# Reinforcement Learning

# Reinforcement Learning Overview

- We've focused a lot on supervised learning:
  - Training examples: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2),\ldots$

## Reinforcement Learning Overview

- We've focused a lot on supervised learning:
  - Training examples: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots$
- But consider a different type of learning problem, in which a robot has to learn to do tasks in a particular environment.

## Reinforcement Learning Overview

- We've focused a lot on supervised learning:
  - Training examples: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots$
- But consider a different type of learning problem, in which a robot has to learn to do tasks in a particular environment.
  - E.g.,
    - Navigate without crashing into anything

# Reinforcement Learning Overview

- We've focused a lot on supervised learning:
  - Training examples: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots$

- But consider a different type of learning problem, in which a robot has to learn to do tasks in a particular environment.
  - E.g.,
    - Navigate without crashing into anything
    - Locate and retrieve an object

## Reinforcement Learning Overview

- We’ve focused a lot on supervised learning:
  - Training examples: $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$,...

- But consider a different type of learning problem, in which a robot has to learn to do tasks in a particular environment.
  - E.g.,
    - Navigate without crashing into anything
    - Locate and retrieve an object
    - Perform some multi-step manipulation of objects resulting in a desired configuration (e.g., sorting objects)

- This type of problem doesn’t typically provide clear “training examples” with detailed feedback at each step.

- This type of problem doesn’t typically provide clear “training examples” with detailed feedback at each step.

- Rather, robot gets intermittent “rewards” for doing “the right thing”, or “punishment” for doing “the wrong thing”.

- This type of problem doesn’t typically provide clear “training examples” with detailed feedback at each step.

- Rather, robot gets intermittent “rewards” for doing “the right thing”, or “punishment” for doing “the wrong thing”.

- Goal: To have robot (or other learning system) learn, based on such intermittent rewards, what action to take in a given situation.

- This type of problem doesn’t typically provide clear “training examples” with detailed feedback at each step.

- Rather, robot gets intermittent “rewards” for doing “the right thing”, or “punishment” for doing “the wrong thing”.

- Goal: To have robot (or other learning system) learn, based on such intermittent rewards, what action to take in a given situation.

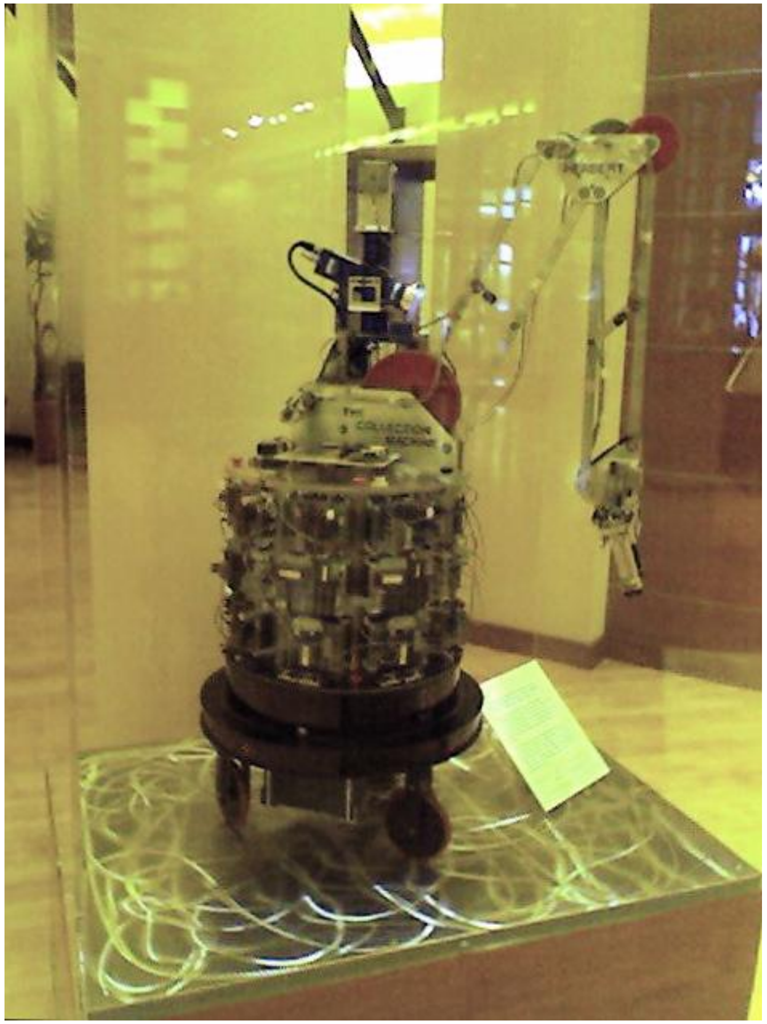
- Ideas for this have been inspired by “reinforcement learning” experiments in psychology literature.

## Applications of reinforcement learning: A few examples

- Learning to play backgammon (and more recently, Go)
- Learning to play video games
- Robot arm control (juggling)
- Robo-soccer
- Robot navigation
- Robot helicopoter
- Elevator dispatching
- Power systems stability control

# Herbert, the Soda-Can Collecting Robot

## (Connell, Brooks, Ning; MIT, 1980s)



http://cyberneticzoo.com/cyberneticanimals/1986c-herbert-the-collection-machine-brooks-connell-ning-a
merican/

# **Robby the Robot** can learn via reinforcement learning

Sensors:

H(ere), N,S,E,W,

***"policy" = "strategy"***

Actions:

Move N

Move S

Move E

Move W

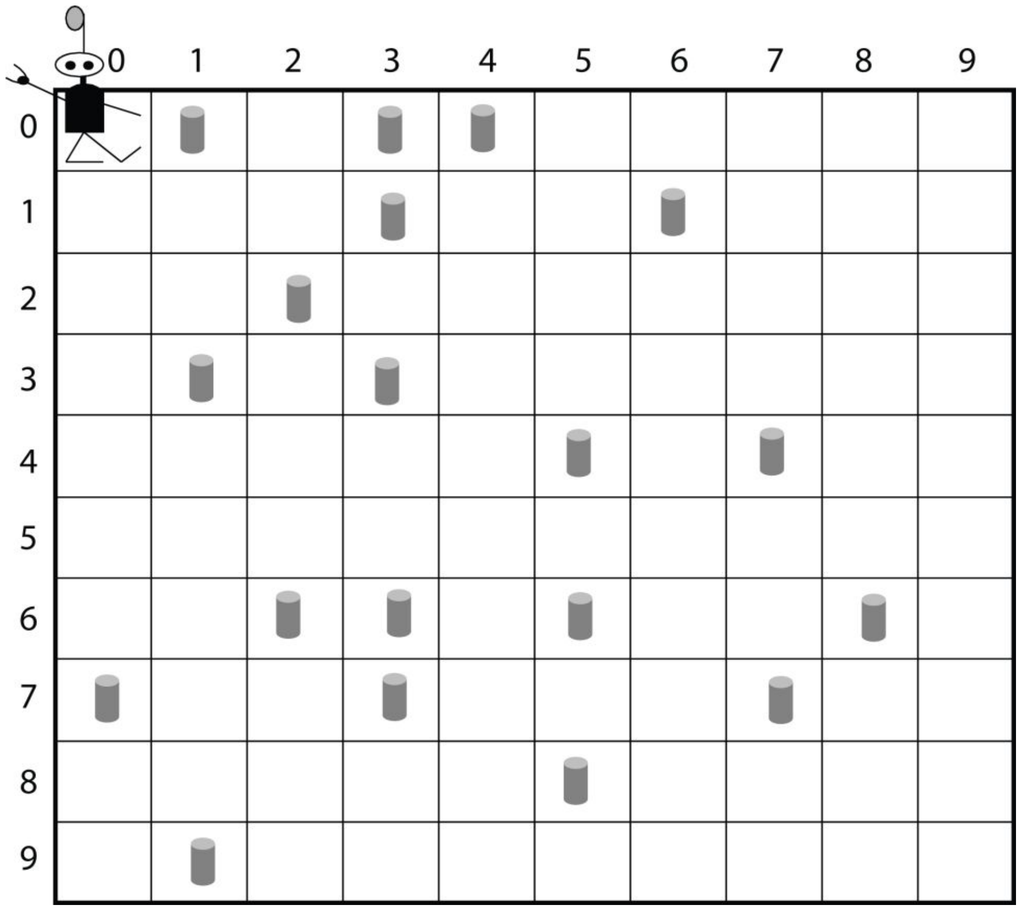
Pick up can

Rewards/Penalties (points):

Picks up can: 10

Pick up can on empty site: -1

Crashes into wall: -5



## Formalization

Reinforcement learning is typically formalized as a **Markov Decision Process** (MDP):

Agent *L only knows current state and actions available from that* state.

Agent *L* can:

- perceive a set *S* of distinct states of an environment
- perform a set *A* of actions.

# Formalization

Reinforcement learning is typically formalized as a **Markov Decision Process** (MDP):

Agent *L only knows current state and actions available from that* state.

Agent *L* can:

- perceive a set *S* of distinct states of an environment
- perform a set *A* of actions.

Components of Markov Decision Process (MDP):

- Possible set *S* of states (state space)
- Possible set *A* of actions (action space)
- State transition function (unknown to learner L)

  $\delta : S \times A \longrightarrow S \qquad \delta(s_t, a_t) = s_{t+1}$

- Reward function (unknown to learner L)

  $r : S \times A \longrightarrow \Re \qquad r(s_t, a_t) = r_t$

# Formalization

Reinforcement learning is typically formalized as a **Markov Decision Process** (MDP):

Agent *L only knows current state and actions available from that* state.

Agent *L* can:

- perceive a set *S* of distinct states of an environment
- perform a set *A* of actions.

Components of Markov Decision Process (MDP):

- Possible set *S* of states (state space)
- Possible set *A* of actions (action space)
- State transition function (unknown to learner L)

$$\delta : S \times A \longrightarrow S \qquad \delta(s_t, a_t) = s_{t+1}$$

- Reward function (unknown to learner L)

$$r : S \times A \longrightarrow \Re \qquad r(s_t, a_t) = r_t$$

Note: Both $\delta$ and *r* can be deterministic or probabilistic.

In the Robby the Robot example, they are both deterministic

Goal is for agent $L$ to learn policy $\pi$, $\pi : S \longrightarrow A$, such that $\pi$ maximizes cumulative reward.

## What should the system learn?

**Policy Learning:**

Learn function $\pi$ *that directly* maps states to actions:

$$\pi(s) = a$$

## What should the system learn?

**Policy Learning:**

Learn function $\pi$ *that directly* maps states to actions:

$$\pi(s) = a$$

***Q* Learning:**

Learn value function $Q$ that maps state, action pairs to values:

$$Q(s, a) = v$$

where $v$ is the prediction of future cumulative reward if system is in state $s$ and takes action $a$, *assuming the system follows the best policy thereafter.*

# What should the system learn?

**Policy Learning:**

Learn function $\pi$ *that directly* maps states to actions:

$$\pi(s) = a$$

***Q* Learning:**

Learn value function $Q$ that maps state, action pairs to values:

$$Q(s,a) = v$$

where $v$ is the prediction of future cumulative reward if system is in state $s$ and takes action $a$, *assuming the system follows the best policy thereafter.*

If you have the correct Q function, the best policy is:

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

## How to learn $Q$ ?
## (In theory)

Recall that we want to learn $Q(s,a)$ so that it gives predicted cumulative reward to system from starting in state $s$, taking action $a$, and following best policy thereafter.

Thus we have:

$$Q(s,a) = r(s,a) + \max_{a'} \left[ Q(\delta(s,a), a') \right]$$

## How to learn $Q$ ? (In theory)

Recall that we want to learn $Q(s,a)$ so that it gives predicted cumulative reward to system from starting in state $s$, taking action $a$, and following best policy thereafter.

Thus we have:

$$Q(s,a) = r(s,a) + \max_{a'}\left[Q(\delta(s,a),a')\right]$$

Can our learning system compute this expression directly?

## How to learn $Q$ ? (In theory)

Recall that we want to learn $Q(s,a)$ so that it gives predicted cumulative reward to system from starting in state $s$, taking action $a$, and following best policy thereafter.

Thus we have:

$$Q(s,a) = r(s,a) + \max_{a'}\left[Q(\delta(s,a),a')\right]$$

Can our learning system compute this expression directly?

**No! Instead it has to estimate *Q* via iterative exploration.**

# Estimating *Q* as look-up table

- Suppose there are *n* states and *m* possible actions.

- Then *Q* can be represented as an $n \times m$ matrix (or "look-up table):

| | $a_1$ | $a_2$ | $a_3$ | etc. |
|---|---|---|---|---|
| $s_1$ | | | | |
| $s_2$ | | | | |
| $s_3$ | | | | |
| $s_4$ | | | | |
| etc. | | | | |

- The *Q* table is typically initialized to all zeros.

## Temporal Difference Learning via Exploration

We want to iteratively estimate:

$$Q(s,a) = r(s,a) + \max_{a'} \left[ Q(\delta(s,a), a') \right]$$

# Temporal Difference Learning via Exploration

We want to iteratively estimate:

$$Q(s,a) = r(s,a) + \max_{a'}\left[Q(\delta(s,a),a')\right]$$

Suppose at time $t$ we are in state $s_t$, and have a current estimate for function $Q$.

## Temporal Difference Learning via Exploration

We want to iteratively estimate:

$$Q(s,a) = r(s,a) + \max_{a'}\left[Q(\delta(s,a),a')\right]$$

Suppose at time *t* we are in state $s_t$ , and have a current estimate for function *Q*.

We can take an action, $a_t$ , and observe what happens.

## Temporal Difference Learning via Exploration

When we are in state $s_t$ and take action $a_t$ :

## Temporal Difference Learning via Exploration

When we are in state $s_t$ and take action $a_t$ :

- We observe that we go into state $s_{t+1}$ and receive reward $r(s_t, a_t)$

# Temporal Difference Learning via Exploration

When we are in state $s_t$ and take action $a_t$ :

- We observe that we go into state $s_{t+1}$ and receive reward $r(s_t, a_t)$
- If our current estimate of $Q$ were perfect, then we would have:

$$Q(s_t, a_t) = r(s_t, a_t) + \max_{a'} \left[ Q(s_{t+1}, a') \right]$$

## Temporal Difference Learning via Exploration

When we are in state $s_t$ and take action $a_t$ :

- We observe that we go into state $s_{t+1}$ and receive reward $r(s_t, a_t)$
- If our current estimate of $Q$ were perfect, then we would have:

$$Q(s_t, a_t) = r(s_t, a_t) + \max_{a'} \left[ Q(s_{t+1}, a') \right]$$

- The "temporal difference" is

$$r(s_t, a_t) + \max_{a'} \left[ Q(s_{t+1}, a') \right] - Q(s_t, a_t)$$

## Temporal Difference Learning via Exploration

When we are in state $s_t$ and take action $a_t$ :

- We observe that we go into state $s_{t+1}$ and receive reward $r(s_t, a_t)$
- If our current estimate of $Q$ were perfect, then we would have:

$$Q(s_t, a_t) = r(s_t, a_t) + \max_{a'}\left[Q(s_{t+1}, a')\right]$$

- The "temporal difference" is

$$r(s_t, a_t) + \max_{a'}\left[Q(s_{t+1}, a')\right] - Q(s_t, a_t)$$

- Update $Q$ in order to reduce the temporal difference:

$$Q(s_t, a_t) \Leftarrow Q(s_t, a_t) + \eta\left(r(s_t, a_t) + \max_{a'}\left[Q(s_{t+1}, a')\right] - Q(s_t, a_t)\right)$$

## Discount factor

- More generally, we include a “discount factor”, $\gamma$ (between 0 and 1), that expresses how much we value immediate versus future rewards.

$$Q(s_t,a_t) \Leftarrow Q(s_t,a_t) + \eta\left( r(s_t,a_t) + \gamma \max_{a'}\left[Q(s_{t+1},a')\right] - Q(s_t,a_t)\right)$$

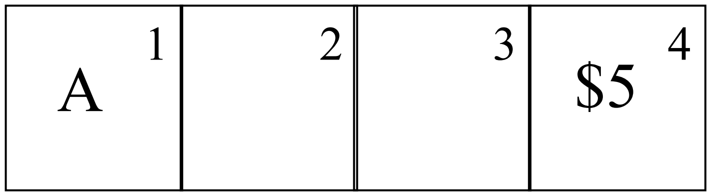
Low $\gamma$: We don’t care much about future rewards. Value actions that emphasize immediate reward.

High $\gamma$: We care a lot about future rewards. Value actions that emphasize future reward.

## How to learn $Q$ in practice

- Initialize $Q(s,a)$ to all zeros
- Initialize $s$
- Repeat forever (or as long as you have time for):
  - Select action $a$
  - Take action $a$ and receive reward $r$
  - Observe new state $s'$
  - Update $Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(\mathrm{s}, \mathrm{a}))$
  - Update s $\Leftarrow s'$

## Example



A is our agent, who takes an action at each time step.

Only action in square 1 is **F**orward.

Actions in squares 2 and 3 are (**F**orward, **B**ack)

Being in square 4 gives reward of $5

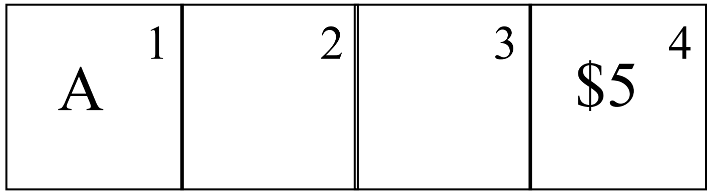
Only action in square 4 is **S**top

No other rewards or penalties.

Set $\gamma = .9$

Set $\eta = 1$

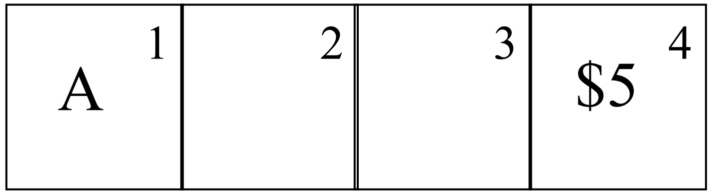
$$Q(s,a) \Leftarrow Q(s,a) + \eta\ (r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s},\ \mathrm{a}))$$



Episode 1
Current state *s* = 1

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

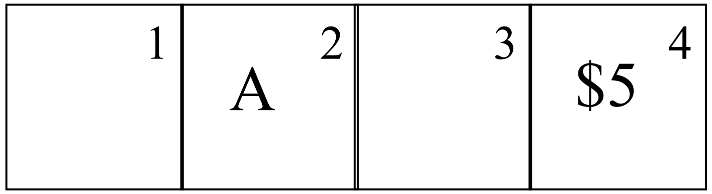
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s}, \mathrm{a}))$$



Episode 1
Current state *s* = 1
Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

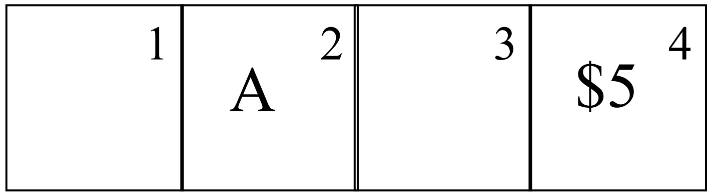
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 1
Current state *s* = 1
Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 1

Current state *s* = 1

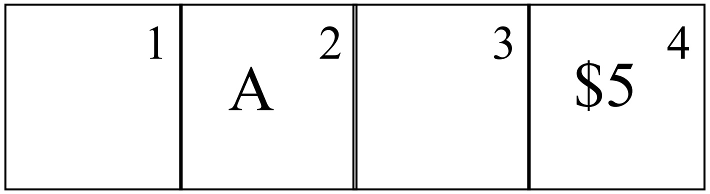
Action = F

*r* = 0

*s′* = 2

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

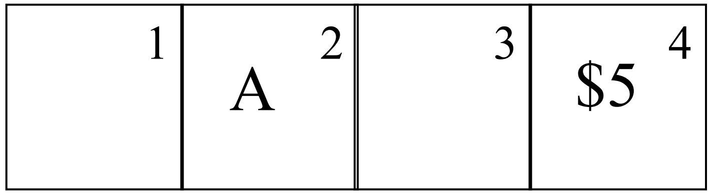


Episode 1
Current state *s* = 1
Action = F
*r* = 0
*s'* = 2

$$Q(1,F) = 0 + (0 + .9 \max_{a'} Q(2,a') - Q(1,F)) = 0$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

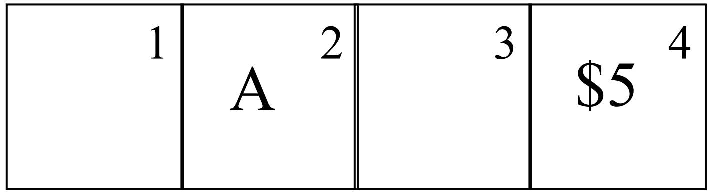
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 1
Current state $s$ = 2

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

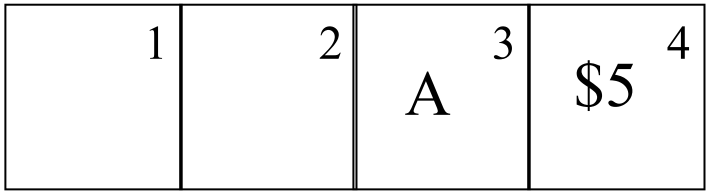
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q(s, a)})$$



Episode 1
Current state *s* = 2
Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

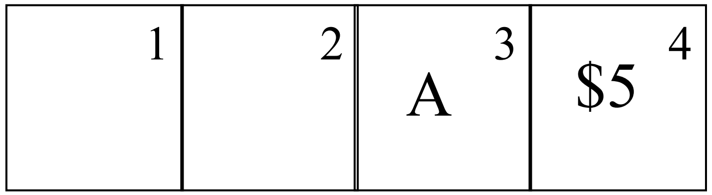
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 1
Current state *s* = 2
Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

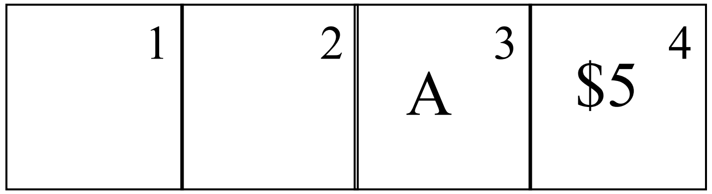
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



Episode 1
Current state *s* = 2
Action = F
*r* = 0
*s'* = 3

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\ (r + \gamma \max_{a'} Q(s',a') - \mathrm{Q(s, a)})$$

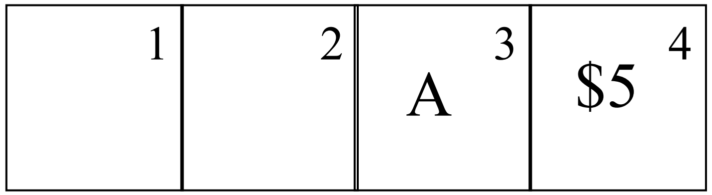


Episode 1
Current state *s* = 2
Action = F
*r* = 0
*s'* = 3

$$Q(2,F) = 0 + (0 + .9 \max_{a'} Q(3,a') - Q(2,F)) = 0$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

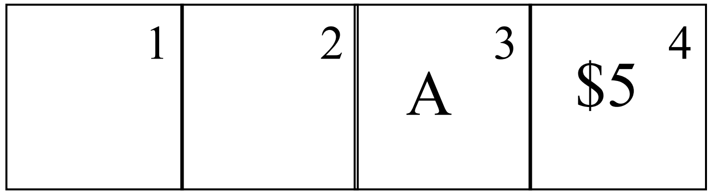
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 1
Current state *s* = 3

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

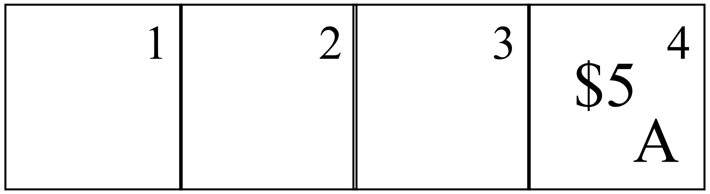
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 1
Current state *s* = 3
Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

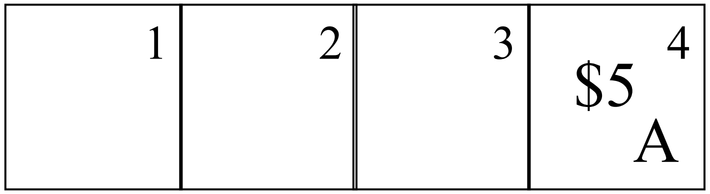
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



Episode 1
Current state *s* = 3
Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

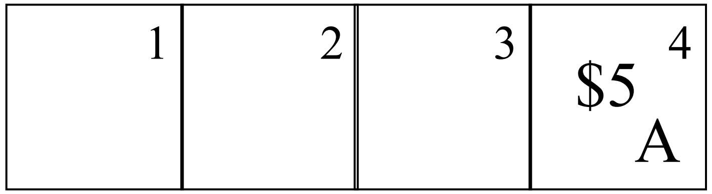
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 1
Current state *s* = 3
Action = F
*r* = $5
*s'* = 4

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$

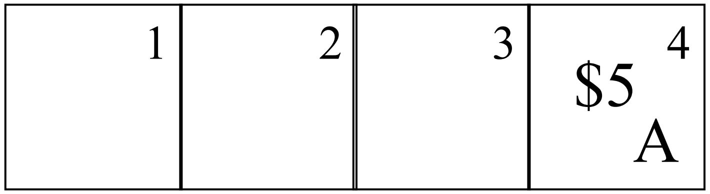


Episode 1
Current state *s* = 3
Action = F
*r* = $5
*s'* = 4

$$Q(3,F) = 0 + (\$5 + .9 \max_{a'} Q(4,a') - Q(3,F)) = \$5$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | 0 | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



Episode 1

Current state *s* = 3

Action = F

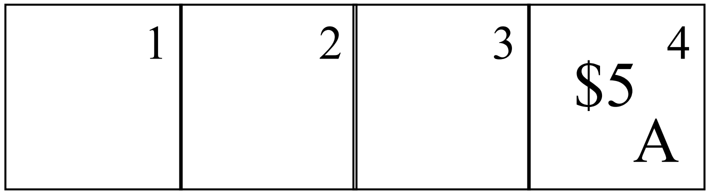
*r* = $5

*s'* = 4

$$Q(3,F) = 0 + (\$5 + .9 \max_{a'} Q(4,a') - Q(3,F)) = \$5$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

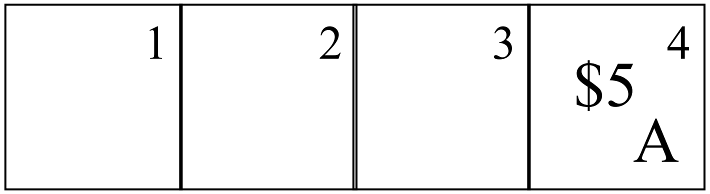


Episode 1

Current state *s* = 4

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



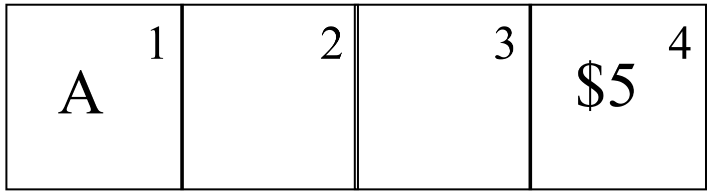
Episode 1

Current state *s* = 4

Action = Stop

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

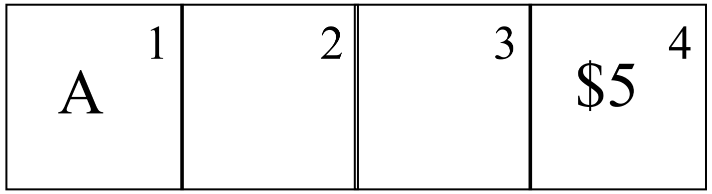
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 2
Current state *s* = 1

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



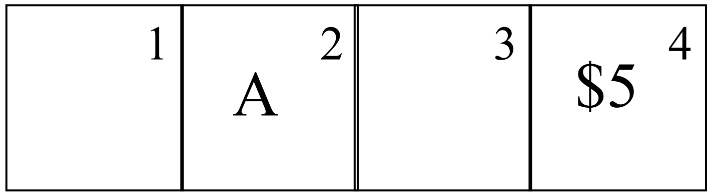
Episode 2

Current state *s* = 1

Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



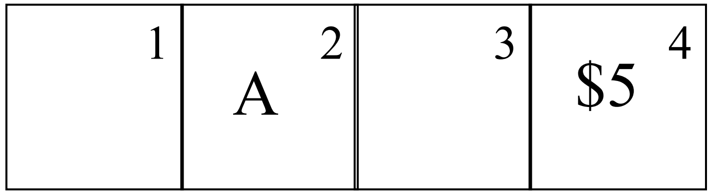
Episode 2

Current state *s* = 1

Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 2

Current state *s* = 1

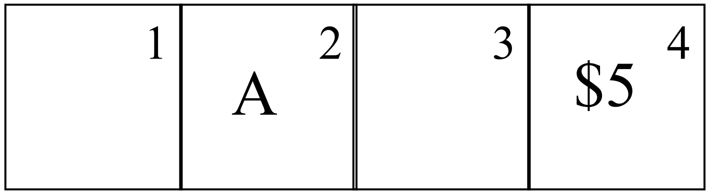
Action = F

*r* = 0

*s'* = 2

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 2

Current state *s* = 1

Action = F

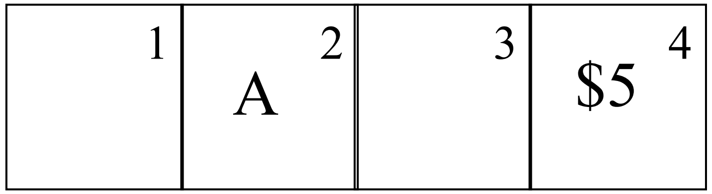
*r* = 0

*s’* = 2

$$Q(1,F) = 0 + (0 + .9 \max_{a'} Q(2,a') - Q(1,F)) = 0$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

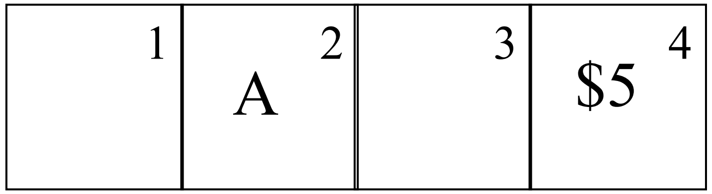
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



Episode 2
Current state *s* = 2

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



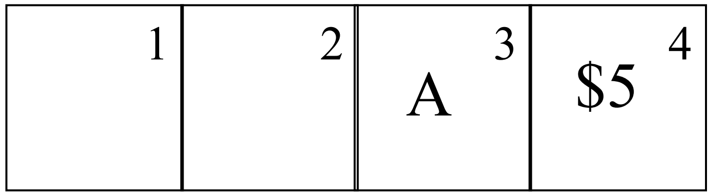
Episode 2

Current state *s* = 2

Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

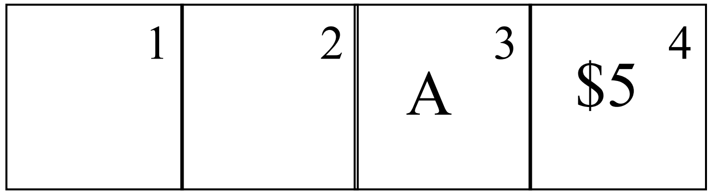
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 2
Current state *s* = 2
Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 2

Current state *s* = 2

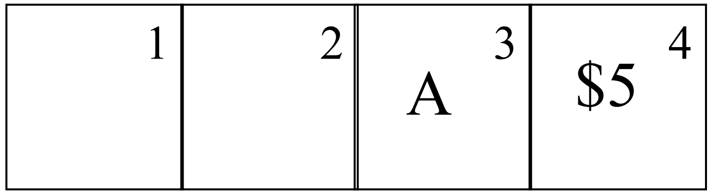
Action = F

*r* = 0

*s'* = 3

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



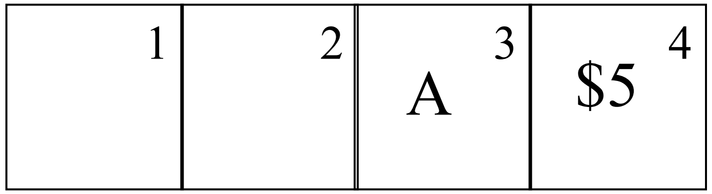
Episode 2
Current state *s* = 2
Action = F
*r* = 0
*s'* = 3

$$Q(2,F) = 0 + (0 + .9 \max_{a'} Q(3,a') - Q(2,F))$$

$$= 0 + 0 + (.9)(\$5) = \$4.50$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | 0 | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

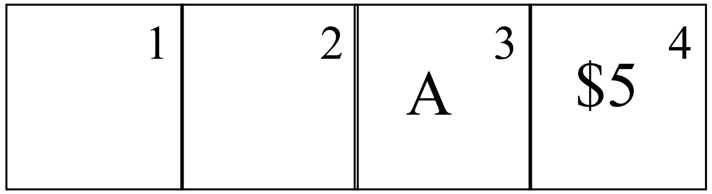


Episode 2
Current state $s$ = 2
Action = F
$r$ = 0
$s'$ = 3

$$Q(2,F) = 0 + (0 + .9 \max_{a'} Q(3,a') - Q(2,F))$$
$$= 0 + 0 + (.9)(\$5) = \$4.50$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **\$4.50** | 0 | X |
| 3 | **\$5** | 0 | X |
| 4 | X | X | 0 |

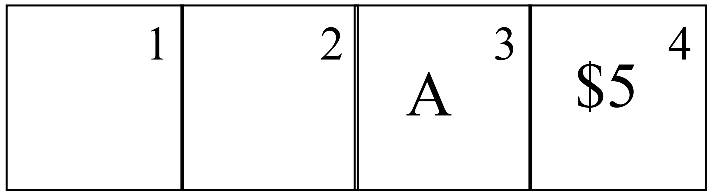
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



Episode 2
Current state *s* = 3

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\, (r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



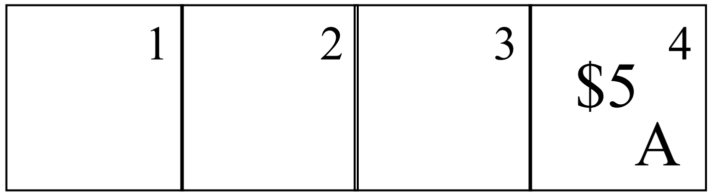
Episode 2

Current state *s* = 3

Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

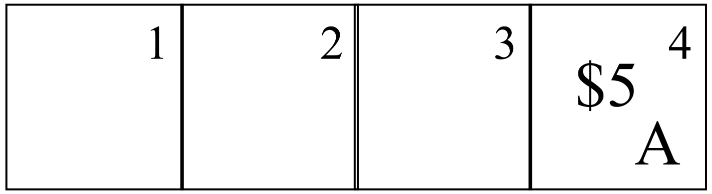
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 2
Current state *s* = 3
Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

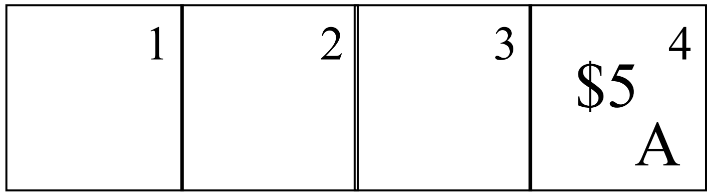
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 2
Current state *s* = 3
Action = F
*r* = \$5
*s'* = 4

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **\$4.50** | 0 | X |
| 3 | **\$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

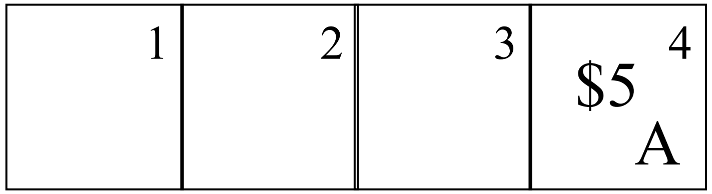


Episode 2
Current state *s* = 3
Action = F
*r* = $5
*s'* = 4

$$Q(3,F) = \$5 + (\$5 + .9 \max_{a'} Q(4,a') - Q(3,F))$$
$$= 0 + \$5 + 0 = \$5$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

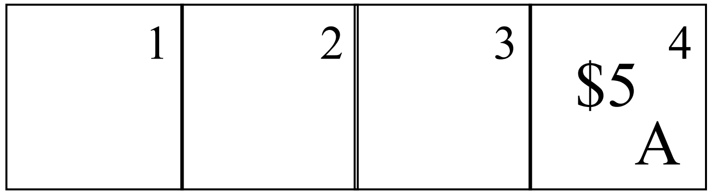
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 2
Current state *s* = 4

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

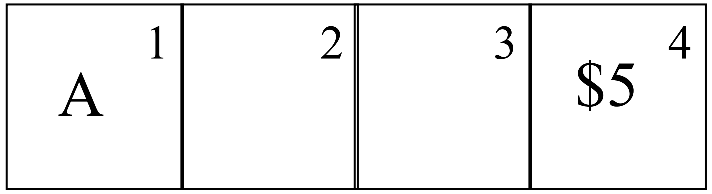
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 2
Current state *s* = 4
Action = Stop

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

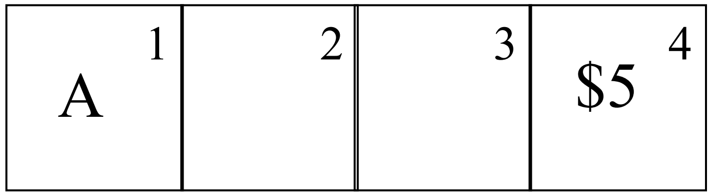
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3
Current state *s* = 1

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



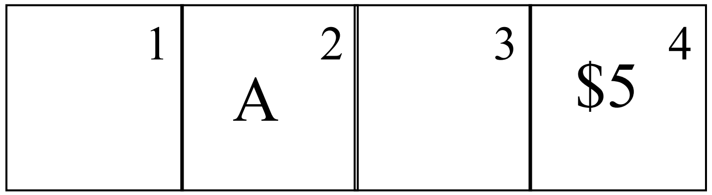
Episode 3

Current state *s* = 1

Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

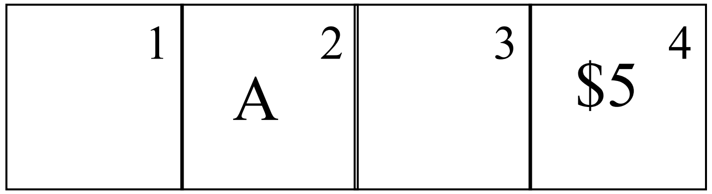
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3
Current state *s* = 1
Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



Episode 3

Current state *s* = 1

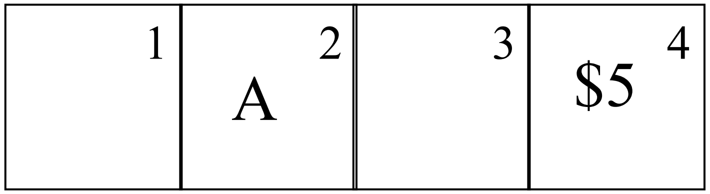
Action = F

*r* = 0

*s'* = 2

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3

Current state *s* = 1

Action = F

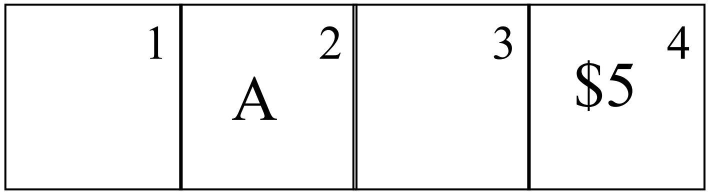
*r* = 0

*s'* = 2

$$Q(1,F) = 0 + (0 + .9 \max_{a'} Q(2,a') - Q(1,F))$$
$$= 0 + 0 + (.9)(\$4.50) - 0 = \$4.05$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | 0 | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3

Current state *s* = 1

Action = F

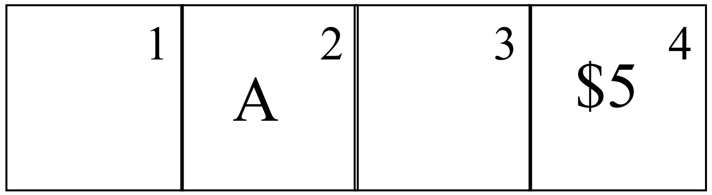
*r* = 0

*s'* = 2

$$Q(1,F) = 0 + (0 + .9 \max_{a'} Q(2,a') - Q(1,F))$$
$$= 0 + 0 + (.9)(\$4.50) - 0 = \$4.05$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

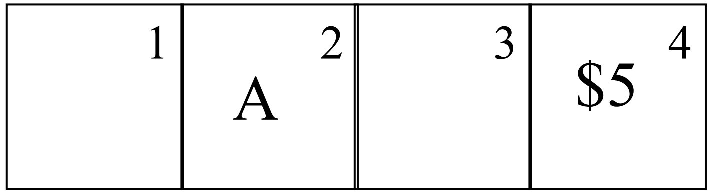
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3
Current state *s* = 2

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

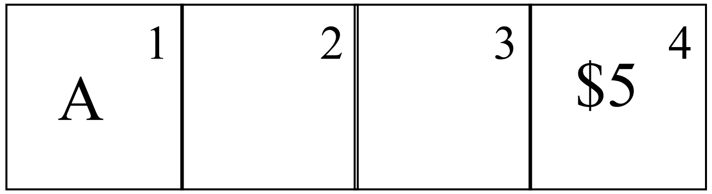
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



Episode 3
Current state *s* = 2
Action = B

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

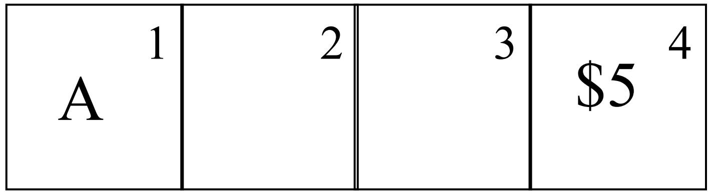
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3
Current state *s* = 2
Action = B

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3

Current state *s* = 2

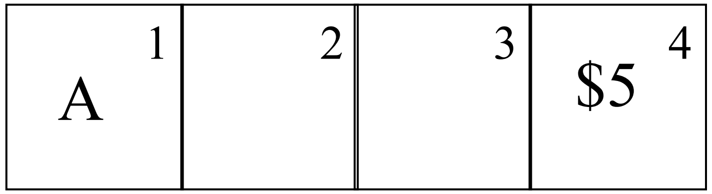
Action = B

*r* = 0

*s'* = 1

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3

Current state *s* = 2

Action = B

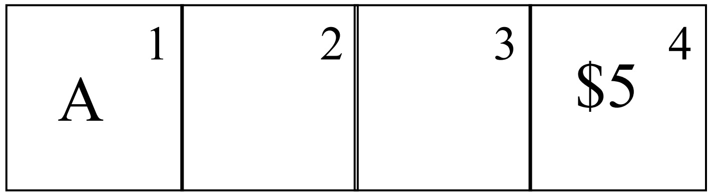
*r* = 0

*s'* = 1

$$Q(2,B) = 0 + (0 + .9 \max_{a'} Q(1,a') - Q(2,B))$$
$$= 0 + 0 + (.9)(\$4.05) - 0 = \$3.65$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | 0 | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3

Current state *s* = 2

Action = B

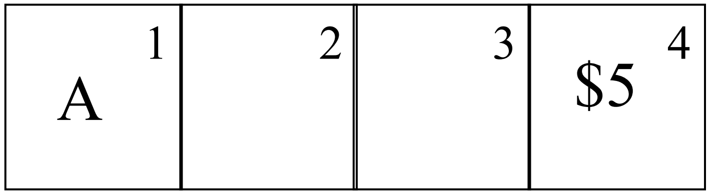
*r* = 0

*s'* = 1

$$Q(2,B) = 0 + (0 + .9\max_{a'} Q(1,a') - Q(2,B))$$
$$= 0 + 0 + (.9)(\$4.05) - 0 = \$3.65$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | **$3.65** | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

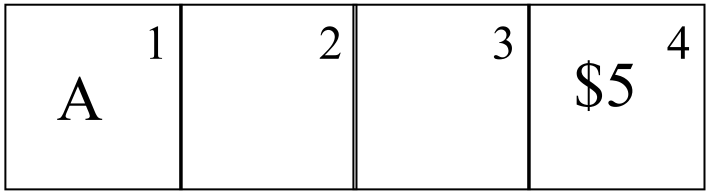
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3
Current state $s$ = 1

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | **$3.65** | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

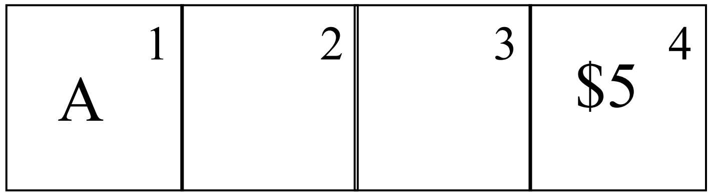
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



Episode 3
Current state *s* = 1
Action = F

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | **$3.65** | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



Episode 3

Current state *s* = 1

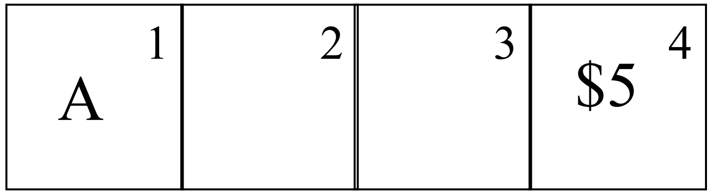
Action = F

*r* = 0

*s'* = 2

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | **$3.65** | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



Episode 3

Current state *s* = 1

Action = F

*r* = 0

*s'* = 2

$$Q(1,F) = \$4.05 + (0 + .9 \max_{a'} Q(2,a') - Q(1,F))$$

$$\$4.05 + 0 + (.9)(\$4.50) - \$4.05 = \$4.05$$

| Q(s,a) | Forward | Back | Stop |
|---|---|---|---|
| 1 | **$4.05** | X | X |
| 2 | **$4.50** | **$3.65** | X |
| 3 | **$5** | 0 | X |
| 4 | X | X | 0 |

## How to choose actions?

- Naïve strategy: at each time step, choose action that maximizes current $Q(s,a)$

## How to choose actions?

- Naïve strategy: at each time step, choose action that maximizes current $Q(s,a)$

- This exploits current $Q$ but doesn’t further explore the state-action space (in case $Q$ is way off)

## How to choose actions?

- Naïve strategy: at each time step, choose action that maximizes current $Q(s,a)$

- This exploits current $Q$ but doesn't further explore the state-action space (in case $Q$ is way off)

- Common in Q learning to use "epsilon greedy" approach:
  - With probability $(1 - \varepsilon)$ choose action that maximizes current $Q(s,a)$

  - With probability $\varepsilon$ choose random action

## How to choose actions?

- Naïve strategy: at each time step, choose action that maximizes current $Q(s,a)$

- This exploits current $Q$ but doesn't further explore the state-action space (in case $Q$ is way off)

- Common in Q learning to use "epsilon greedy" approach:
  - With probability $(1-\varepsilon)$ choose action that maximizes current $Q(s,a)$

  - With probability $\varepsilon$ choose random action

- Can start with high $\varepsilon$, and decrease it over the run

## Q Learning for Robby the Robot

# **Robby the Robot** can learn via reinforcement learning

Sensors (**State**):

H(ere), N,S,E,W

***"policy" = "strategy"***

**Actions**:

Move N

Move S

Move E

Move W

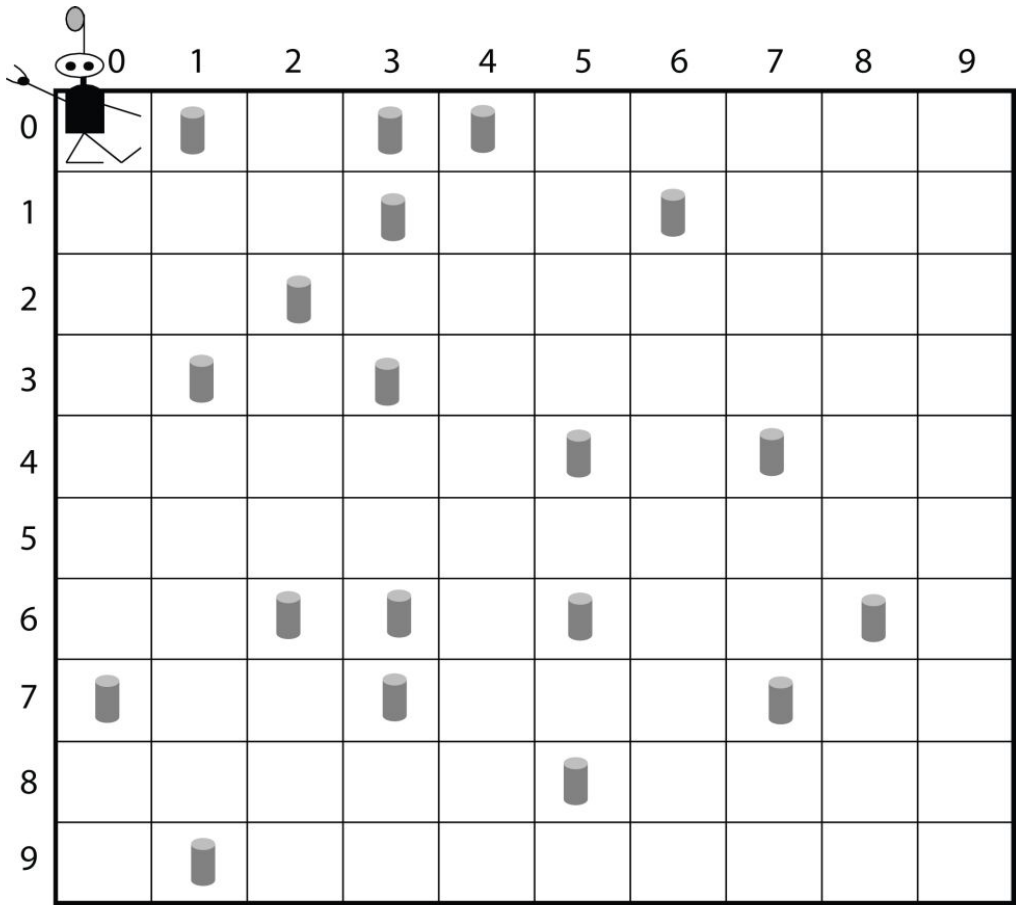
Pick up can

**Rewards**:

Picks up can: 10

Pick up can on empty site: -1

Crashes into wall: -5



## Representation of $Q(s, a)$

- Note that in all of the above discussion, $Q(s, a)$ was assumed to be a look-up table, with a distinct table entry for each distinct $(s,a)$ pair.

- More commonly, $Q(s, a)$ is represented as a function (e.g., a neural network), and the function is estimated (e.g., through back-propagation).

## Example:  Learning to play backgammon

## Complexity of Backgammon

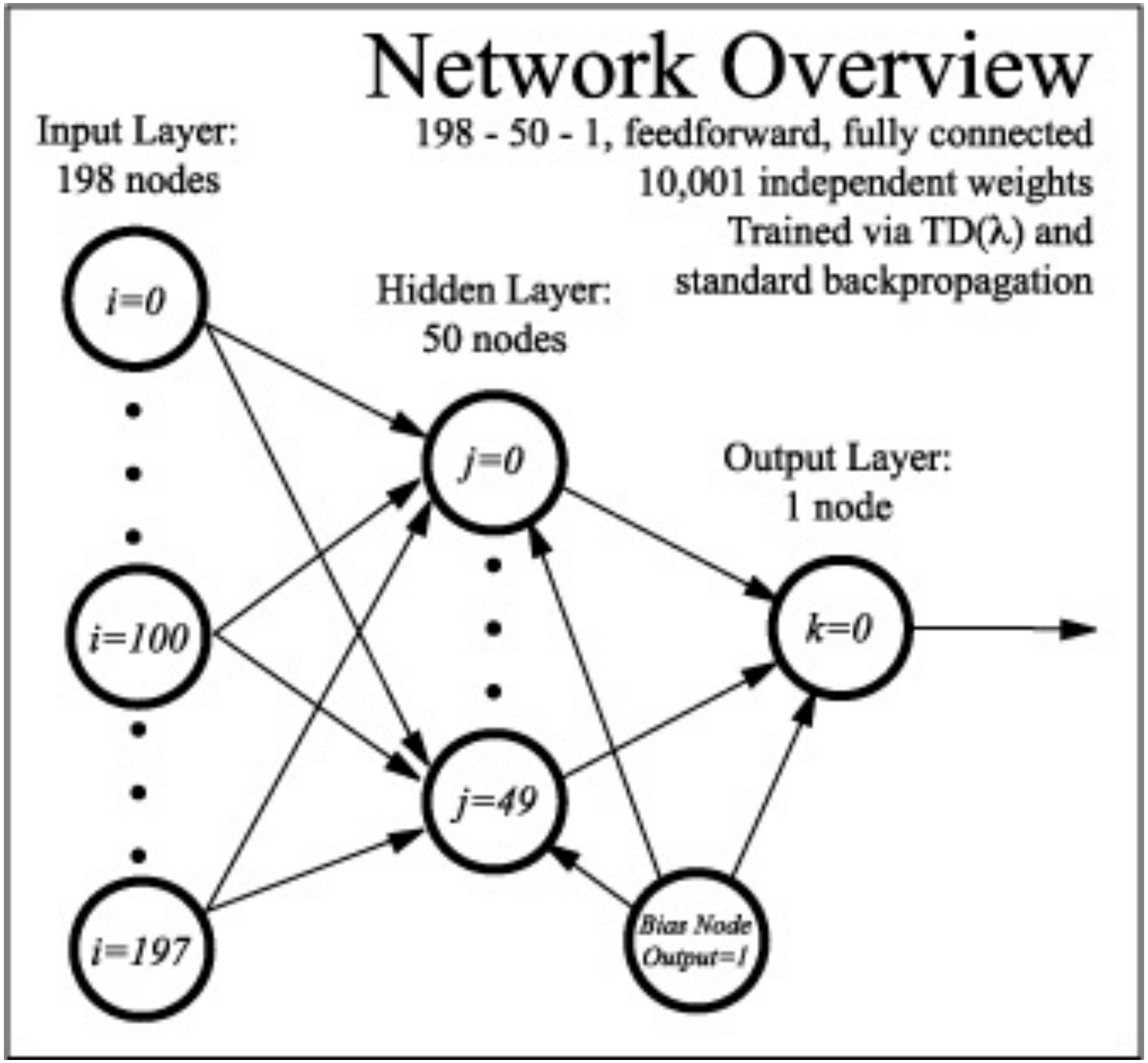
- Over $10^{20}$ possible states.
- At each ply, 21 dice combinations, with average of about 20 legal moves per dice combination. Result is branching ratio of several hundred per ply.
- Chess has branching ratio of about 30-40 per ply.
- Brute-force look-ahead search is not practical!

## Neurogammon (Tesauro, 1989)

- Used supervised learning approach:  multilayer NN trained by back-propagation on data base of recorded expert games.
- Input:  raw board information (number of pieces at each location), and a few hand-crafted features that encoded important expert concepts.
- Neurogammon achieved strong intermediate level of play.
- Won backgammon championship at 1989 International Computer Olympiad.    But not close to beating best human players.

## TD-Gammon
## (G. Tesauro, 1994)

- Program had two main parts:
  - **Move Generator**: Program that generates all legal moves from current board configuration.
  - **Predictor network**: multi-layer NN that predicts $Q(s,a)$: probability of winning the game from the current board configuration.
- Predictor network scores all legal moves. Highest scoring move is chosen.
- **Rewards:** Zero for all time steps except those on which game is won or lost.


Network Overview
198 - 50 - 1, feedforward, fully connected
10,001 independent weights
Trained via TD(λ) and
standard backpropagation
Input Layer:
198 nodes
Hidden Layer:
50 nodes
Output Layer:
1 node
i=0
i=100
i=197
j=0
j=49
k=0
Bias Node
Output=1

- Input: 198 units
  - 24 positions, 8 input units for each position (192 input units)
    - First 4 input units of each group of 8 represent # white pieces at that position,
    - Second 4 represent # black units at that position
  - Two inputs represent who is moving (white or black)
  - Two inputs represent pieces on the bar
  - Two inputs represent number of pieces borne off by each player.

- 50 hidden units
- 1 output unit (activation represents probability that white will win from given board configuration)

Program plays against itself.

On each turn:

- Use network to evaluate all possible moves from current board configuration. Choose the move with the highest (lowest as black) evaluation. This produces a new board configuration.

- If this is end of game, run back-propagation, with target output activation of 1 or 0 depending on whether white won or lost.

- Else evaluate new board configuration with neural network. Calculate difference between current evaluation and previous evaluation.

- Run back-propagation, using the current evaluation as target output, and the board position previous to the current move as the input.

| Program | Training Games | Opponents | Results |
|---|---|---|---|
| TDG 1.0 | 300,000 | Robertie, Davis, Magriel | -13 pts/51 games (-0.25 ppg) |
| TDG 2.0 | 800,000 | Goulding, Woolsey, Snellings, Russell, Sylvester | -7 pts/38 games (-0.18 ppg) |
| TDG 2.1 | 1,500,000 | Robertie | -1 pt/40 games (-0.02 ppg) |

**Table 1.** Results of testing TD-Gammon in play against world-class human opponents. Version 1.0 used 1-play search for move selection; versions 2.0 and 2.1 used 2-ply search. Version 2.0 had 40 hidden units; versions 1.0 and 2.1 had 80 hidden units.

- From Sutton & Barto, *Reinforcement Learning: An Introduction*:

  "After playing about 300,000 games against itself, TD-Gammon 0.0 as described above learned to play approximately as well as the best previous backgammon computer programs."

  "TD-Gammon 3.0 appears to be at, or very near, the playing strength of the best human players in the world. It may already be the world champion. These programs have already changed the way the best human players play the game. For example, TD-Gammon learned to play certain opening positions differently than was the convention among the best human players. Based on TD-Gammon's success and further analysis, the best human players now play these positions as TD-Gammon does (Tesauro, 1995)."

## Deep Reinforcement Learning

## Learning to play Atari video games with Deep Reinforcement Learning

Mnih et al., DeepMind Technologies, 2013

- **Deep learning:**
  - Requires large amount of hand-labeled data
  - Assumes data samples are iid, with stationary distribution
- **Reinforcement learning:**
  - Must learn from sparse, noisy, delayed reward function
  - “Samples” are not independent
  - Data distribution can change as system learns “online”.

# Learning to play Atari video games with Deep Reinforcement Learning

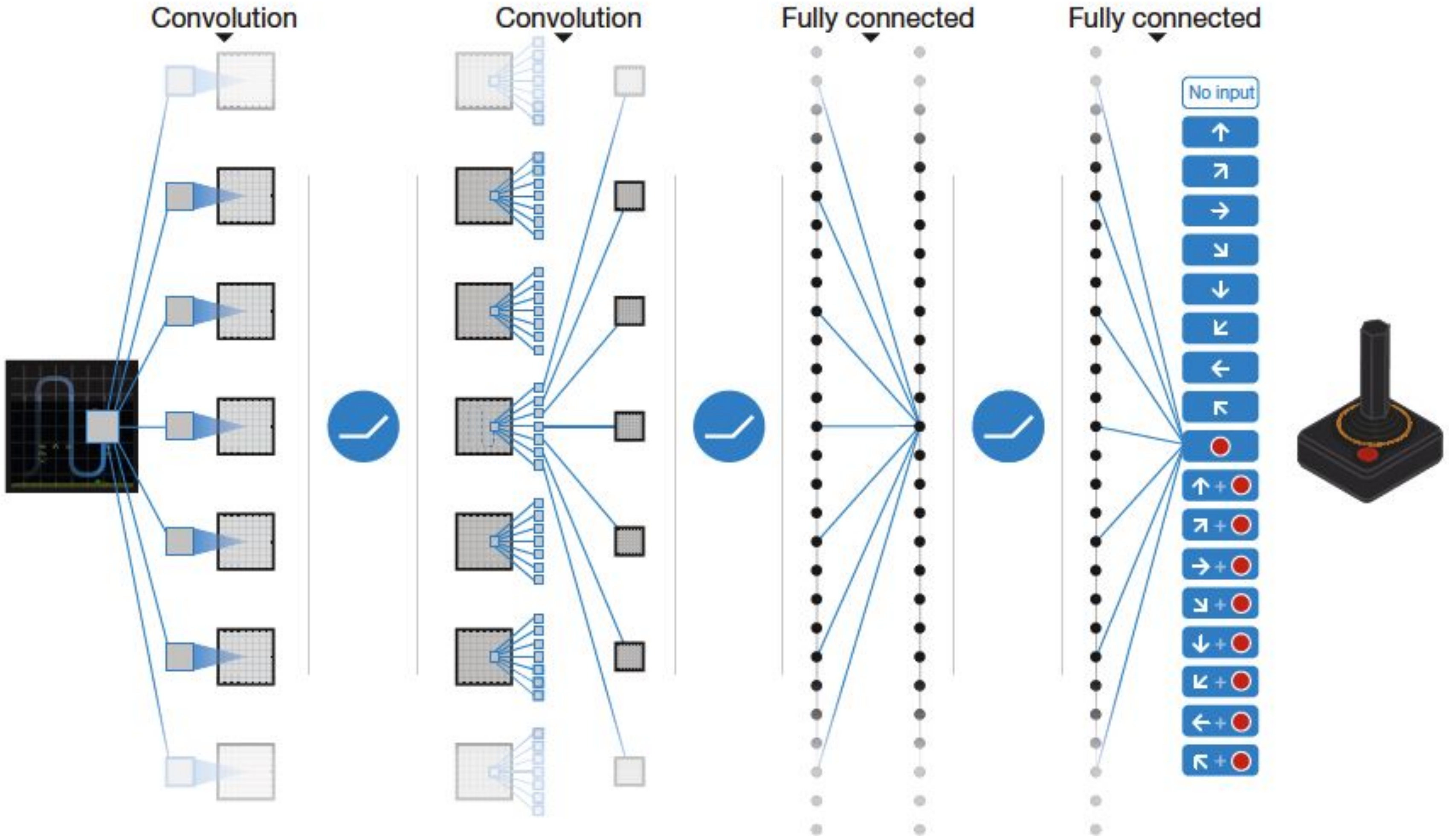
Mnih et al., DeepMind Technologies, 2013

- https://www.youtube.com/watch?v=Up-a5x3coC0

- Uses convolutional neural network:
  - Input is raw pixels of video frames (~ the “state”)

- Output is estimated $Q(s,a)$ for each possible action

- Their system learns to play Atari 2600 games.
- 210x160 RGB video at 60 Hz.
- Designed to be difficult for human players
- “Our goal is to create a single neural network agent that is able to successfully learn to play as many of the games as  possible.”
- No game-specific info provided.  No hand-designed visual features.
- Learns exclusively from video input, the reward, and terminal signals. Network architecture and hyperparameters kept constant across all games.


Convolution
Convolution
Fully connected
Fully connected
No input

- **CNN:** Input is set of 4 most recent video frames, output vector is $Q$-value of each possible action, given the input state

- At each time step $t$, agent selects action $a_t$ from set of legal actions $A = \{1,\ldots, K\}$.

- Action $a_t$ is passed to emulator.

- Game state and current score are updated.

- Agent observes image $x_\mathrm{t}$ from emulator (vector of raw pixels representing current screen)

- Agent receives reward $r_t$ representing change in game score. (Usually 0.)

**Loss function:**

$$L_t(\theta_t) = \frac{1}{2}\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a_t)\right)^2$$

**Three possible sources of training instability / non-convergence:**

1. Temporally adjacent "experiences" ($s$, $a$, $r$, $s'$) are clearly not independent or identically distributed.

2. Small updates to $Q$ can significantly change the policy

3. Correlations between the action values $Q$ and "target" values

**Proposed methods to address these:**

1. Use "experience replay"

2. Use two networks: one for action values, $Q(s, a ; \theta)$, and one for targets, $Q(s, a ; \theta-)$ , and only update the target network periodically.

## Experience Replay

During training, to alleviate problem of correlated data and non-stationary distributions, use **“experience reply” mechanism**, which randomly samples previous transitions, and “thereby smooths the training distribution over many past behaviors.”

“During learning, we apply Q-learning updates, on samples (or minibatches) of experience ($s$,$a$,$r$,$s$’) drawn uniformly at random from the pool of stored samples.”

## Separating action-value and target networks

**Use two networks:**

$Q(s, a; \theta)$ is the network that estimates the current estimated value of taking action $a$ in state $s$

$Q\ (s', a'; \theta-)$ is the network used for estimating the target:

$$r + \gamma \max_{a'} Q(s', a')$$

**New loss function:**

$$L_t(\boldsymbol{\theta}_t) = \left( r + \gamma \max_{a'} Q(s', a' \mid \boldsymbol{\theta}_t^-) - Q(s, a \mid \boldsymbol{\theta}_t) \right)^2$$

## Updating networks

Every time step, $Q(s, a; \theta_t)$ is updated using the target computed using $Q(s', a'; \theta_t-)$.

(Minimize loss function via stochastic gradient descent on $\theta_t$ (weights).)

Every $C$ time steps, $\theta-$ is replaced by $\theta$.

output of CNN

**Algorithm 1** Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        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 $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta^{-}) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for
```

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]. \quad (3)$$

## Experiments

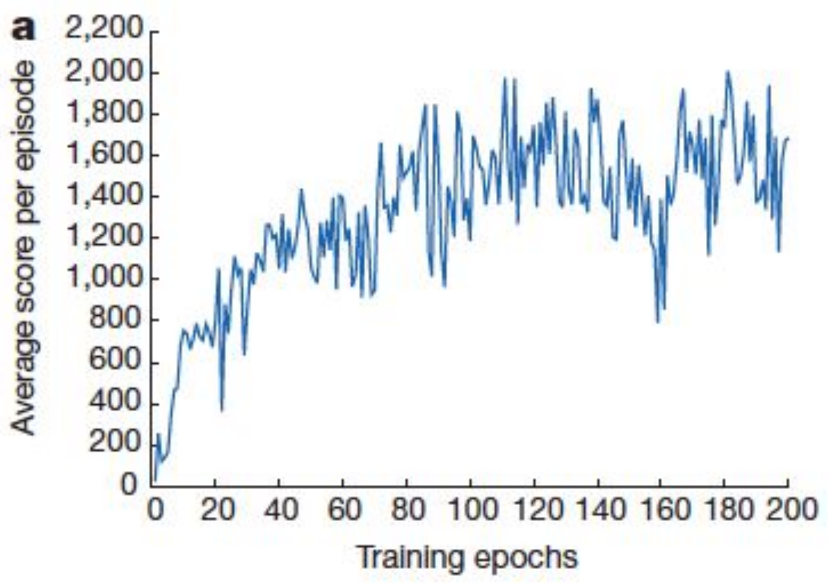
- Used same CNN architecture, learning algorithm, and parameters across a suite of Atari games.

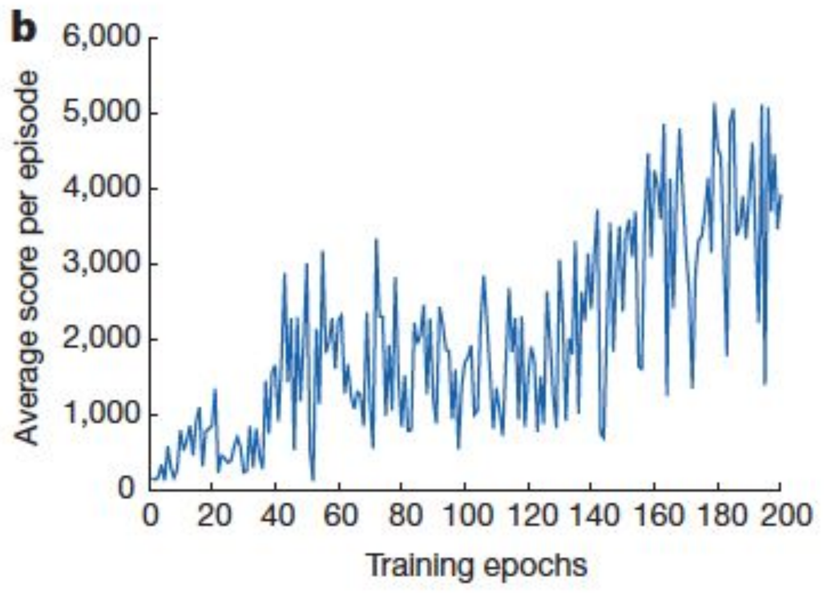
## Video

https://www.youtube.com/watch?v=Ih8EfvOzBOY

## More detailed results

**Space Invaders** **Seaquest**





Epsilon-greedy action section, with epsilon = 0.05

**Extended Data Table 4 | Comparison of DQN performance with linear function approximator**

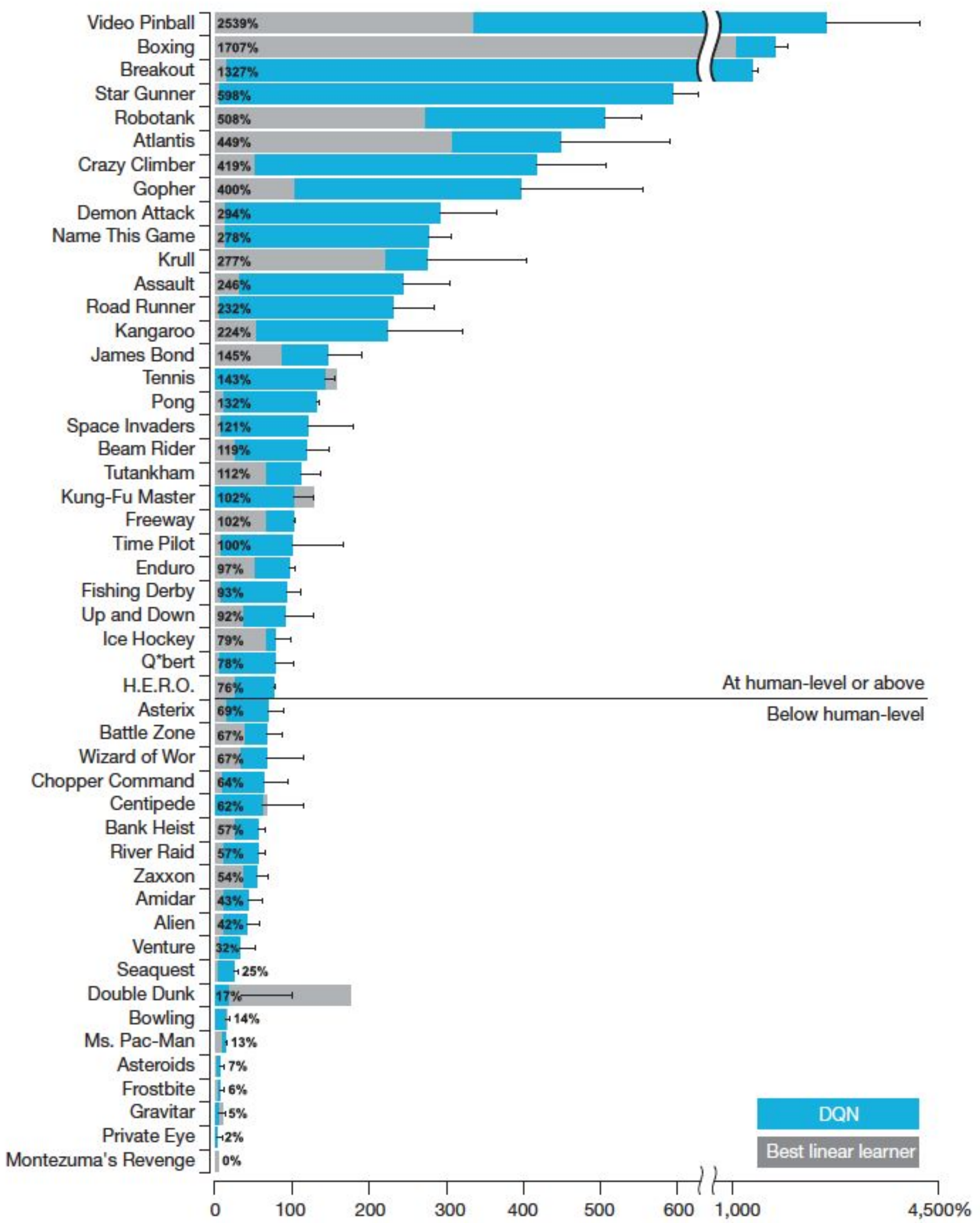
| Game | DQN | Linear |
|---|---|---|
| Breakout | 316.8 | 3.00 |
| Enduro | 1006.3 | 62.0 |
| River Raid | 7446.6 | 2346.9 |
| Seaquest | 2894.4 | 656.9 |
| Space Invaders | 1088.9 | 301.3 |

The performance of the DQN agent is compared with the performance of a linear function approximator on the 5 validation games (that is, where a single linear layer was used instead of the convolutional network, in combination with replay and separate target network). Agents were trained for 10 million frames using standard hyperparameters, and three different learning rates. Each agent was evaluated every 250,000 training frames for 135,000 validation frames and the highest average episode score is reported. Note that these evaluation episodes were not truncated at 5 min leading to higher scores on Enduro than the ones reported in Extended Data Table 2. Note also that the number of training frames was shorter (10 million frames) as compared to the main results presented in Extended Data Table 2 (50 million frames).

"The performance of DQN is normalized with respect to a professional human games tester (that is, 100% level) and random play (that is, 0% level).

Note that the normalized performance of DQN, expressed as a percentage, is calculated as:

100 * (DQN score − random play score) / (human score − random play score)."


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
At human-level or above
Below human-level
DQN
Best linear learner