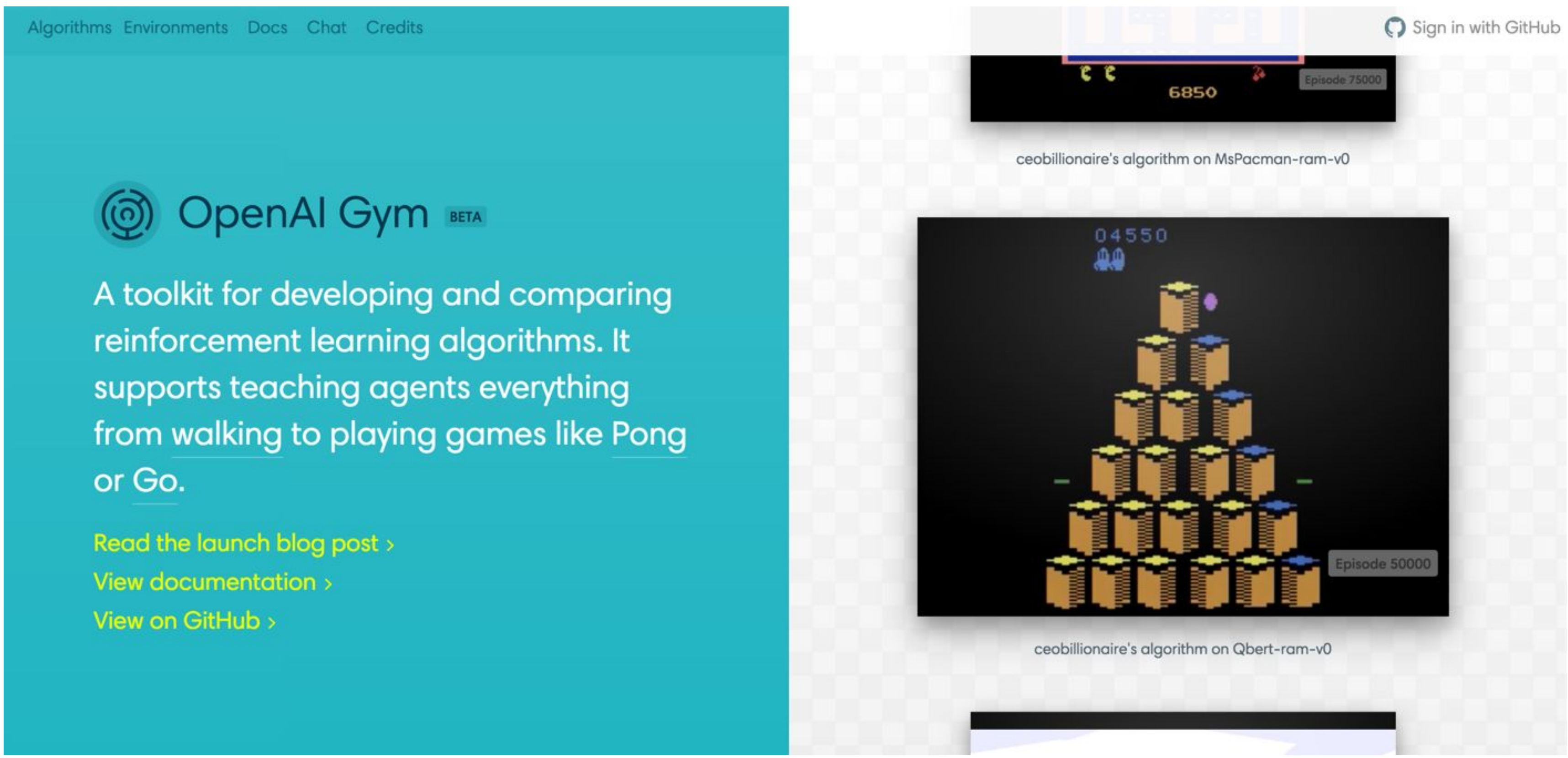
Can we train DQN across multiple tasks and “transfer” knowledge?

For example:

- Train DQN on multiple Atari games
- Train DQN on one game (e.g. Pong), then fine-tune network on another (e.g. Breakout)

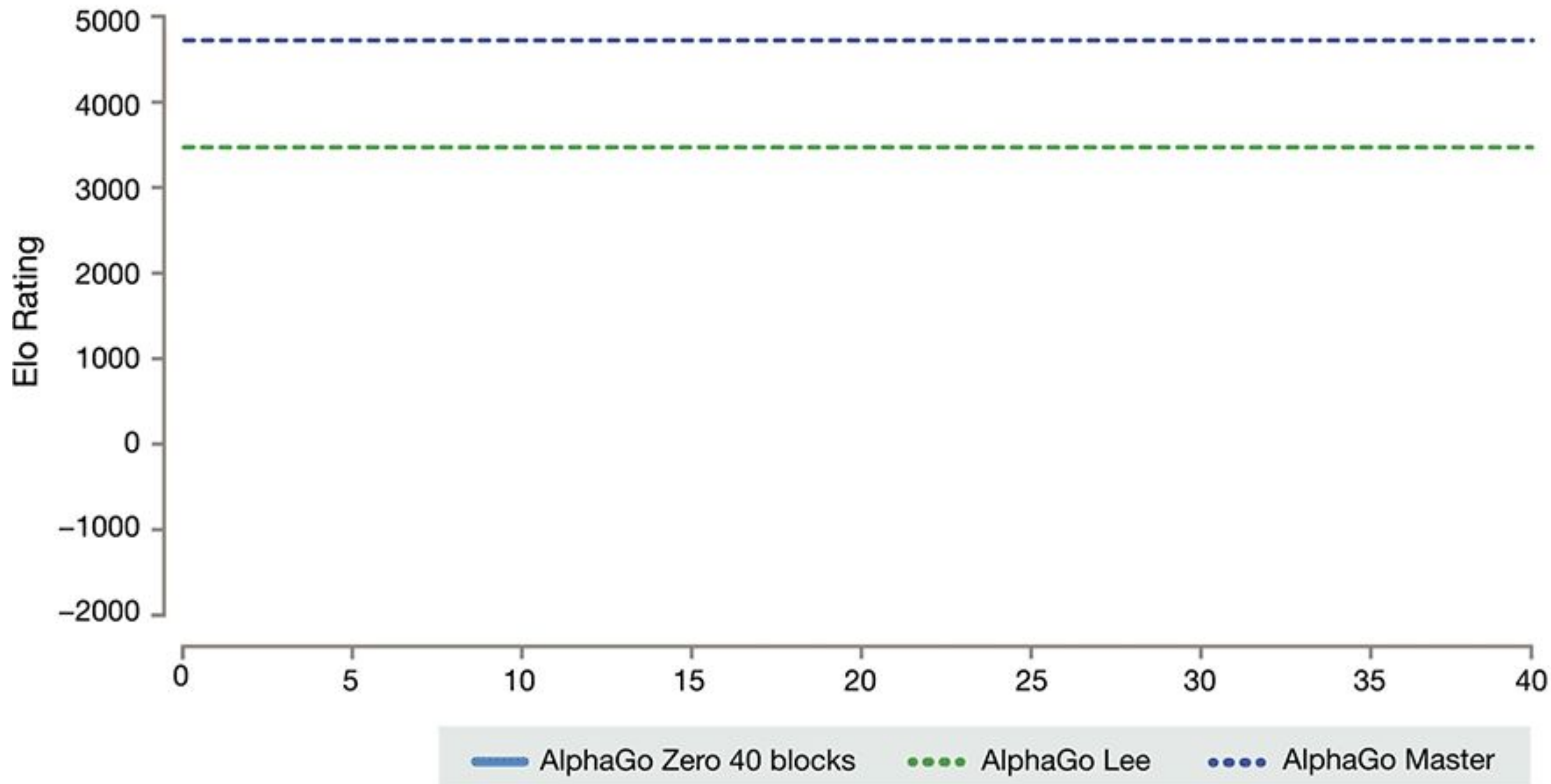
Try it out yourself!

OpenAI Gym offers many RL environments that you can play with, including Atari games.



More RL Applications!

AlphaGo “Zero”



Summary

Deep Reinforcement Learning: combination of techniques we’ve learned in class

DQN: A deep neural net that predicts the Q values for all actions with sensory input

Learning from just sensory input and no domain knowledge → step towards artificial general intelligence (AGI)


001 2 1
You won!

Sources

[1] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

[2] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).