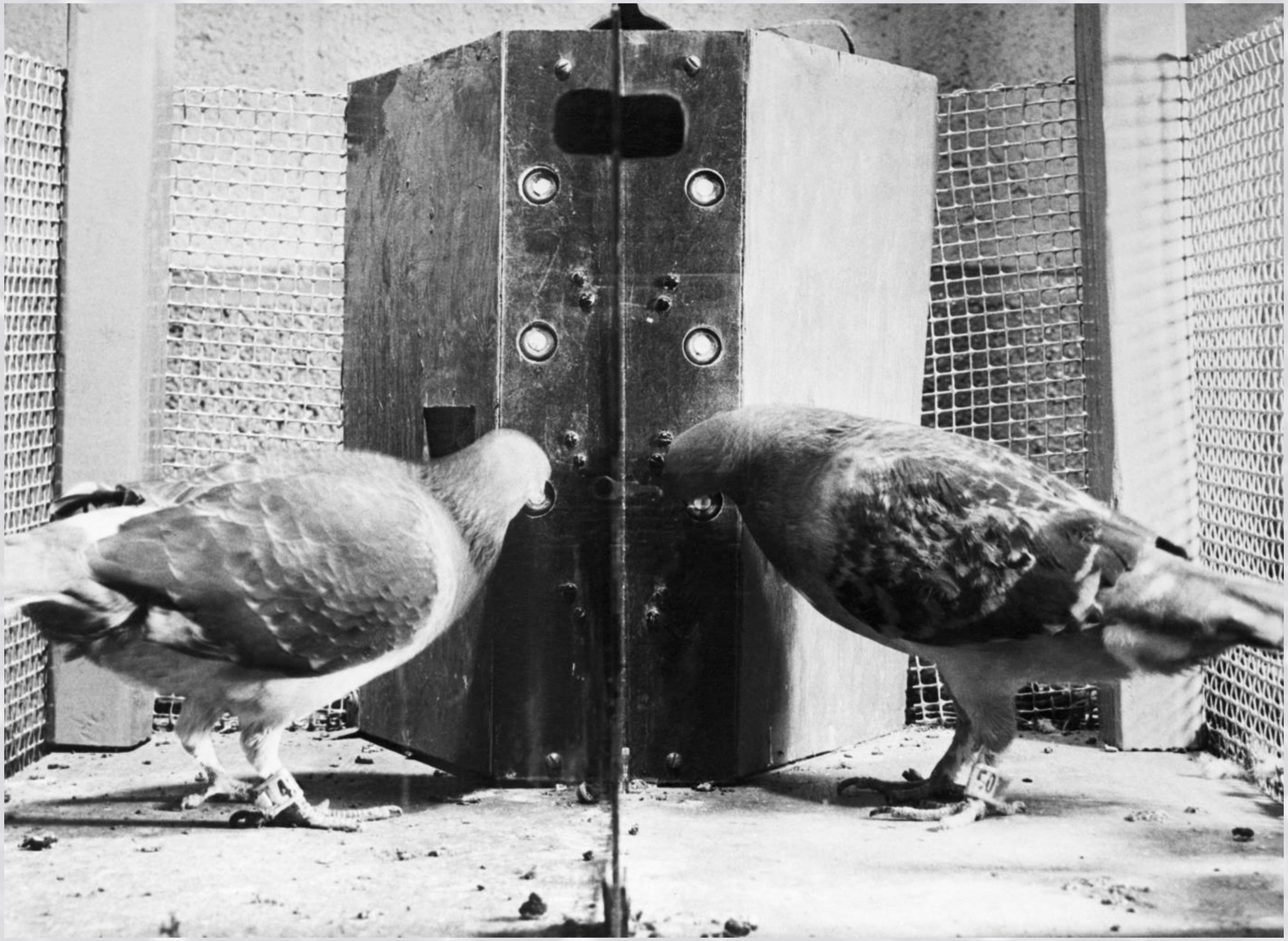
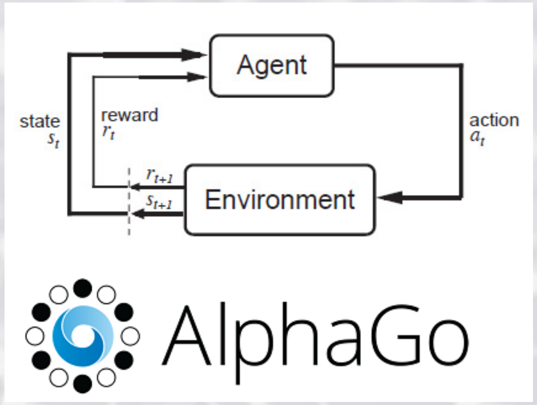
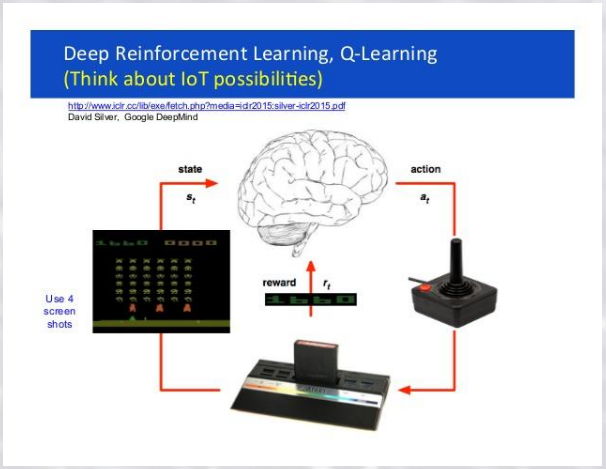
Artificial Intelligence: Reinforcement Learning



It’s an exciting time for RL!

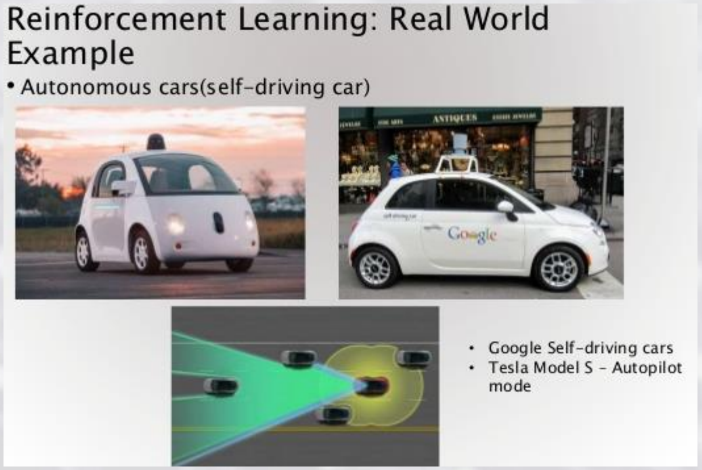




Deep Reinforcement Learning, Q-Learning
(Think about IoT possibilities)

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf
David Silver, Google DeepMind

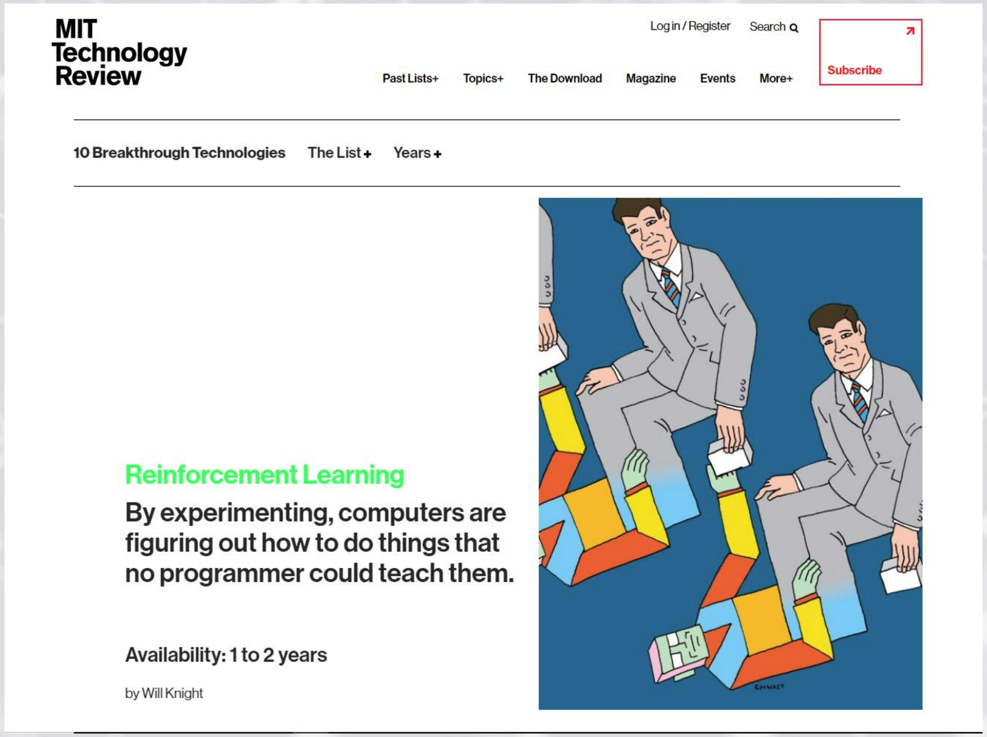
state
s_t
action
a_t
reward
r_t
Use 4 screen shots



Reinforcement Learning: Real World Example

- Autonomous cars(self-driving car)

- Google Self-driving cars
- Tesla Model S – Autopilot mode



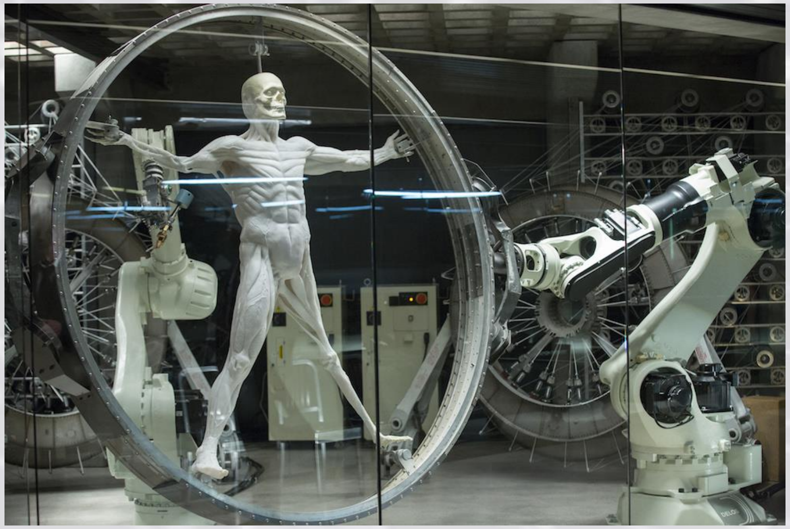
MIT Technology Review

Reinforcement Learning

By experimenting, computers are figuring out how to do things that no programmer could teach them.

Availability: 1 to 2 years

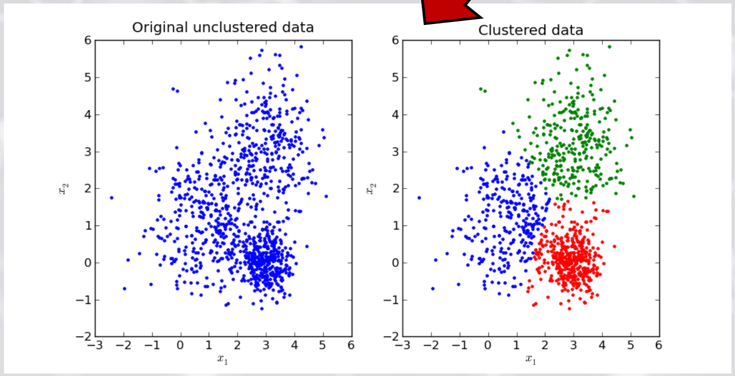
by Will Knight

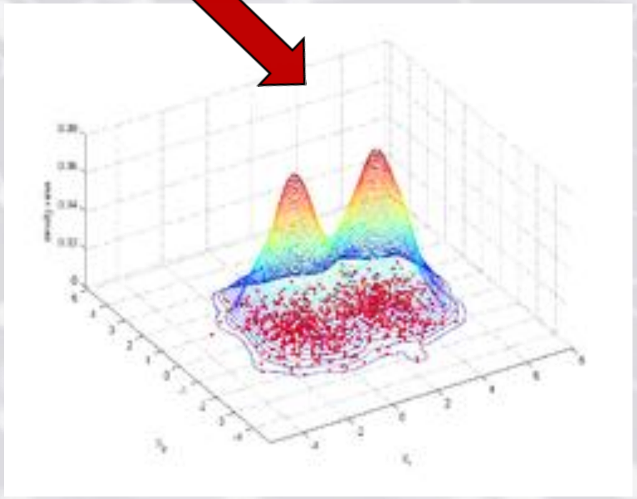


SkyNet

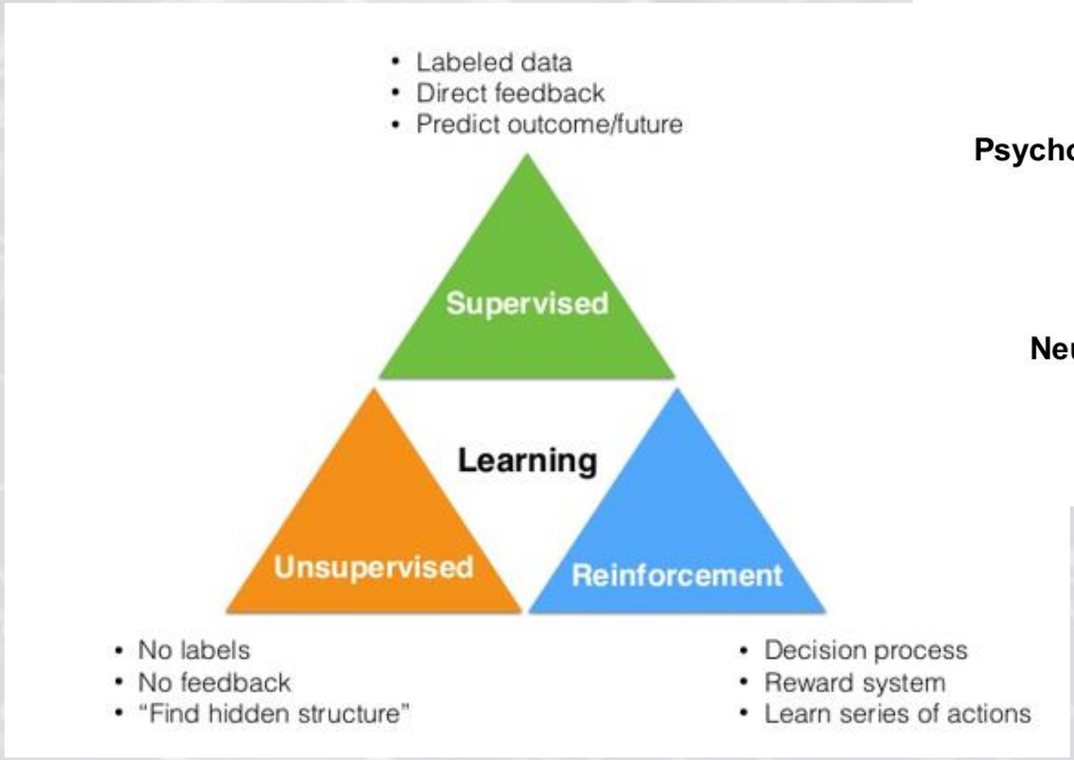
Recall: Unsupervised Learning

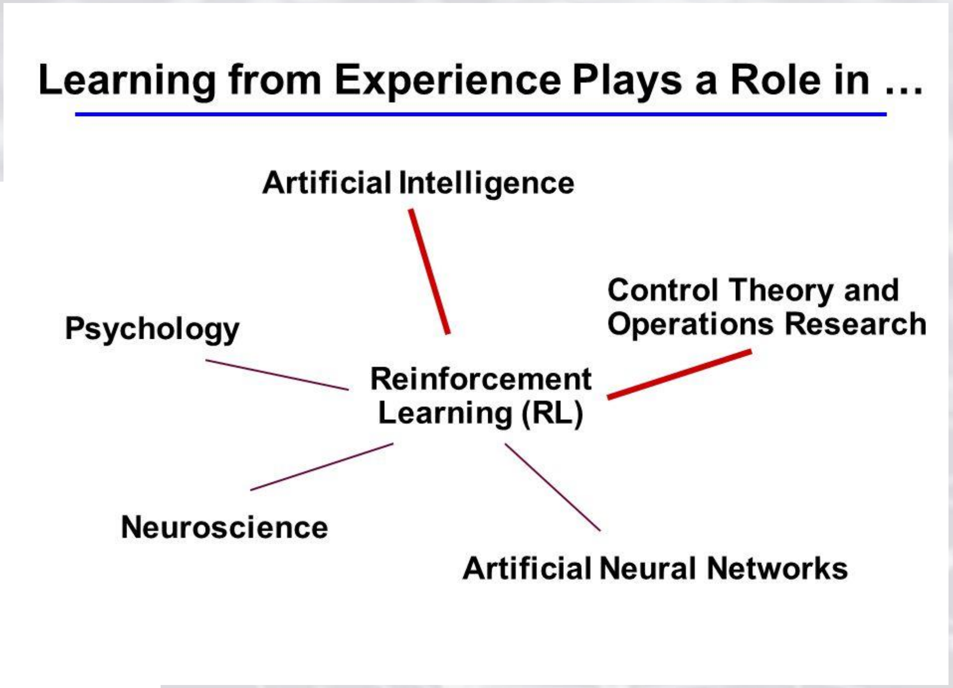
- We are given data, *x*, without labels.
- Goal: Learn underlying structure, patterns in the data.
- Examples: Clustering (k-means), density estimation.





Learning Models

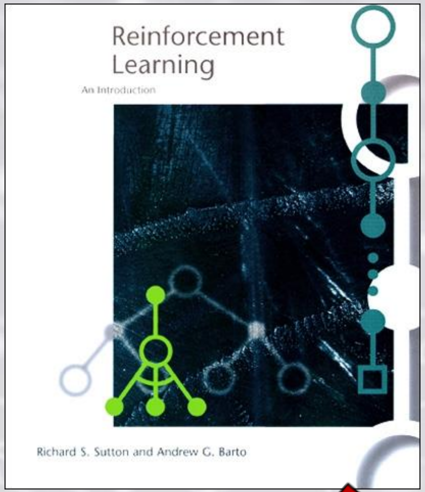




Learning from Experience Plays a Role in …
Artificial Intelligence
Control Theory and Operations Research
Psychology
Reinforcement Learning (RL)
Neuroscience
Artificial Neural Networks

Reinforcement Learning Overview

- Many AI and ML tasks focus 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)



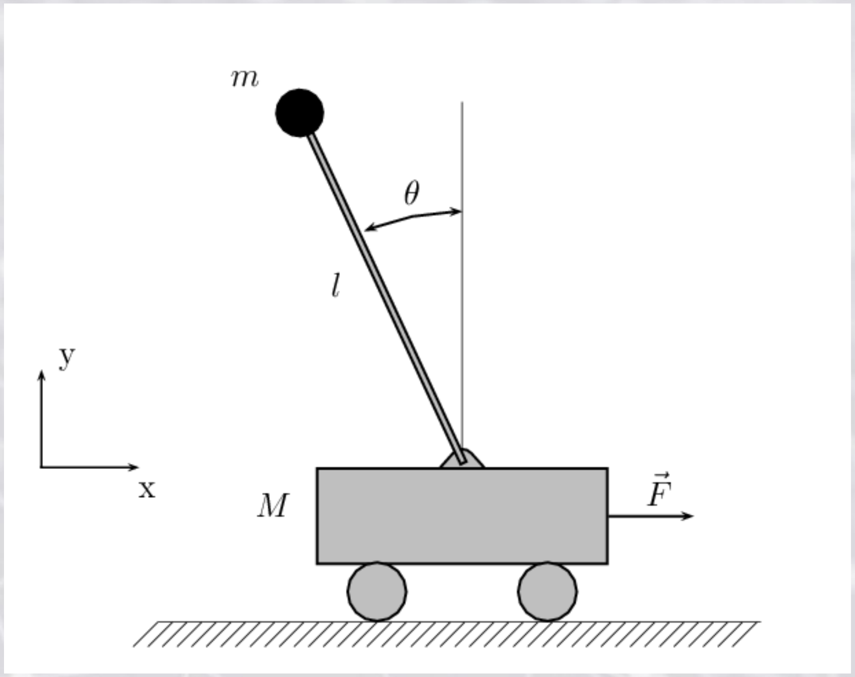
Recommended Text on RL: Sutton/Barto

- 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)
- Robot arm control (juggling)
- Robot Locomotion
- Robot navigation
- Elevator dispatching
- Power systems stability control
- Job-shop scheduling
- Air traffic control

Cart-Pole Problem



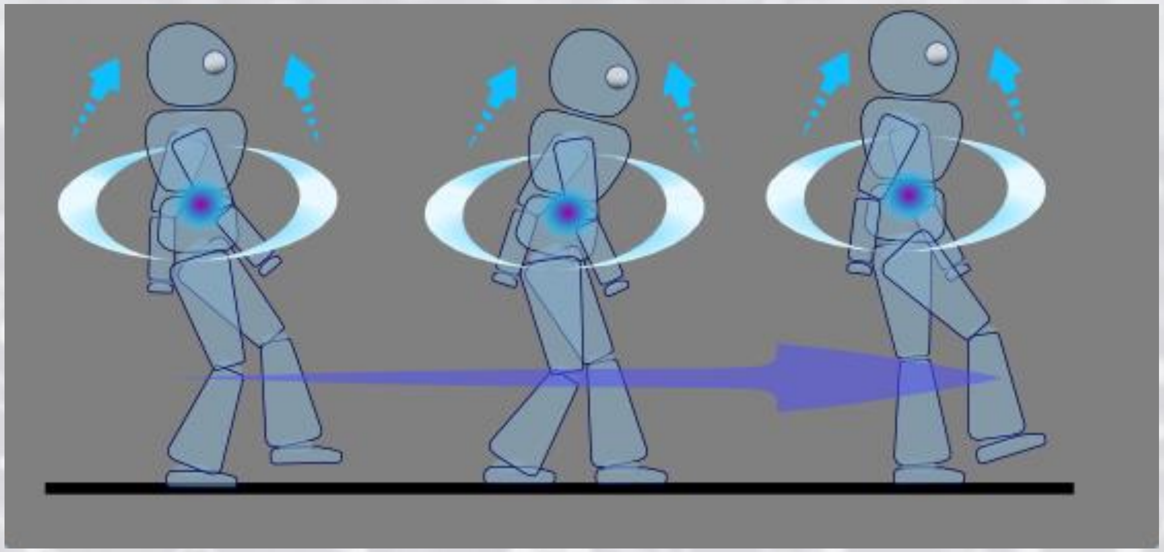
Objective: Balance a pole on top of movable cart.

State: Angle, angular speed, position, horizontal velocity.

Action: Horizontal force applied to cart.
Reward: +1 at each time step if the pole is upright.

https://www.youtube.com/watch?v=_Mmc3i7jZ2c

Robot Locomotion (and pancake flipping!)



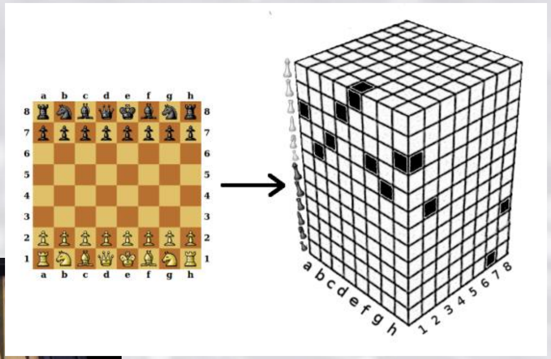
Objective: Make the robot move forward successfully.

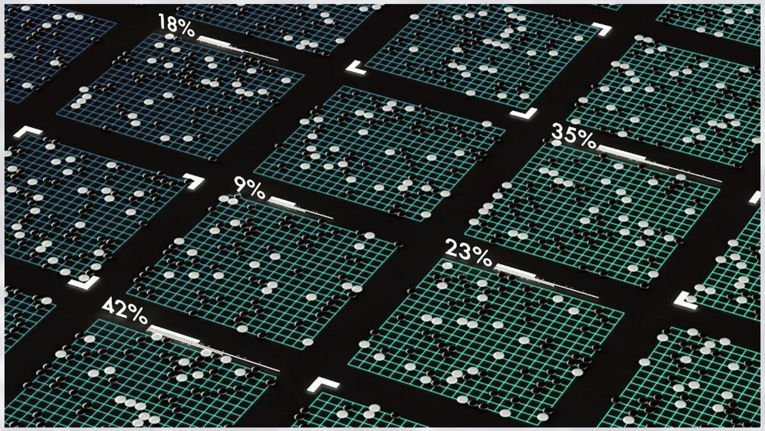
State: Angle and position of joints.
Action: Torques applied on joints.
Reward: +1 at each time step the robot is upright and moving forward.

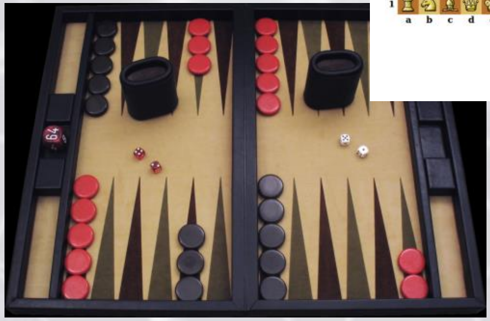
https://www.youtube.com/watch?v=gn4nRCC9TwQ

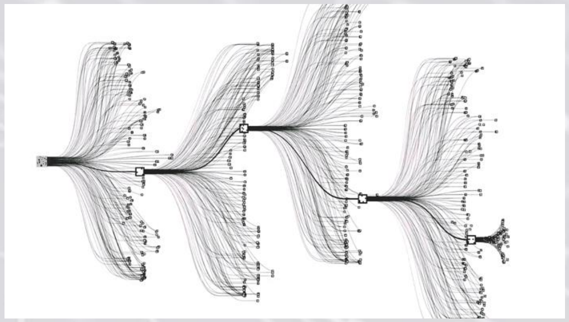
https://www.youtube.com/watch?v=W_gxLKSsSIE

Board Games (Backgammon, Chess, Go)





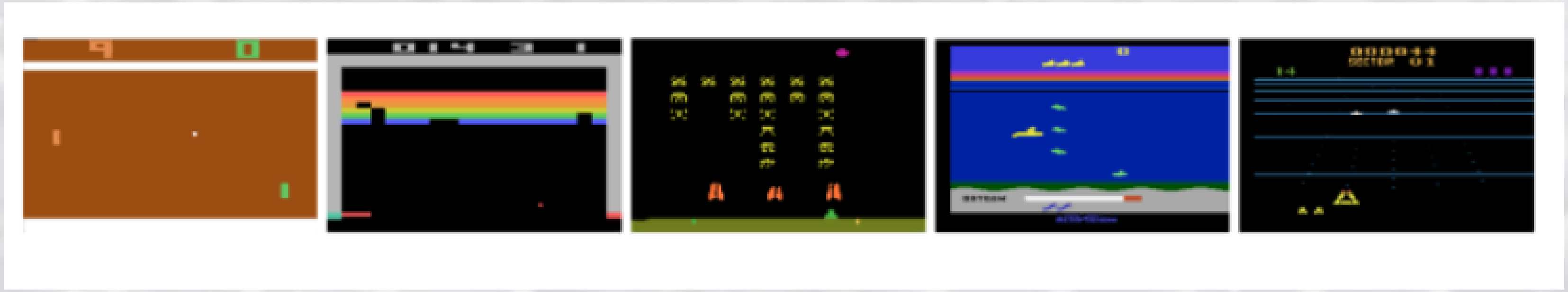




Objective: Win the game.

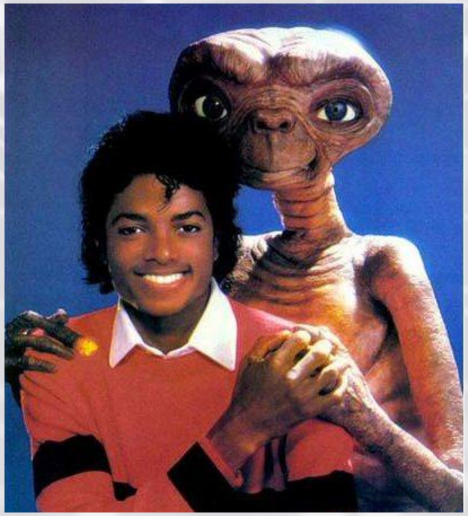
State: Position of pieces.
Action: Next move/placement of next piece.
Reward: +1 for win, 0 for loss.

Atari Games



Objective: ~~World domination~~ Obtain high score (make lots of human friends in the process).

State: Raw pixel inputs.
Action: Game controls, e.g., movement and zap!
Reward: Score differential.



A fatal exce e current
application

* Press any
* Press CTR ou will
  lose any

Press any key to continue _

Exploitation vs. Exploration

- *On-line versus off-line learning*
- On-line learning requires the correct balance between “exploitation” and “exploration”

Exploitation

Exploit current good strategies to obtain known reward

Exploration

Explore new strategies in hope of finding ways to increase reward

Multi-armed Bandits

- Stochastic Multi-armed Bandit
  - Set of n arms
  - Each arm is associated with an unknown reward distribution supported on [0,1] with mean θ_i
  - Each time, sample an arm and receive the reward independently drawn from the reward distribution

classic problems in stochastic control, stochastic optimization and online learning

Robby the Robot can learn via reinforcement learning

Sensors:

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

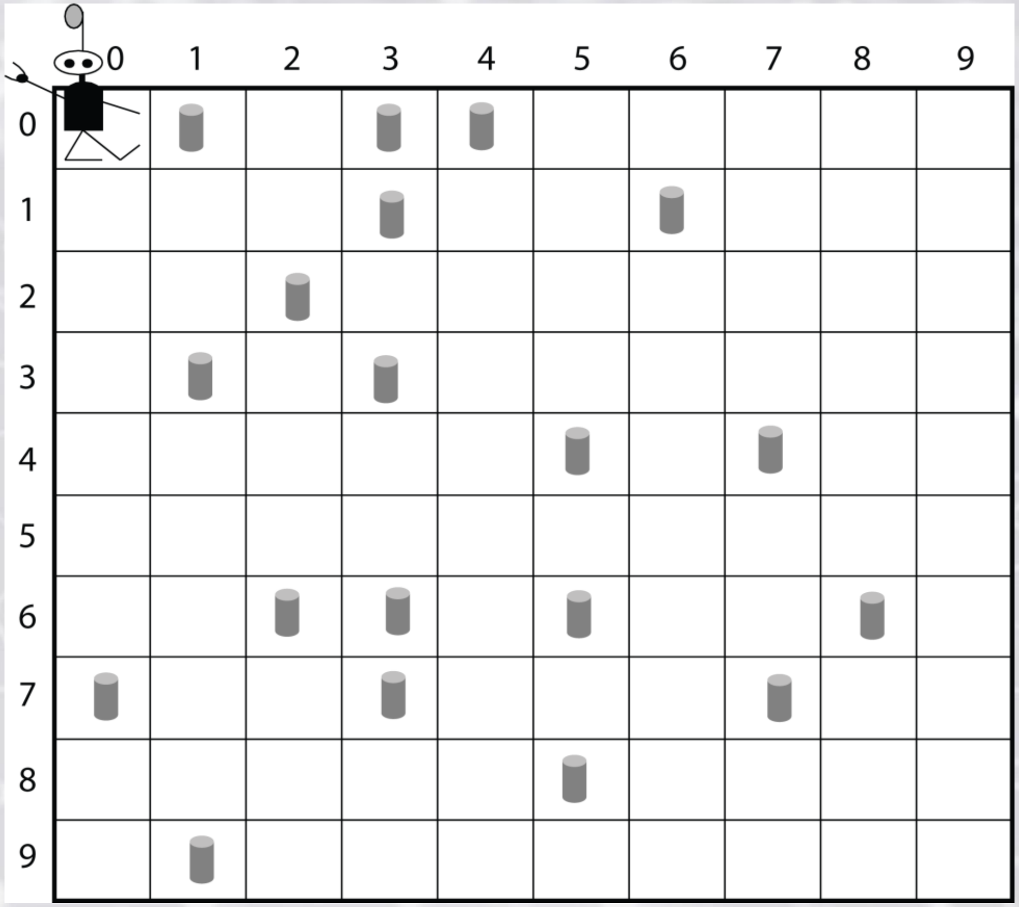
"policy" = "strategy"

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



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

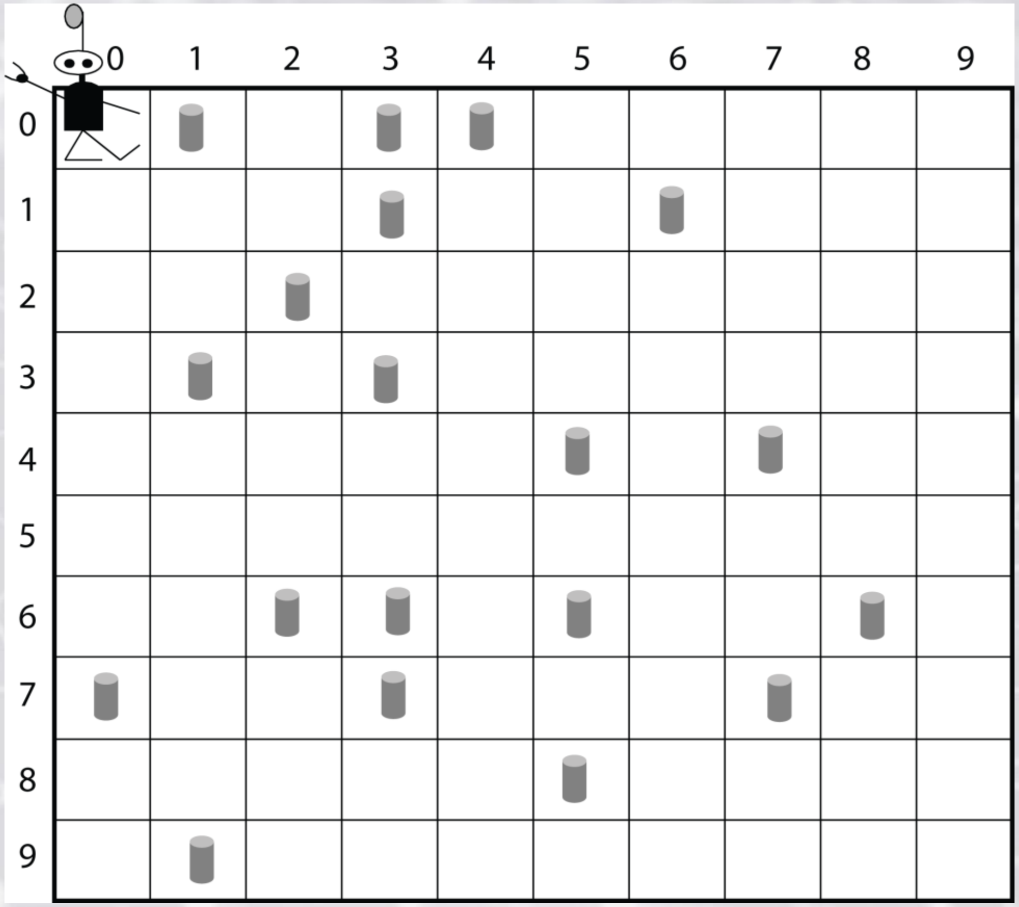
Rewards/Penalties (points):

Picks up can: 10

Pick up can on empty site: -1

Crashes into wall: -5

What is a good policy?



Markov Decision Processes

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

Markov Decision Process (MDP)

$\mathbf{S}$ - Set of States

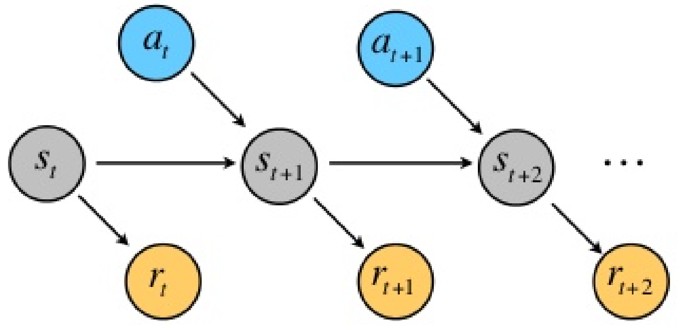
$\mathbf{A}$ - Set of Actions

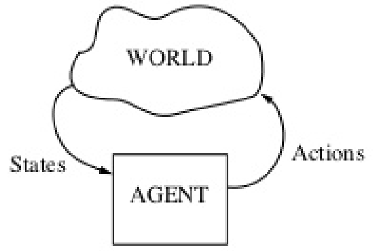
$\Pr(s' \mid a, s)$ - Transitions

α - Starting State Distribution

γ - Discount Factor

$\mathbf{r}(\mathbf{s})$ - Reward [or $r(s,a)$]





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)

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

- Reward function (unknown to learner L)

$$r : S \times A \to R \quad r(s_t, a_t) = r_t$$

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)

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

- Reward function (unknown to learner L)

$$r : S \times A \rightarrow R \quad r(s_t, a_t) = r_t$$

Note: Both δ and r can be deterministic or probabilistic.

In the Robby the Robot example, they are both deterministic

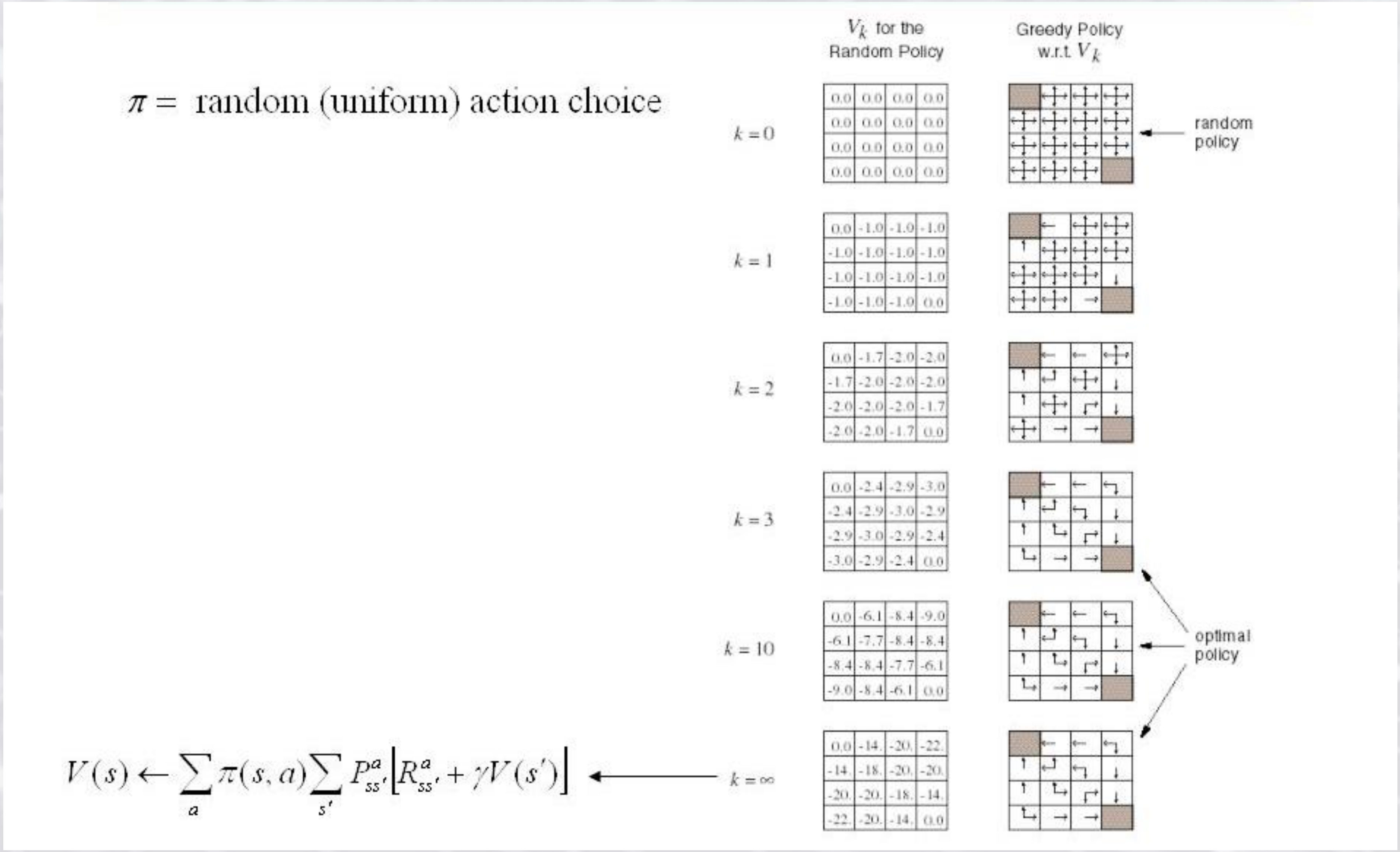
MDPs

- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(\cdot|s_t,a_t)$
  - Environment samples next state $s_{t+1} \sim P(\cdot|s_t,a_t)$
  - Agent receives reward r_t and next state s_{t+1}

A policy **π** is a function from S to A that specifies what action to take in each state.

Objective: find a policy **π*** that maximizes cumulative discount reward: $\sum_{t>0} \gamma^t r_t$

GridWorld Example



$\pi =$ random (uniform) action choice

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$$V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V(s') \right] \longleftarrow k = \infty$$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Finding an Optimal Policy π*

- We wish to determine a policy **π*** that maximizes the sum of rewards: $\sum_{t>0} \gamma^t r_t$

Question: How do we deal with uncertainty in agent-based searches?

Finding an Optimal Policy π*

- We wish to determine a policy **π*** that maximizes the sum of rewards: $\sum_{t>0} \gamma^t r_t$

Question: How do we deal with uncertainty in agent-based searches? Answer: Introduce the formalism of probability (and ***expectation***).

Formally: $\pi^* = \arg\max_{\pi} E\left[\sum_{t\geq 0} \gamma^t r_t \mid \pi\right]$

$$\text{with } s_0 \sim p(s_0),\ a_t \sim \pi(\cdot \mid s_t),\ s_{t+1} \sim p(\cdot \mid s_t, a_t)$$

Value Function and Q-Value

- Following a policy produces sample trajectories (or paths): s_0, a_0, r_0, s_1, a_1, r_1,…

(1) How to assess value of a particular state?

The value function at state s, is the expected cumulative reward from following the policy π from state s:

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

Value Function and Q-Value

- Following a policy produces sample trajectories (or paths): s_0, a_0, r_0, s_1, a_1, r_1,…

(1) How to assess value of a particular state?

The value function at state s, is the expected cumulative reward from following the policy π from state s:

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

(2) What is the quality of a state-action pair?

The Q-value function for state-action pair (s,a) is the expected cumulative reward from taking action *a* in state *s* and then following policy π:

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

Bellman Equations

- The optimal Q-value function Q* is the maximum expected cumulative reward achievable from a given state-action pair:

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

Where Q* satisfied the so-called <u>Bellman equations</u>:

$$Q^*(s,a) = E_{s'\sim\varepsilon}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$

Idea: If the optimal state-action values for the next time-step Q*(s',a') are known, then then optimal strategy is to take the action that maximizes the expected value of : $r + \gamma \max_{a'} Q^*(s',a')$

The optimal policy **π*** corresponds with taking the best action in any state specified by Q*.

Value Iteration

- One strategy: **Value Iteration** (V.I.) algorithm – simply use Bellman Equations as iterative update:

$$Q_{i+1}(s,a) = E\left[r + \gamma \max_{a'} Q_i(s',a') \mid s,a \right]$$

It follows that:

$$\lim_{i \to \infty} Q_i = Q^*$$

However, in general V.I. is not scalable, as we must compute Q(s,a) for every state-action pair (consider Go, etc.).

Q-Learning

- Alternative strategy: **Q-learning.**

Idea: Use function approximation to estimate the action-value function:

$$Q(s,a;\theta) \approx Q^*(s,a)$$

Θ: function parameters/weights

Q-Learning

- Alternative strategy: **Q-learning.**

Idea: Use function approximation to estimate the action-value function:

$$Q(s,a;\theta) \approx Q^*(s,a)$$

Θ: function parameters/weights

If Q is a deep neural network, we get: **deep Q-learning**.

Q-Learning

- Alternative strategy: **Q-learning.**

Idea: Use function approximation to estimate the action-value function:

$$Q(s,a;\theta) \approx Q^*(s,a)$$

Θ: function parameters/weights

If Q is a deep neural network, we get: **deep Q-learning**.

What should the system learn? (recap)

Policy Learning:

Learn function π *that directly* maps states to actions:

$$\pi(s) = a$$

What should the system learn? (recap)

Policy Learning:

Learn function π *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? (recap)

Policy Learning:

Learn function π *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:

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

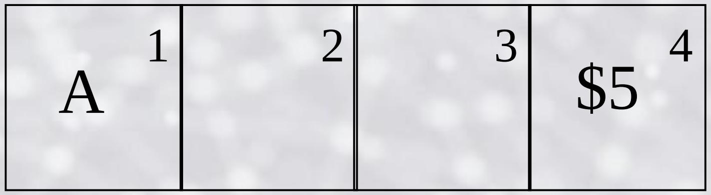
More generally, we include a “discount factor”, *γ* (between 0 and 1), that expresses how much we value immediate versus future rewards.

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

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

Example



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

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

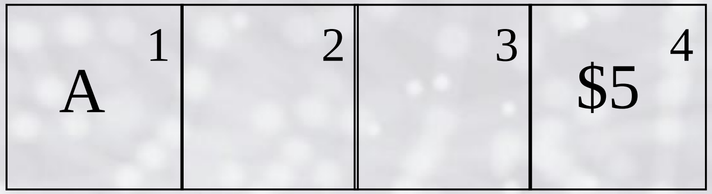
Only action in square 4 is **S**top

No other rewards or penalties.

Set $\gamma = .9$

Set $\eta = 1$

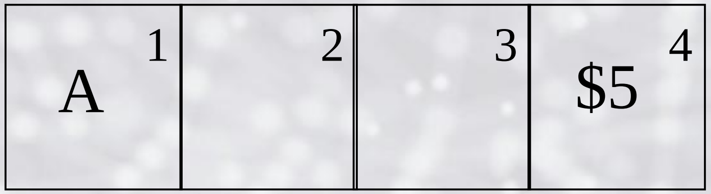
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

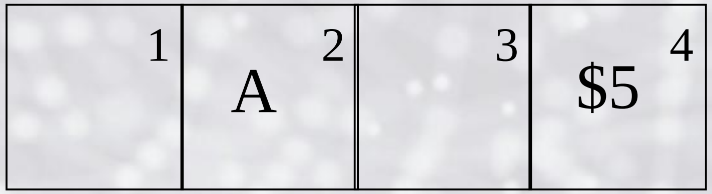
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q(s, a)})$$



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

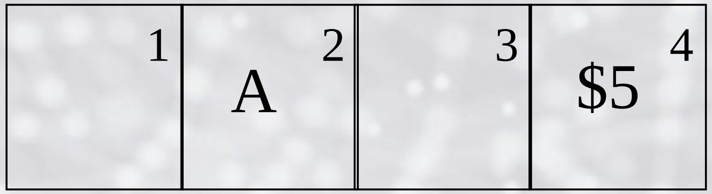
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

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

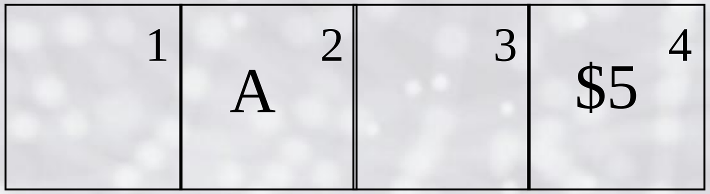
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s},\, \mathrm{a}))$$



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

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

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

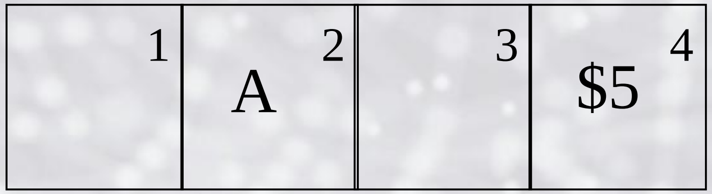


Episode 1
Current state *s* = 1
Action = F
r = 0
s' = 2

$$Q(1,F) = 0 + \max_{a'}\left[Q(2,a')\right] = 0$$

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

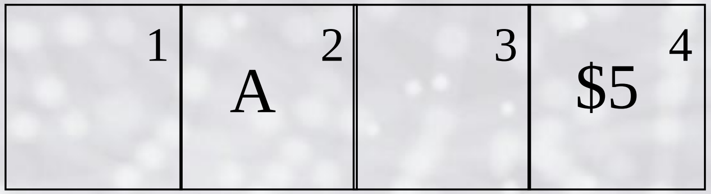
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s},\,\mathrm{a}))$$



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

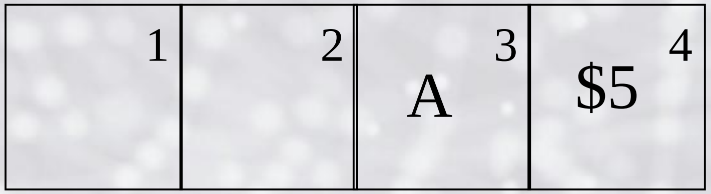
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

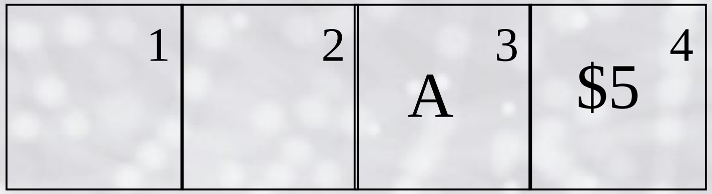
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

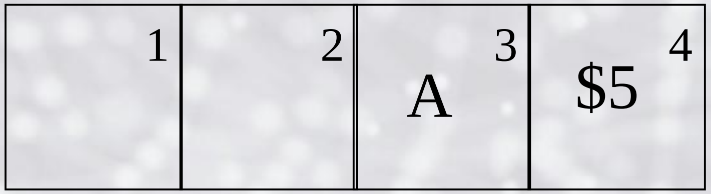
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s},\mathrm{a}))$$



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

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

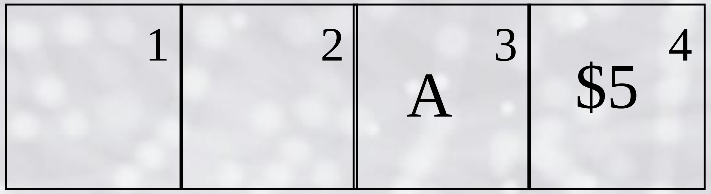


Episode 1
Current state s = 2
Action = F
r = 0
s' = 3

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

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

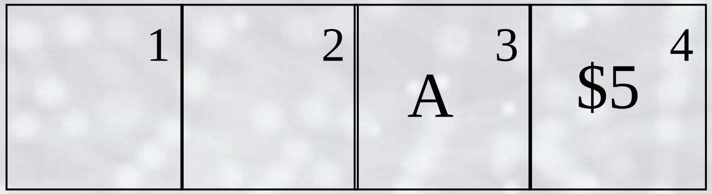
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s}, \mathrm{a}))$$



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

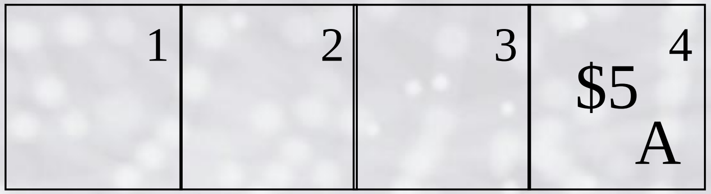
$$Q(s,a) \Leftarrow Q(s,a) + \eta\, (r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

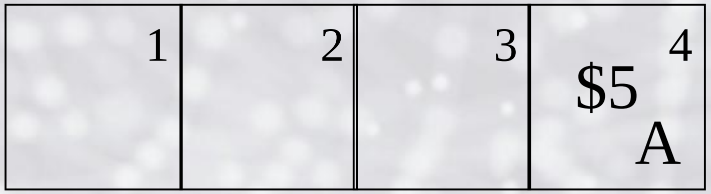
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

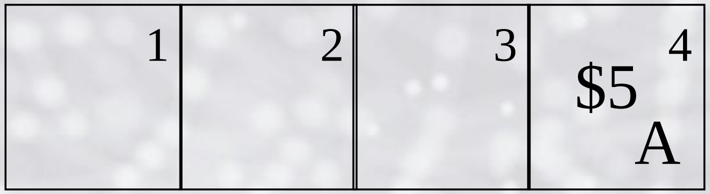
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

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

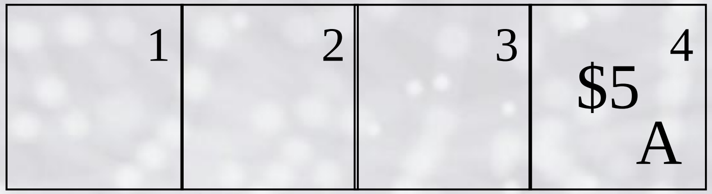


Episode 1
Current state *s* = 3
Action = F
r = $5
s' = 4

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

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

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

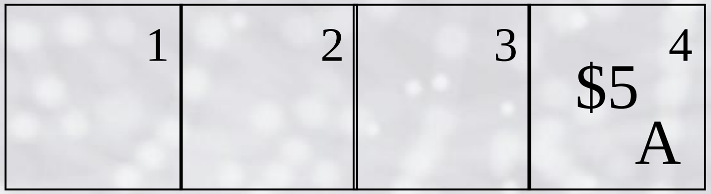


Episode 1
Current state *s* = 3
Action = F
r = $5
s' = 4

$$Q(3,F) = 0 + (\$5 + .9 \max_a Q(s',a') - Q(s,a)) = \$5$$

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

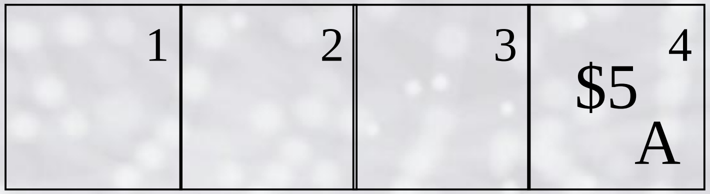
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q(s,\,a)})$$



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

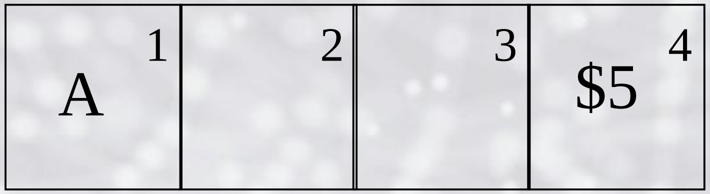
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

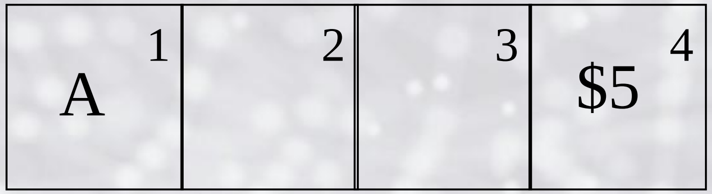
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

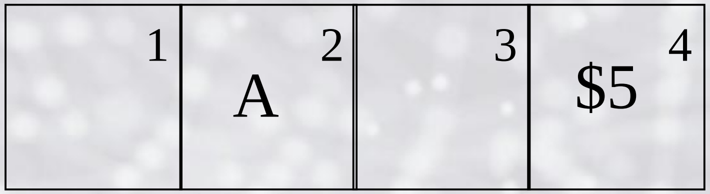
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

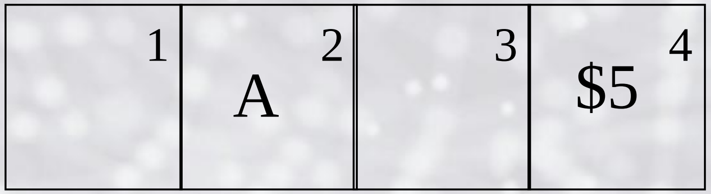
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

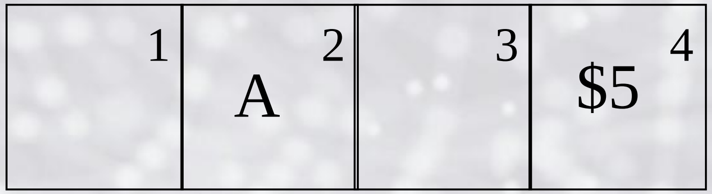
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s}, \mathrm{a}))$$



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

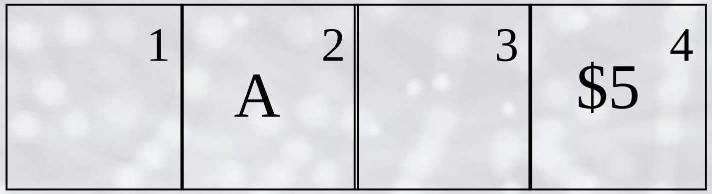
r = 0

s' = 2

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

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

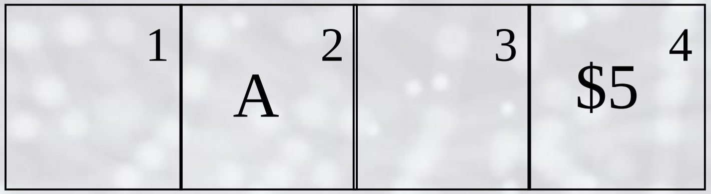
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

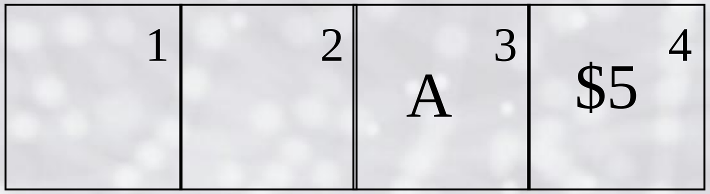
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

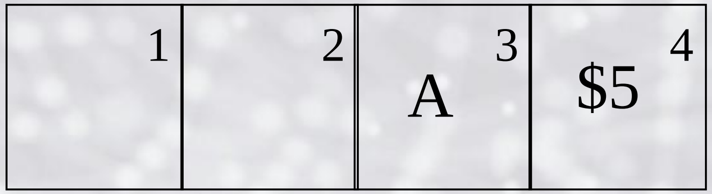
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

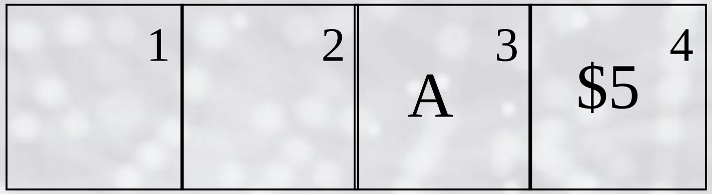
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

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

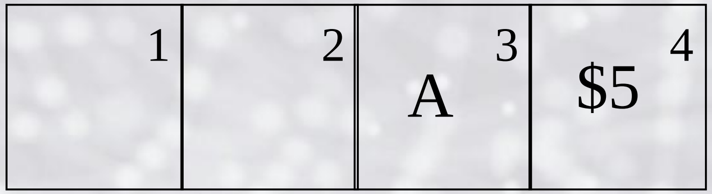


Episode 2
Current state *s* = 2
Action = F
r = 0
s' = 3

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

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

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

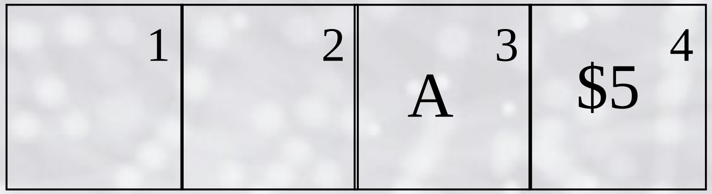


Episode 2
Current state *s* = 2
Action = F
r = 0
s' = 3

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

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

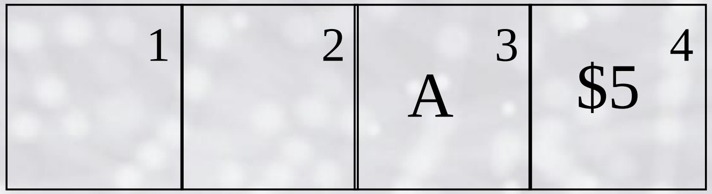
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

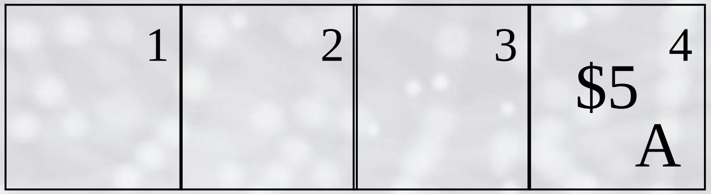
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s}, \mathrm{a}))$$



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

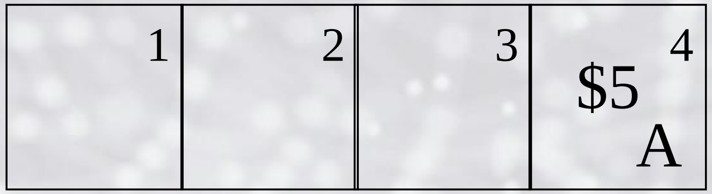
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

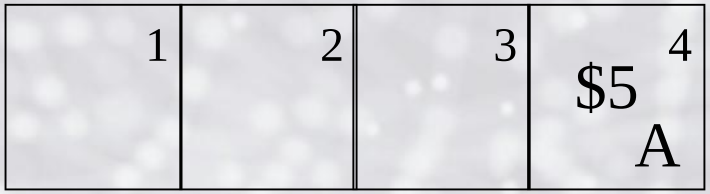
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

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

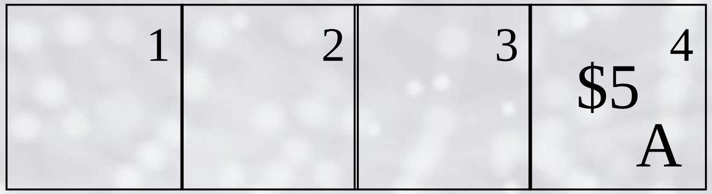


Episode 2
Current state *s* = 3
Action = F
r = $5
s' = 4

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

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

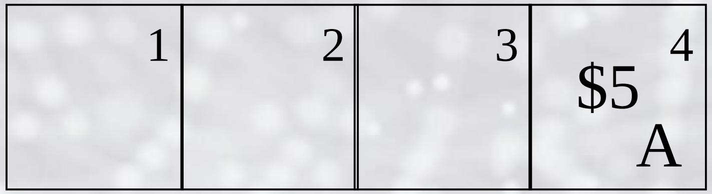
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

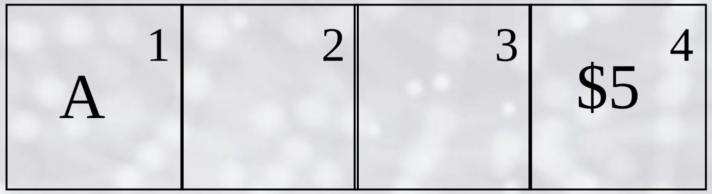
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

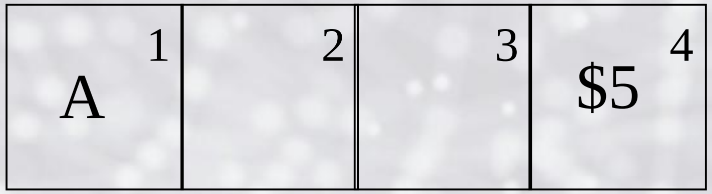
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

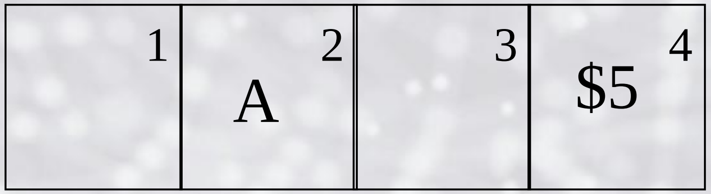
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s}, \mathrm{a}))$$



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

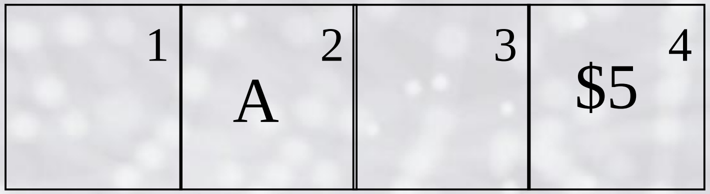
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

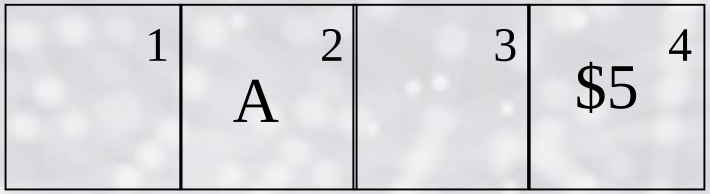
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

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



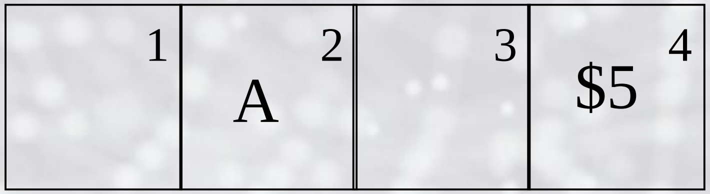
Episode 3
Current state *s* = 1
Action = F
r = 0
s' = 2

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

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

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

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

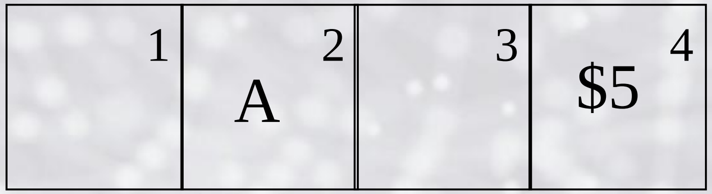


Episode 3
Current state *s* = 1
Action = F
r = 0
s' = 2

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

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

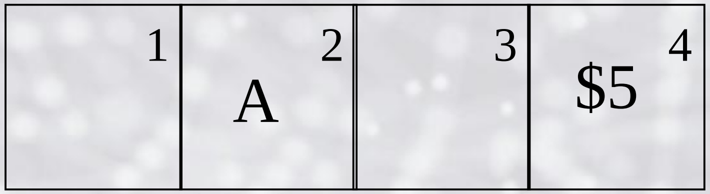
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s}, \mathrm{a}))$$



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

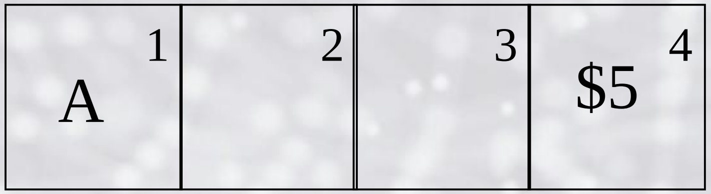
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

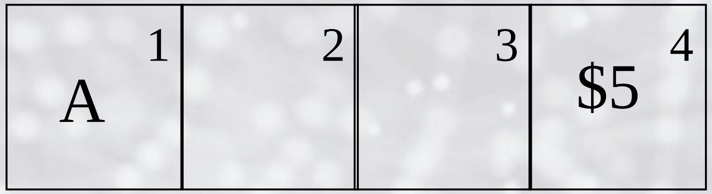
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$$



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

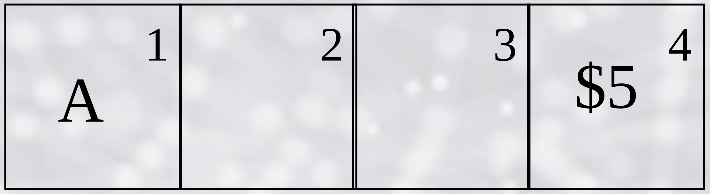
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

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



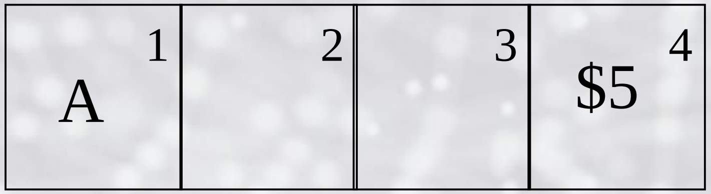
Episode 3
Current state *s* = 2
Action = B
r = 0
s' = 1

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

$$0 + 0 + (.9)(\$4.05) - 0 = \$3.65$$

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

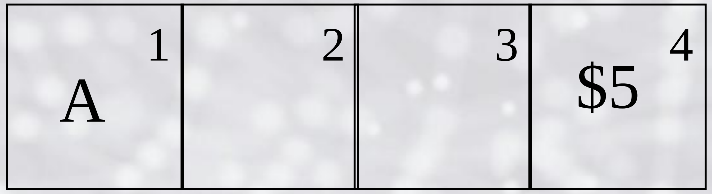
s' = 1

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

$$0 + 0 + (.9)(\$4.05) - 0 = \$3.65$$

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

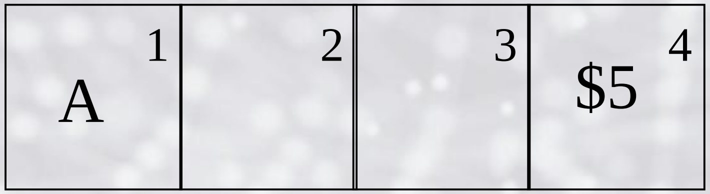
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

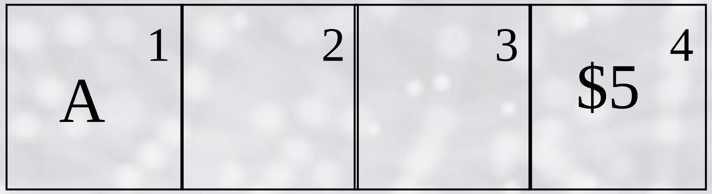
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$



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

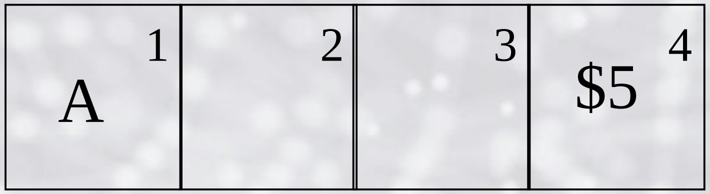
$$Q(s,a) \Leftarrow Q(s,a) + \eta\,(r + \gamma \max_{a'} Q(s',a') - \mathrm{Q}(\mathrm{s}, \mathrm{a}))$$



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

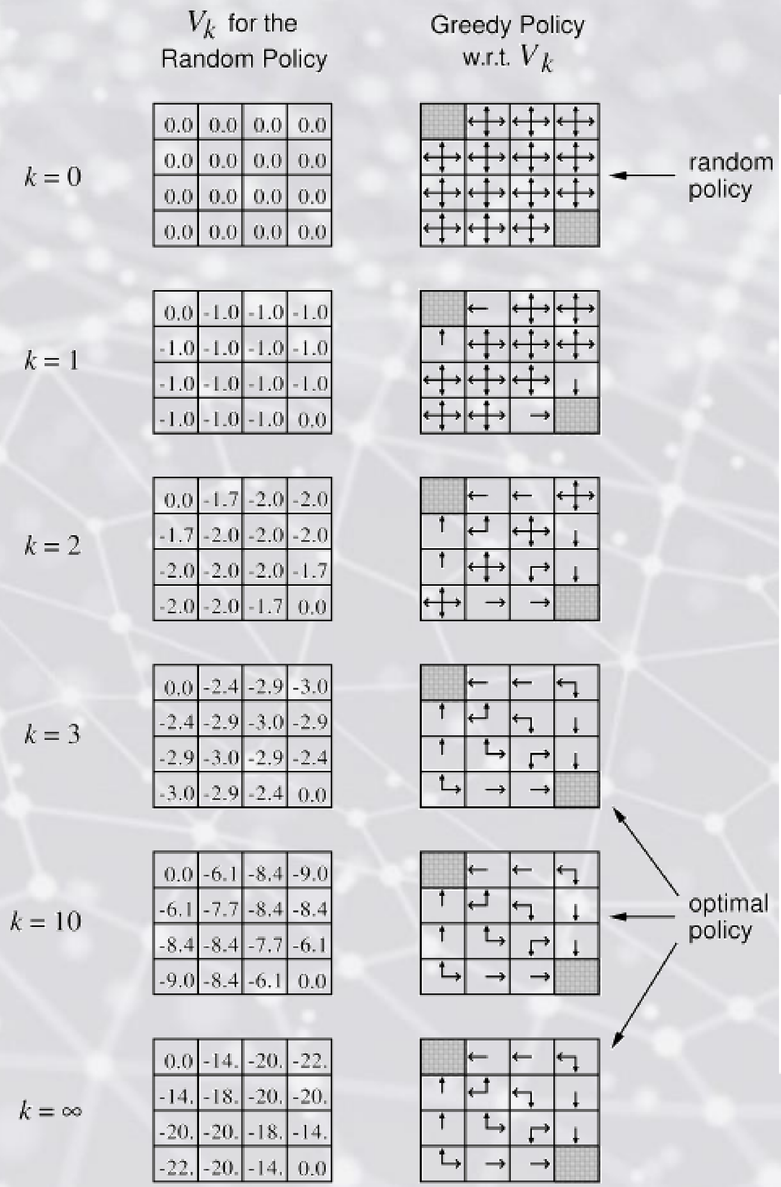
$$Q(1,F) = \$4.05 + (0 + .9 \max_{a} Q(s',a') - \$4.05)$$

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

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

Etc.

Gridworld (revisited)



$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

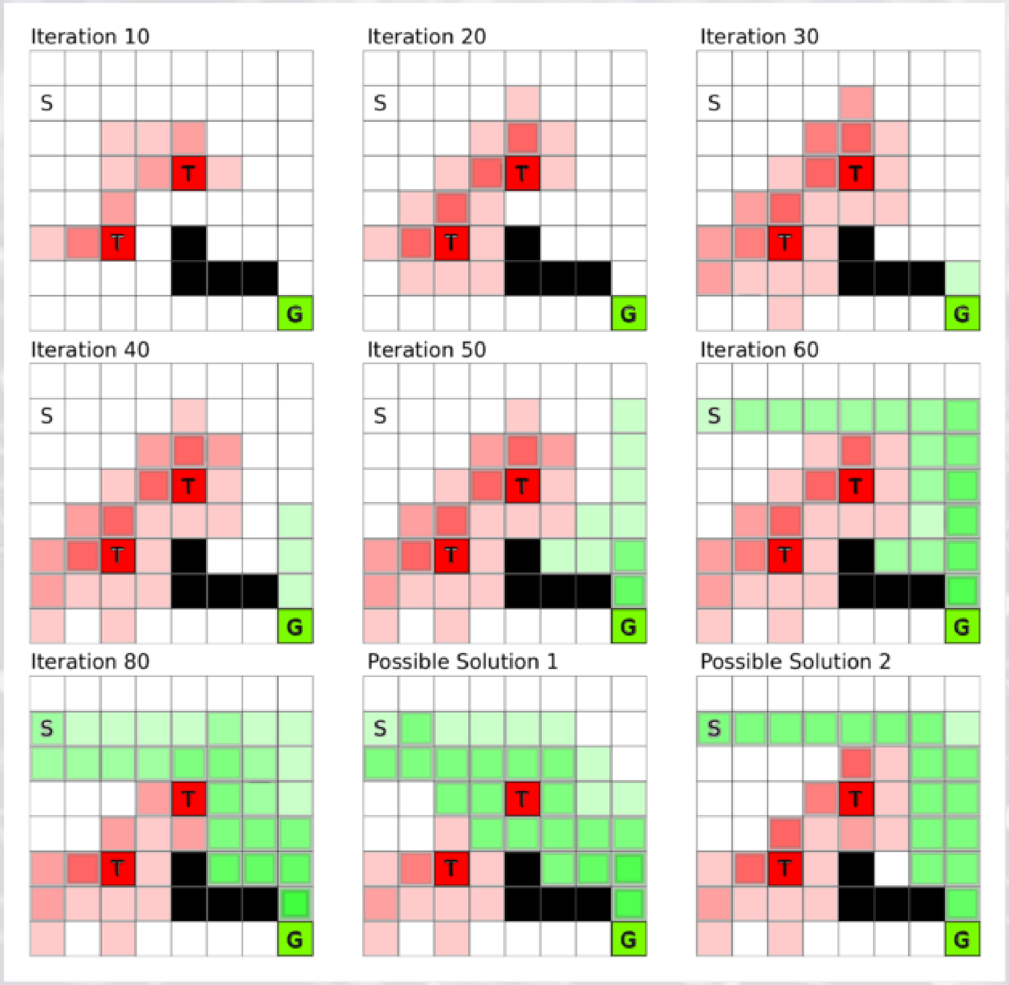
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Some Q-Learning Properties

- Results: Q-learning converges to optimal policy – even if you’re acting suboptimally!
- This is called **off-policy learning**.

Caveats:

- You have to explore sufficiently.
- You have to make learning rate small enough (but also not decrease it too quickly).

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 ε choose random action

- Can start with high ε, and decrease it over the run

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).

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.

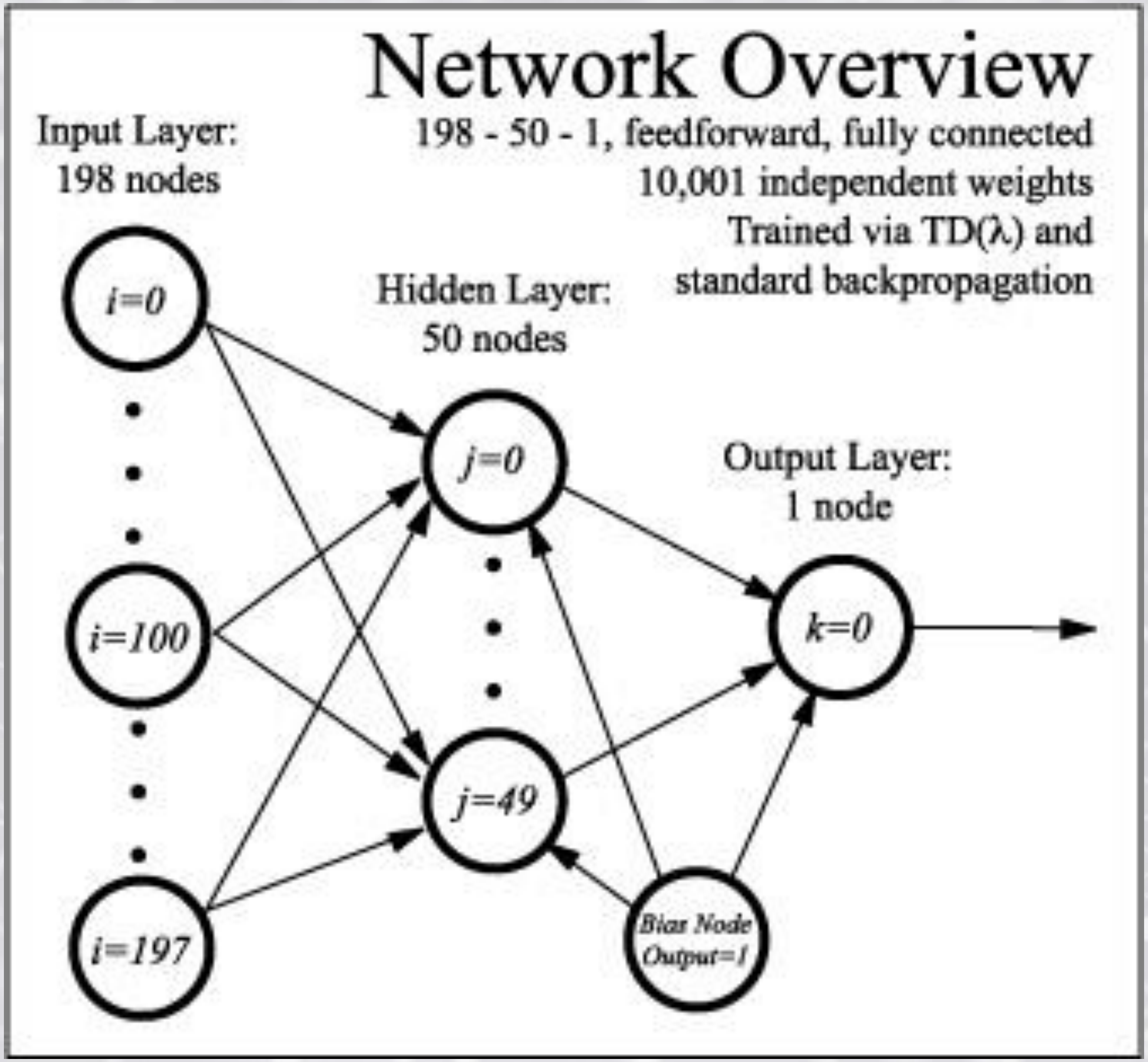
- Won backgammon championship at 1989 International Computer Olympiad.   But not close to beating best human players.

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.

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
Input Layer:
198 nodes
198 - 50 - 1, feedforward, fully connected
10,001 independent weights
Trained via TD(λ) and
standard backpropagation
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.

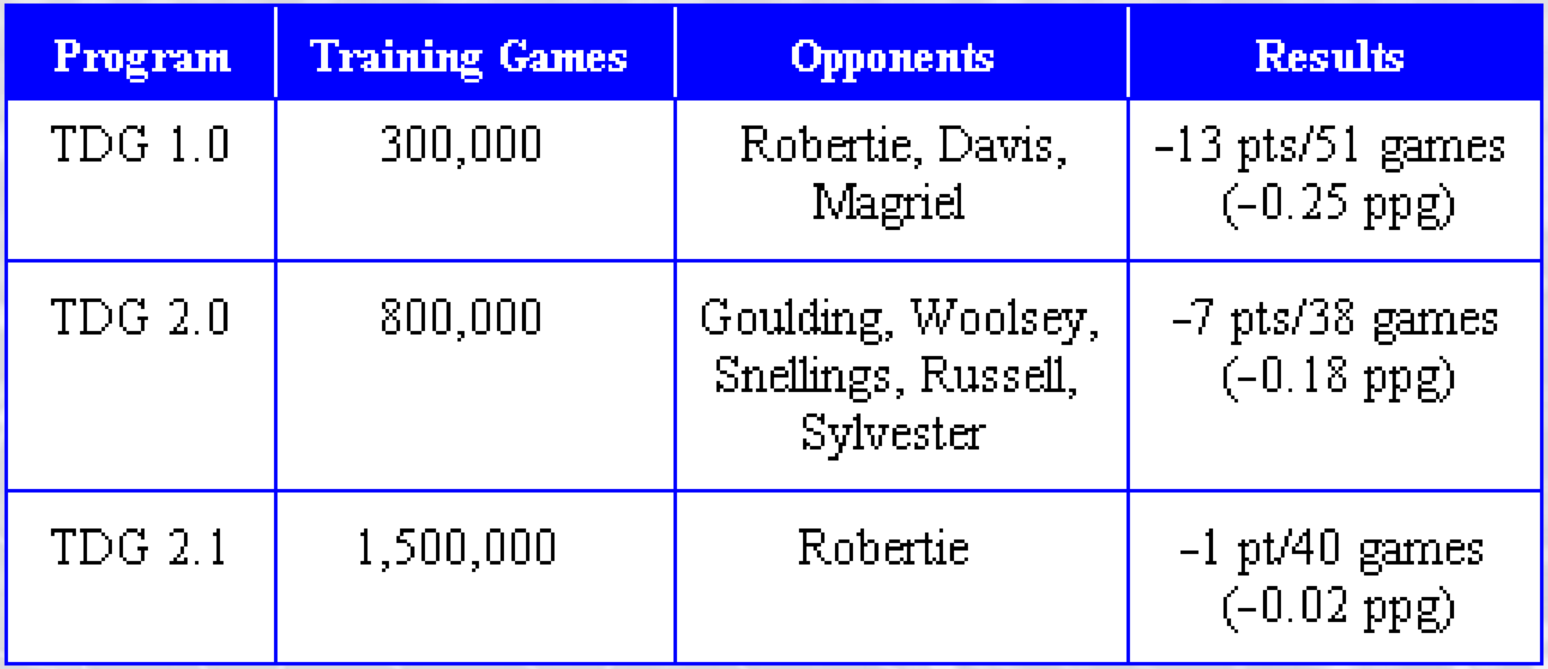
On each turn:

- 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).”

Robby the Robot can learn via reinforcement learning

"policy" = "strategy"

Sensors:

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

Actions:

Move N

Move S

Move E

Move W

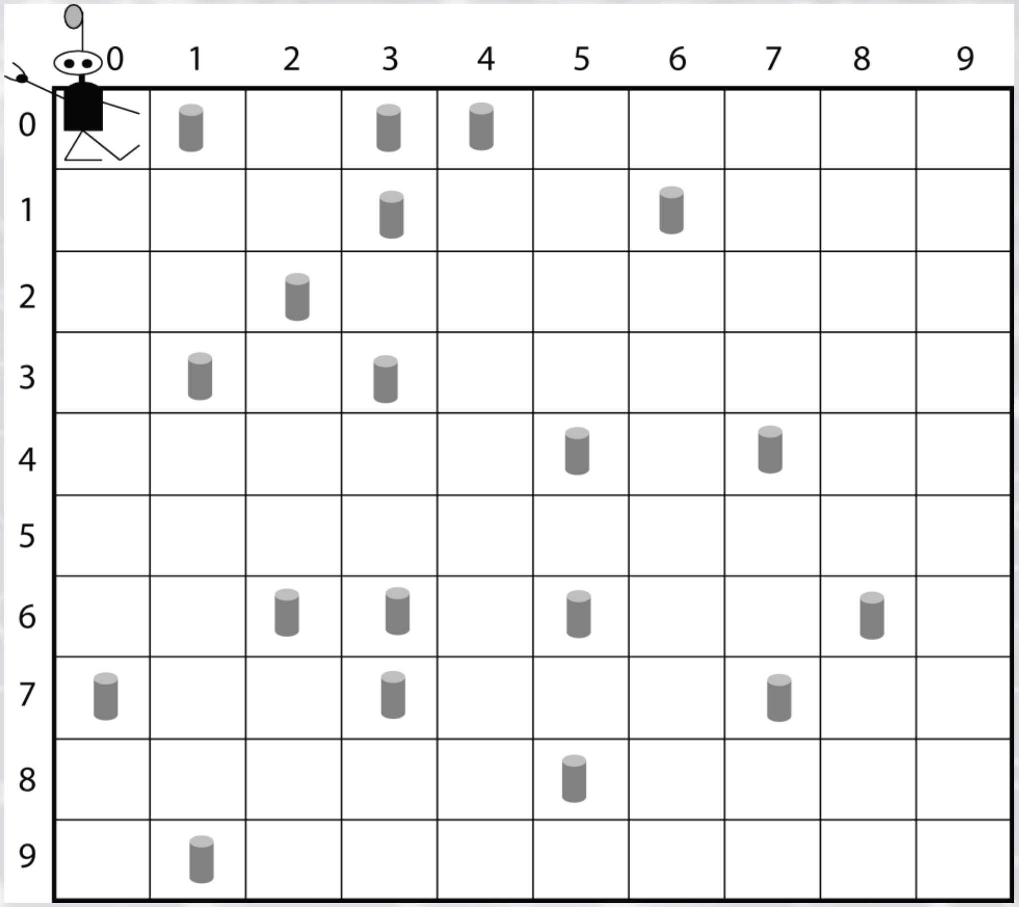
Pick up can

Rewards/Penalties (points):

Picks up can: 10

Pick up can on empty site: -1

Crashes into wall: -5


0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9