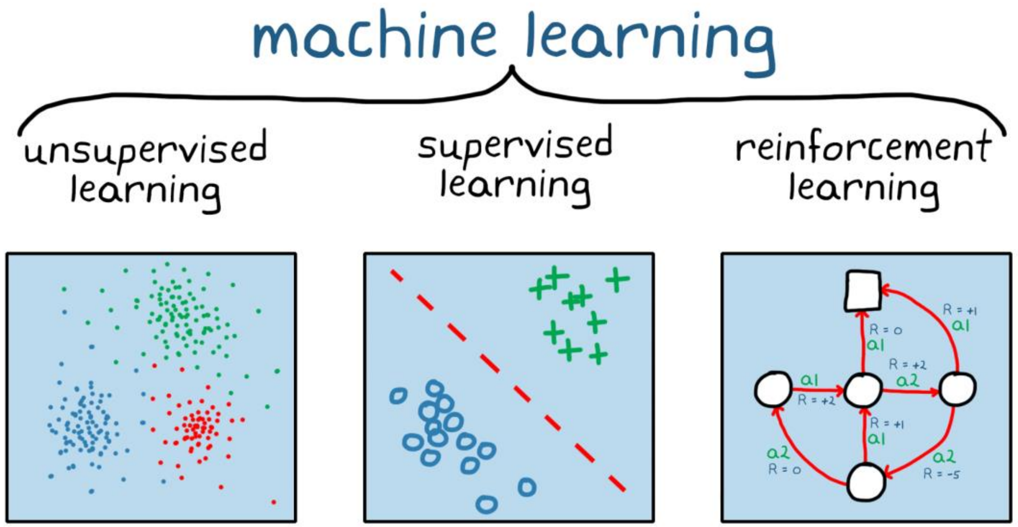
Introduction to Reinforcement Learning (RL)

Vikky Masih,
Research Scholar,
Mehta Family School of Data Science & Artificial Intelligence,
IIT Guwahati

11-12 July 2023, DS&AI Research Scholars' Discussion Group, IITG

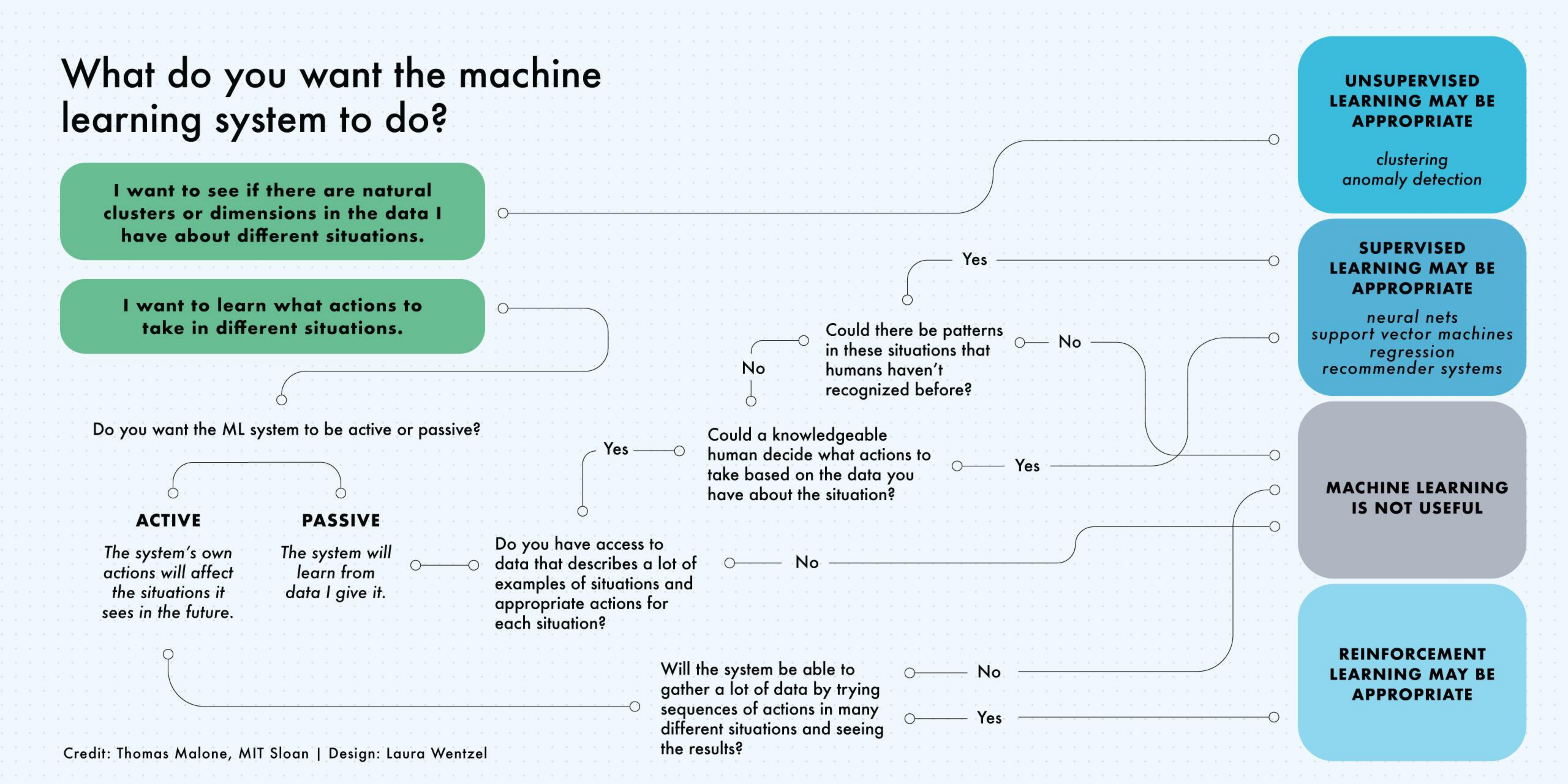
Machine Learning (ML)

- ML is fundamental concept of AI
  - Learn to improve performance via experience



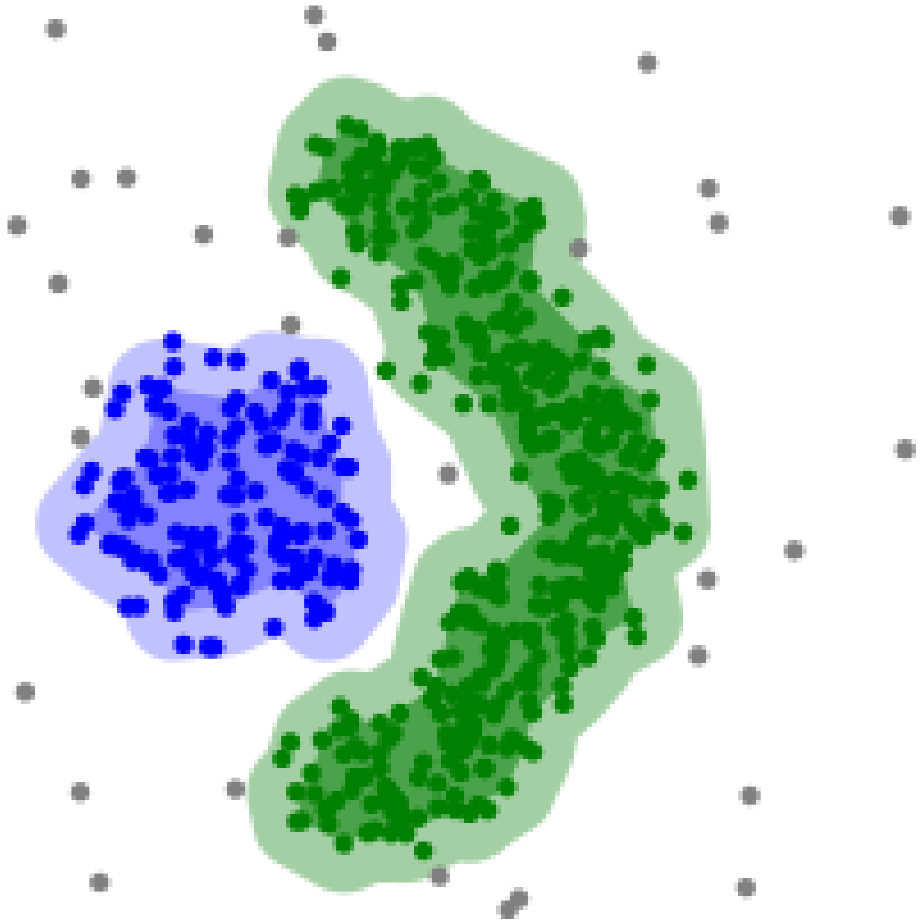
- Unsupervised Learning:
  - Clustering, Anomaly Detection, PCA, etc.
  - Find patterns in input data
- Supervised Learning:
  - Classification and Regression
  - Labelled data for training
- Reinforcement Learning:
  - Decision making under uncertainty
  - Learn to improve performance via interacting with environment

Image Credits: mathworks.com



Ref: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

Machine Learning (ML)

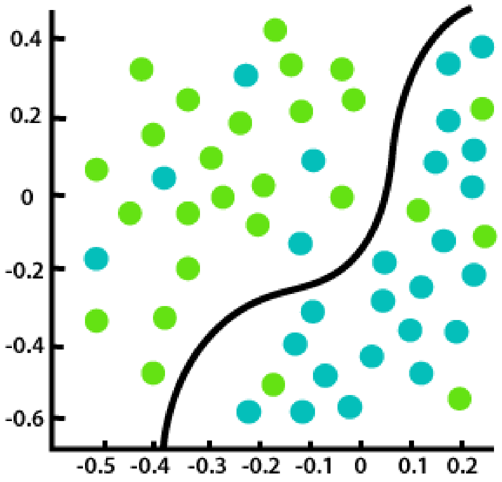


- Unsupervised Learning:
  - Clustering, Anomaly Detection, PCA, etc.
  - Find patterns in input data

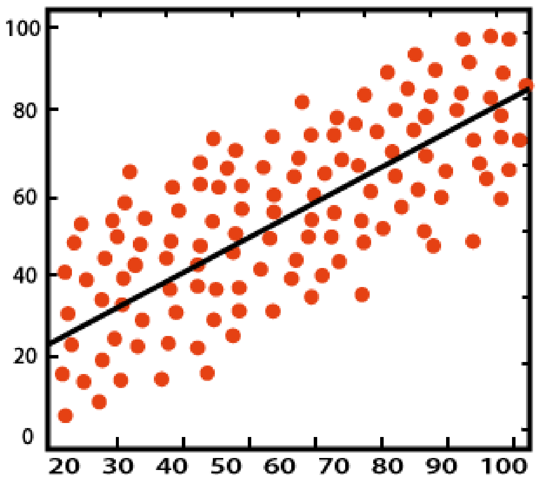
- Unsupervised learning example:
  - Goal: Clustering, Outlier detection
  - Data: $[\overrightarrow{x_i}, \ldots, \overrightarrow{x_n}]$

Image Credits: wikipedia.org

Machine Learning (ML)



Classification

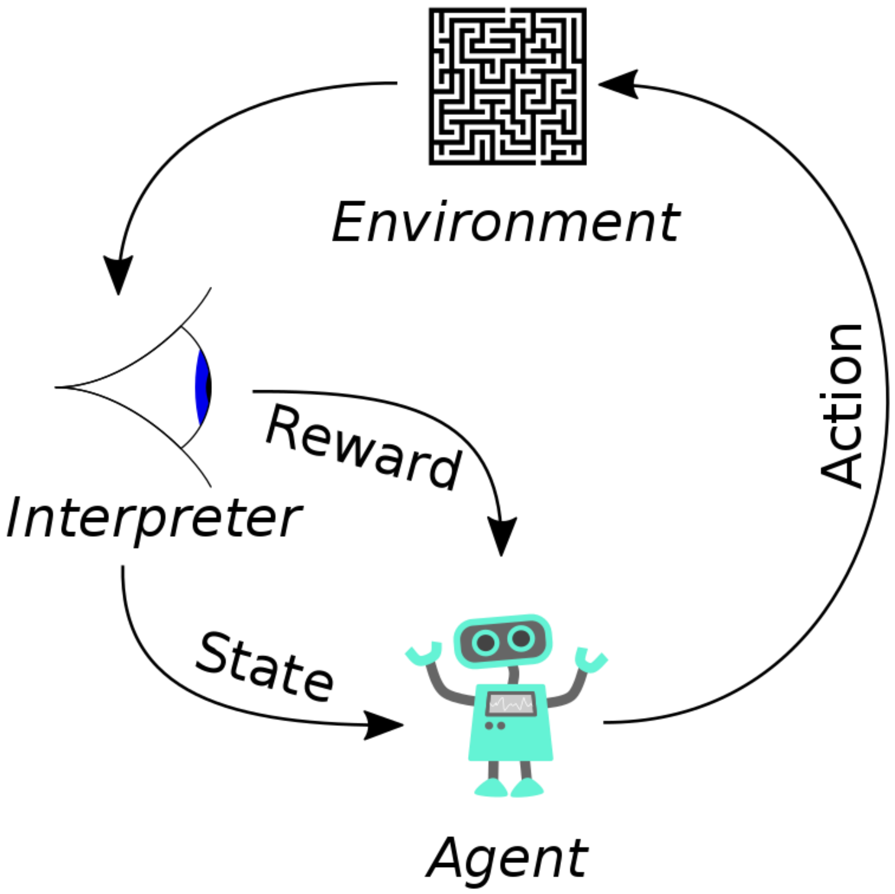


Regression

- Supervised Learning:
  - Classification and Regression
  - Labelled data for training
- Regression:
  - Goal: $f(\vec{x}) = \vec{y},$
  - Data: $[(\overrightarrow{x_i}, \overrightarrow{y_i}), \ldots, (\overrightarrow{x_n}, \overrightarrow{y_n})]$
- Classification:
  - Goal: $\text{argmax}\ P(class|\vec{x}) = \text{C}$
  - Data: $[(\overrightarrow{x_i}, C_i), \ldots, (\overrightarrow{x_n}, C_n)]$

Image Credits: javatpoint.com

Machine Learning (ML)



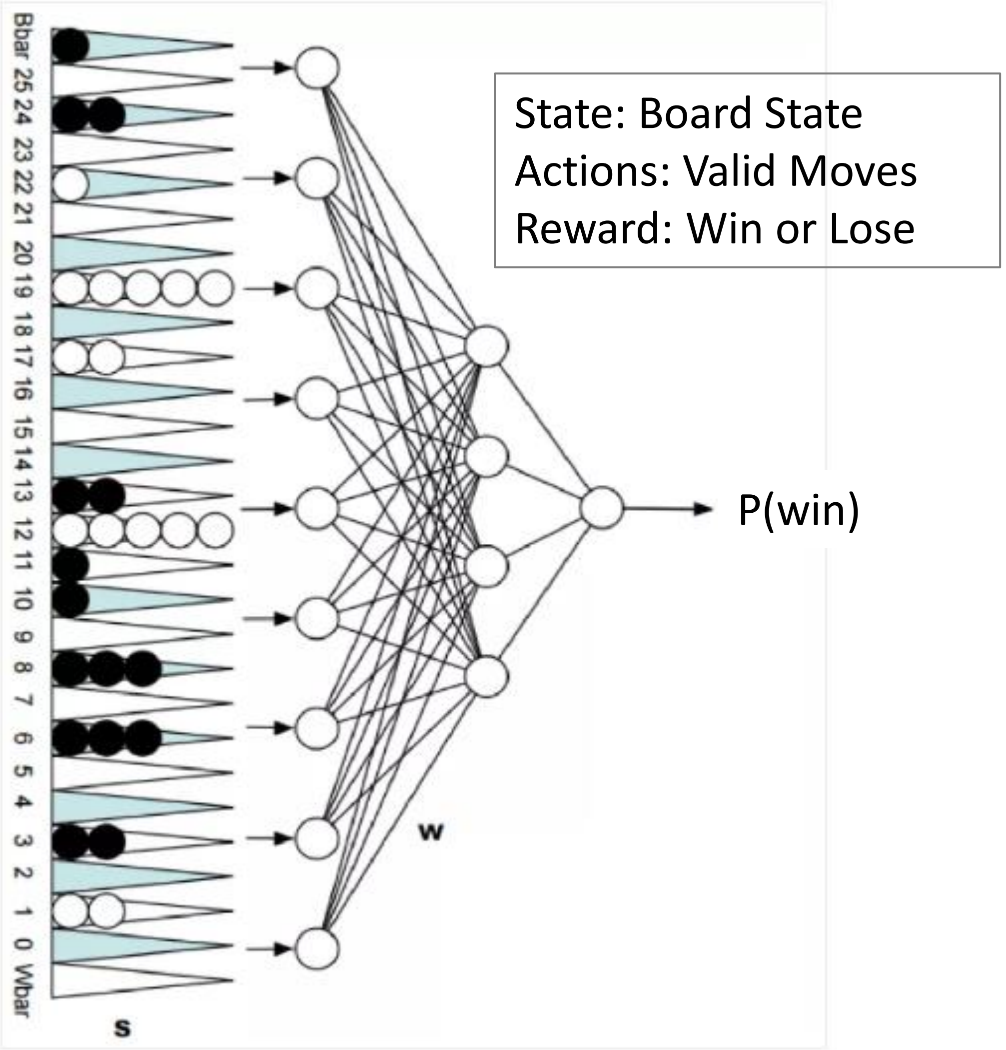
- Reinforcement Learning:
  - Stochastic optimal control
  - Learn to improve performance via interacting with the environment
- RL example:
  - Goal:
    - Maximize cumulative reward

      Maximize $\sum_{i=1}^{\infty} Reward(State_i, Act_i)$
  - Data:

    $Reward_{i+1}, State_{i+1} = Interact(State_i, Action_i)$

Image Credits: wikipedia.org

TD-Gammon – Tesauro ~1995

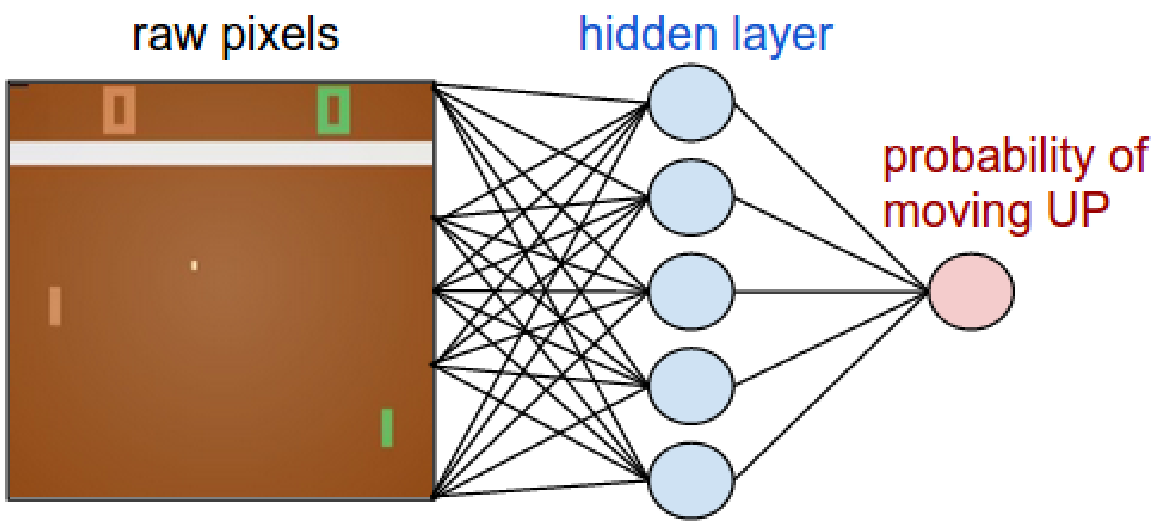


- Net with 80 hidden units, initialize to random weights
- Select move based on network estimate & shallow search
- Learn by playing against itself
- 1.5 million games of training
  - competitive with world class players

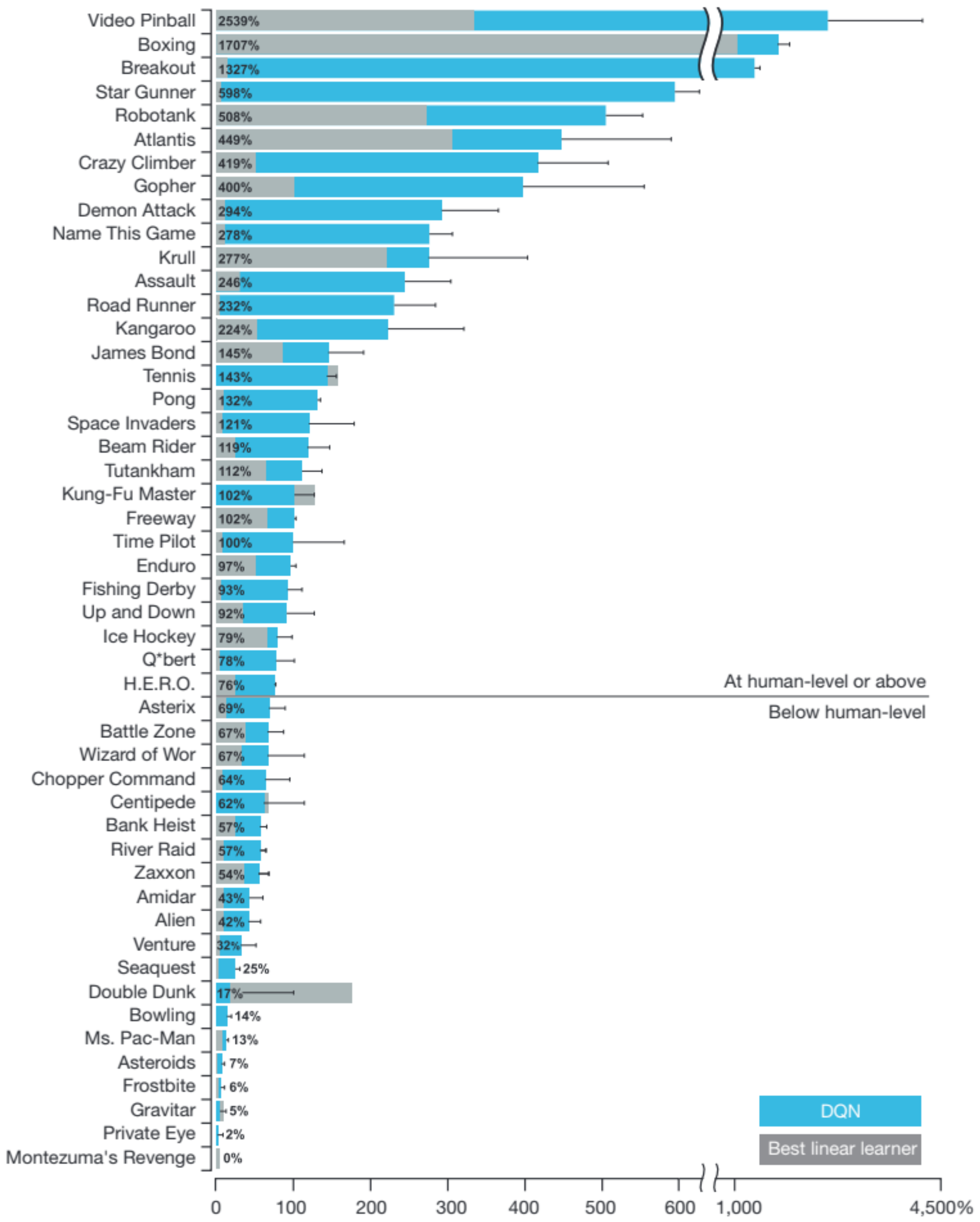
Slide Credits: Geoff Hulten

Atari 2600 games

State: Raw Pixels
Actions: Valid Moves
Reward: Game Score



Same model/parameters for ~50 games



Slide Credits: Geoff Hulten

Robotics and Locomotion

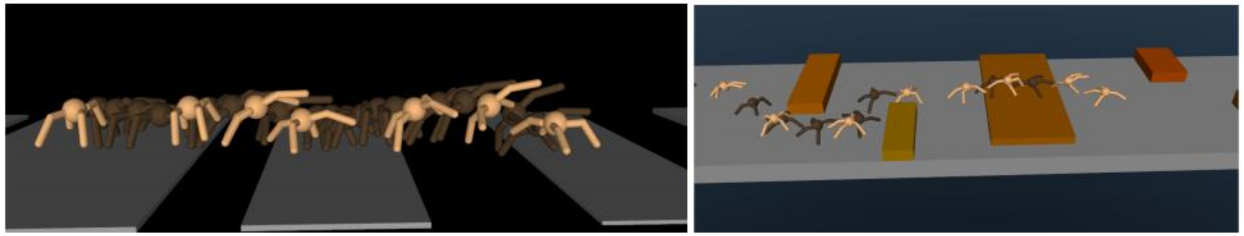
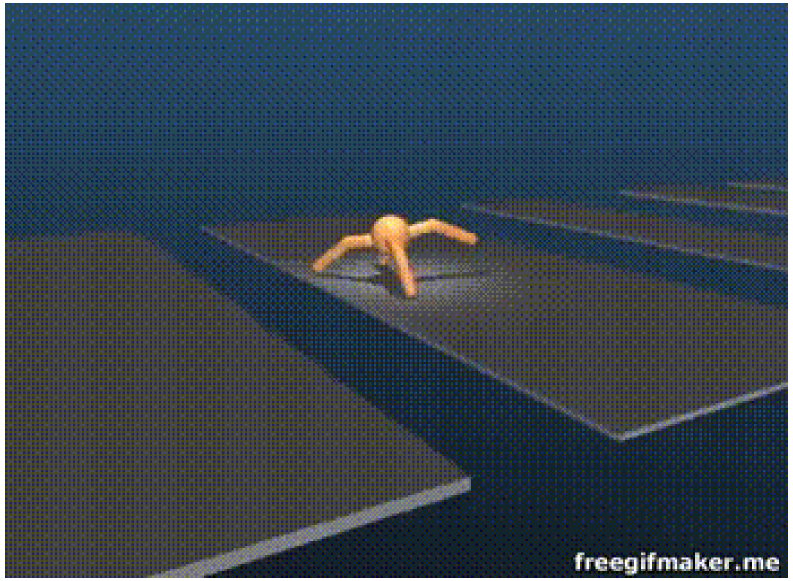


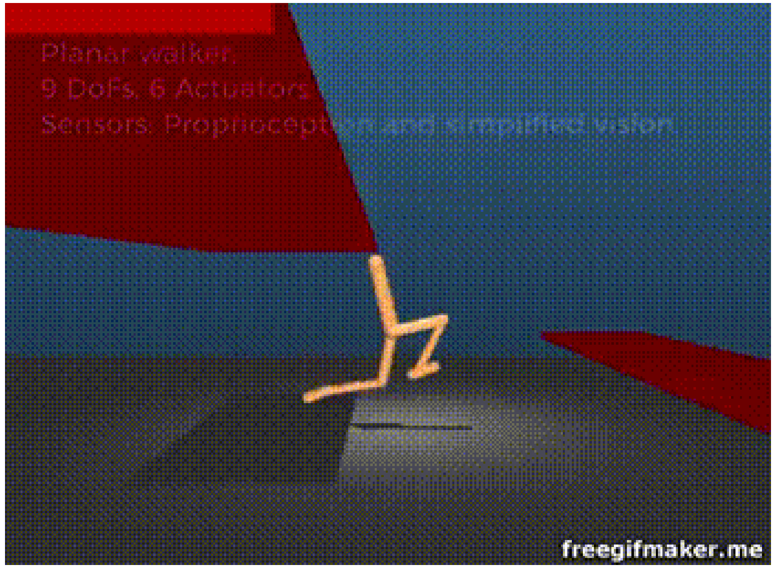
Figure 5: Time-lapse images of a representative *Quadruped* policy traversing gaps (left); and navigating obstacles (right)

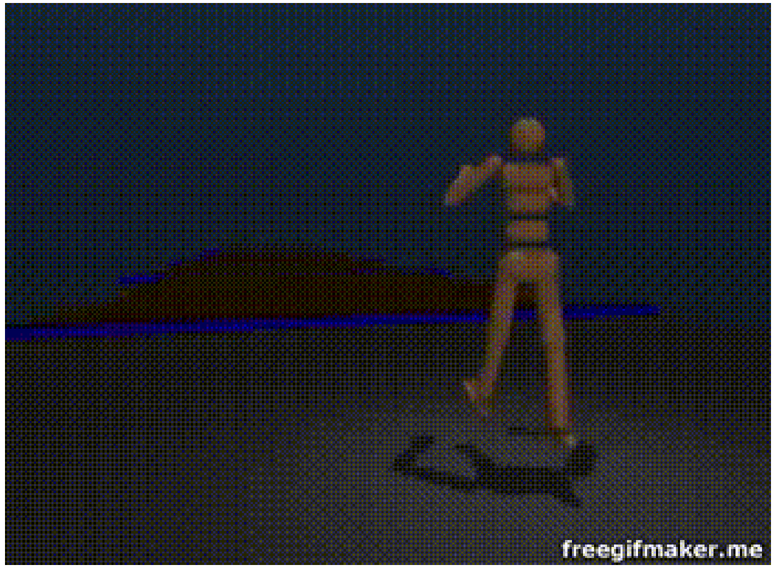
State:
- Joint States/Velocities
- Accelerometer/Gyroscope
- Terrain

Actions: Apply Torque to Joints

Reward: Velocity – { stuff }







https://youtu.be/hx_bgoTF7bs

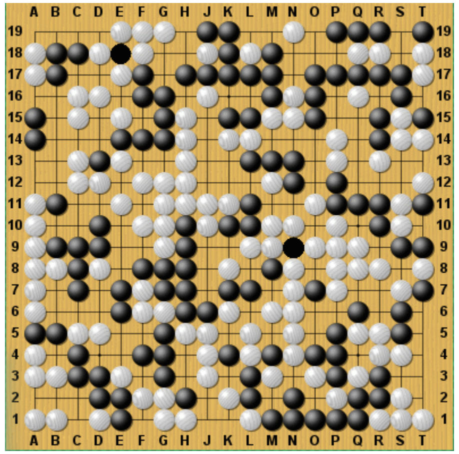
2017 paper https://arxiv.org/pdf/1707.02286.pdf

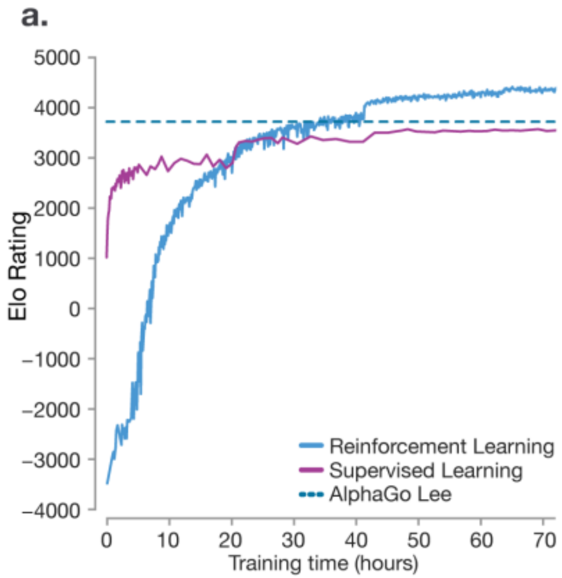
Slide Credits: Geoff Hulten

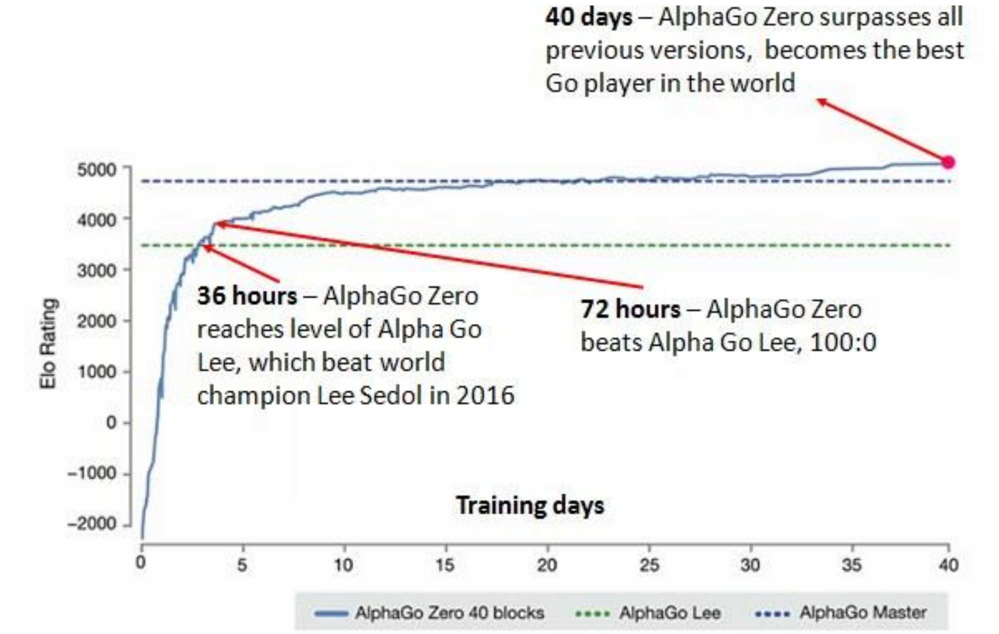
Alpha Go

State: Board State
Actions: Valid Moves
Reward: Win or Lose

- Learning how to beat humans at ‘hard’ games (search space too big)
- Far surpasses (Human) Supervised learning
- Algorithm learned to outplay humans at chess in 24 hours

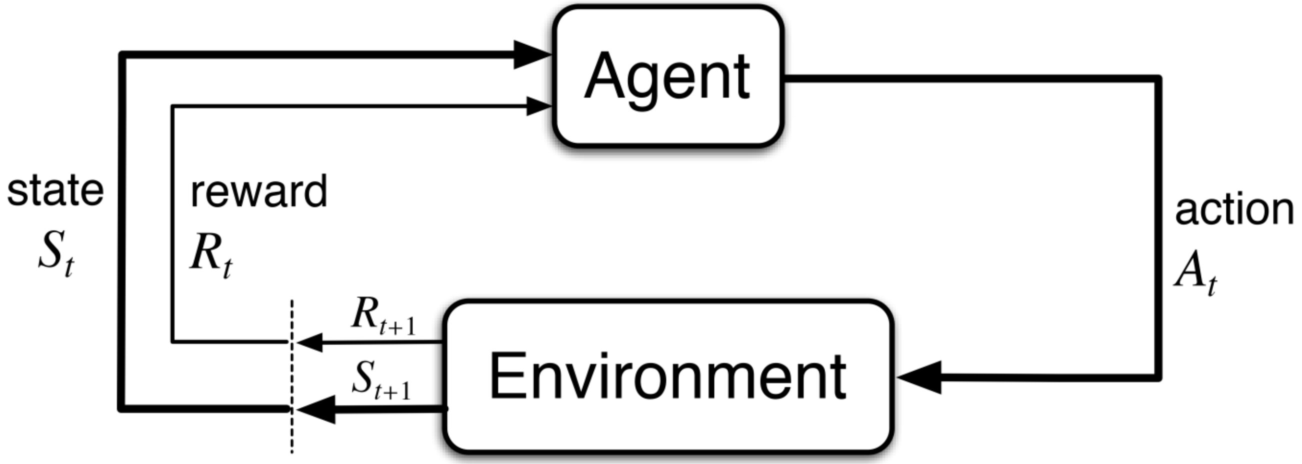






Slide Credits: Geoff Hulten

https://deepmind.com/documents/119/agz_unformatted_nature.pdf



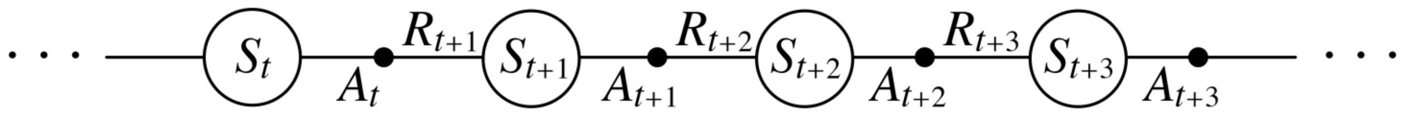
Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, \ldots$

Agent observes state at step t: $S_t \in \mathcal{S}$

produces action at step t : $A_t \in \mathcal{A}(S_t)$

gets resulting reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}^+$

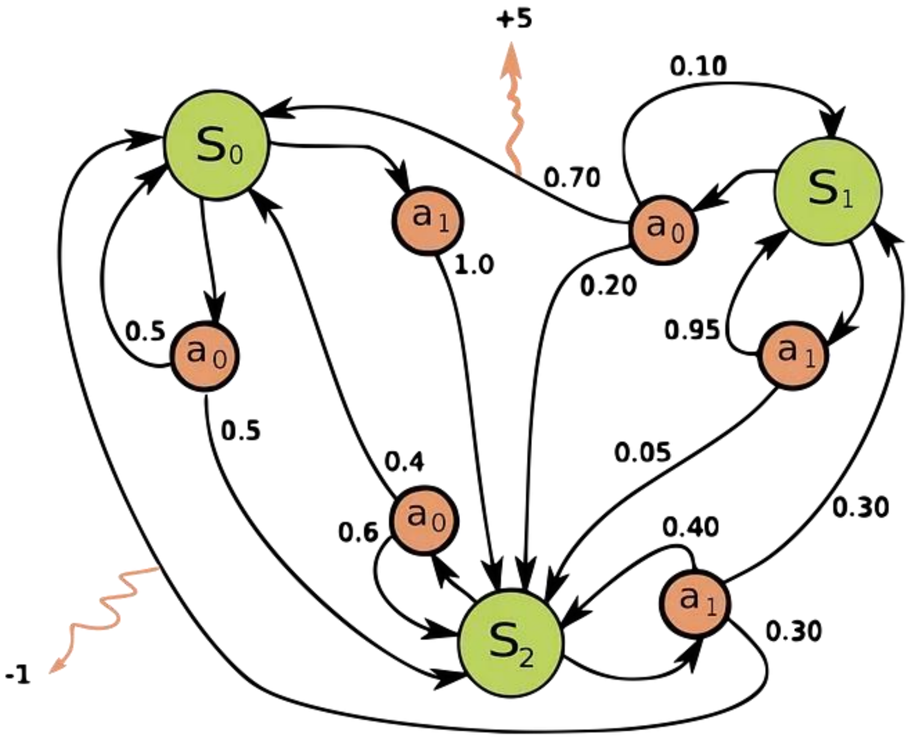


Slide Credits: Katerina Fragkiadaki

Reinforcement Learning

- RL problems are generally posed as Markov Decision Process (MDP)
  - RL is used for MDPs where the transition prob. or reward prob. are unknown.
- MDP: Discrete-Time Stochastic Control Process
  - Markovian Property:
    - Next reward and state does not depend on history.
    - Next reward and state depend only on current state and action.
  - It's a 4-tuple $\left(S, A_s, P_a(s, s'), R_a(s, s')\right)$
    - S → State Space → Set of states
    - A_s → Action Space available at state s → Set of possible actions
    - $P_a(s, s') = \mathbb{P}(s'|s, a)$ → Probability of transitioning from s to s′ after taking action a
    - $R_a(s, s') = \mathbb{E}(r'|s, a)$ → Expected Reward obtained after transitioning from s to s′ after taking action a

Example MDP



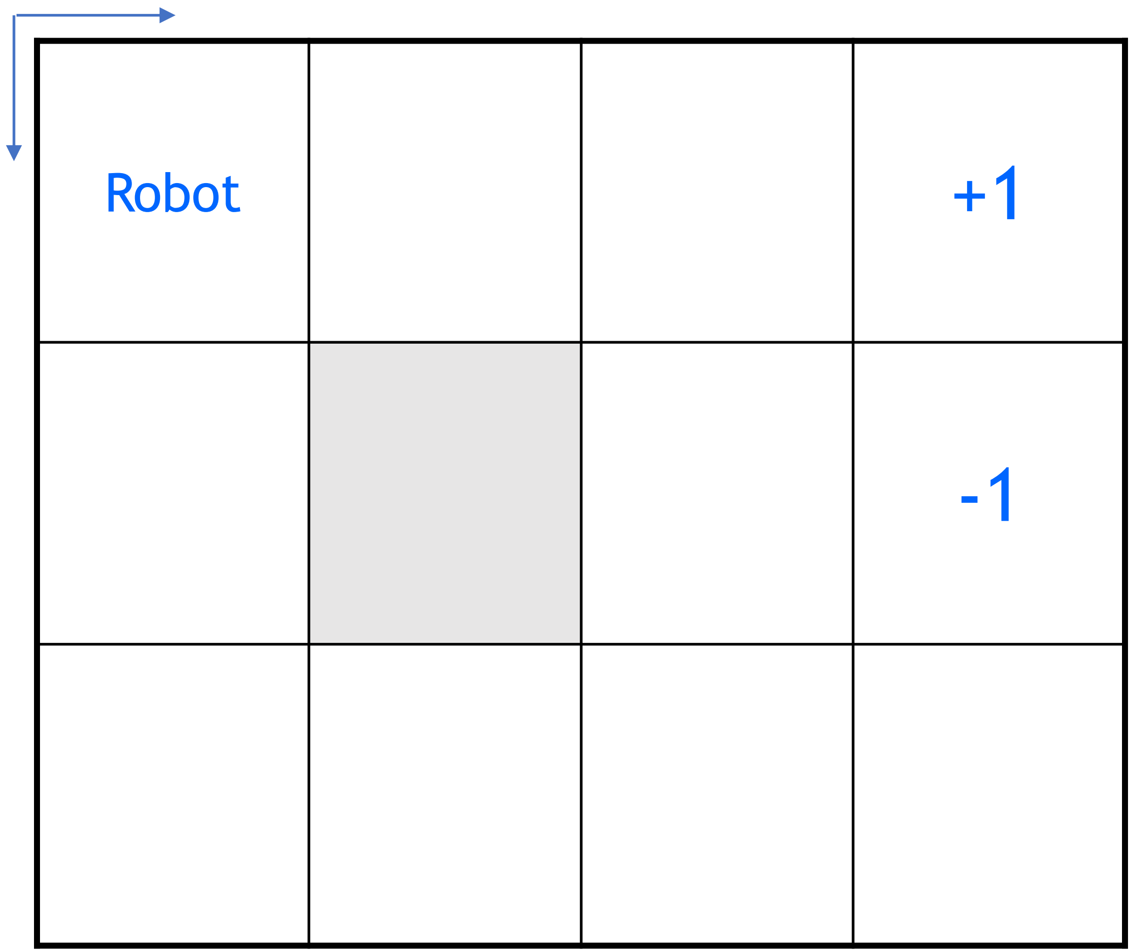
- 3 states (green circles)
  - $\{S_0, S_1, S_2\}$
- 2 actions (orange circles)
  - $\{a_0, a_1\}$
- 2 rewards (orange arrows)
  - $R_{a1}(S_2, S_0) = -1$
  - $R_{a0}(S_1, S_0) = +5$
- Example transition probabilities:
  - $P_{a0}(S_0, S_2) = 0.5$
  - $P_{a0}(S_0, S_0) = 0.5$

Image Credits: wikipedia.org

Reinforcement Learning

- Policy (π): Mapping from state to action
  - Deterministic: $a_t = \pi(s_t)$, or,
  - Stochastic: $a_t = argmax_a \, \pi(a|s_t)$
- Objective of RL:
  - Find a policy that maximizes long term cumulative reward.
  - maximize $\sum_{t=0}^{T} \gamma^t R_{a_t}(s_t, s_{t+1})$, where, $\gamma \in [0,1]$ (Discount factor)
- How to make a decision?
  - Rank State or (State, Action) based on some value derived from experience
  - Value functions measure the goodness of a particular state or state/action pair for a given Policy

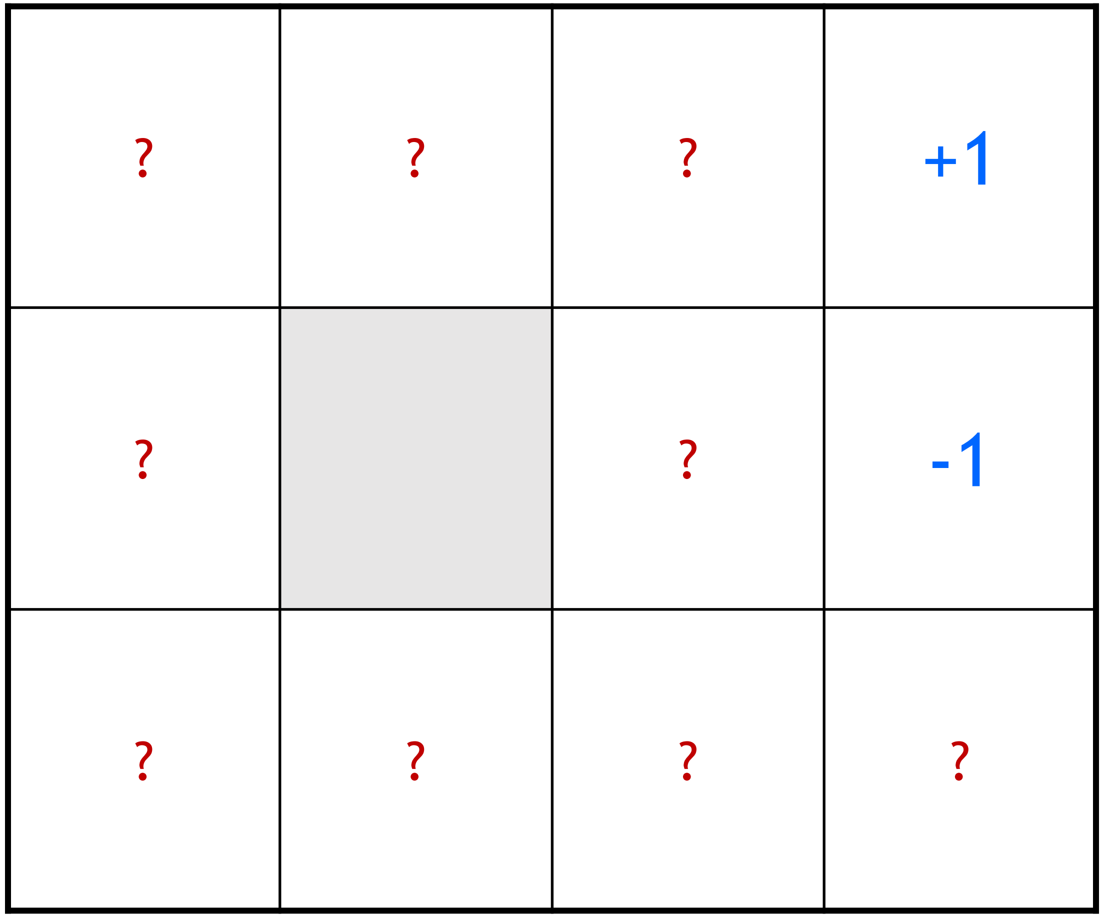
Robot in a room



- States:
  - Location $\in \{(1,1), \ldots, (3,4)\}$
- Actions:
  - UP, DOWN, LEFT, RIGHT
- Terminate at (1,4) or (2,4)
- Note:
  - Transitions and rewards are deterministic.
- Reward +1 at (1,4), -1 at (2,4)
- Reward -0.1 for each step

Slide based on works of **Peter Bodík,** RAD Lab, UC Berkeley

Robot in a room: State Value Function



Reward -0.1 for each step

- State Value Function:
  - $V(s)$
  - Maximum expected reward accumulated when starting from a given state
- Bellman equation (Optimal):
  - $V(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V(s')\right)\right)$
  - $\gamma = 1, s = s_t, s' = s_{t+1}$

Robot in a room: State Value Function

0	0	0	+1
0		0	-1
0	0	0	0

Reward -0.1 for each step

- State Value Function:
  - $V(s)$
  - Maximum expected reward accumulated when starting from a given state
- Bellman equation (Optimal):
  - $V(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V(s')\right)\right)$
  - $\gamma = 1, s = s_t, s' = s_{t+1}$
- Value Iteration
  - Initializing
  - $V_{k+1}(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V_k(s')\right)\right)$

Robot in a room: State Value Function

-0.1	-0.1	0.9	+1
-0.1		-0.1	-1
-0.1	-0.1	-0.1	-0.1

Reward -0.1 for each step

- State Value Function:
  - $V(s)$
  - Maximum expected reward accumulated when starting from a given state
- Bellman equation (Optimal):
  - $V(s) = \max_a \left(\mathbb{E}\left(R_a(s,s') + \gamma V(s')\right)\right)$
  - $\gamma = 1, s = s_t, s' = s_{t+1}$
- Using Bellman equation iteratively
  - $V_{k+1}(s) = \max_a \left(\mathbb{E}\left(R_a(s,s') + \gamma V_k(s')\right)\right)$
  - First Iteration

Robot in a room: State Value Function

-0.1	0.8	0.9	+1
-0.1		0.8	-1
-0.1	-0.1	-0.1	-0.1

Reward -0.1 for each step

- State Value Function:
  - $V(s)$
  - Maximum expected reward accumulated when starting from a given state
- Bellman equation (Optimal):
  - $V(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V(s')\right)\right)$
  - $\gamma = 1, s = s_t, s' = s_{t+1}$
- Using Bellman equation iteratively
  - $V_{k+1}(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V_k(s')\right)\right)$
  - Second Iteration

Robot in a room: State Value Function

0.7	0.8	0.9	+1
-0.1		0.8	-1
-0.1	-0.1	0.7	-0.1

Reward -0.1 for each step

- State Value Function:
  - $V(s)$
  - Maximum expected reward accumulated when starting from a given state
- Bellman equation (Optimal):
  - $V(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V(s') \right) \right)$
  - $\gamma = 1, s = s_t, s' = s_{t+1}$
- Using Bellman equation iteratively
  - $V_{k+1}(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V_k(s') \right) \right)$
  - Third Iteration

Robot in a room: State Value Function

0.7	0.8	0.9	+1
0.6		0.8	-1
-0.1	0.6	0.7	0.6

Reward -0.1 for each step

- State Value Function:
  - $V(s)$
  - Maximum expected reward accumulated when starting from a given state
- Bellman equation (Optimal):
  - $V(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V(s')\right)\right)$
  - $\gamma = 1, s = s_t, s' = s_{t+1}$
- Using Bellman equation iteratively
  - $V_{k+1}(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V_k(s')\right)\right)$
  - Fourth Iteration

Robot in a room: State Value Function

0.7	0.8	0.9	+1
0.6		0.8	-1
0.5	0.6	0.7	0.6

Reward -0.1 for each step

- State Value Function:
  - $V(s)$
  - Maximum expected reward accumulated when starting from a given state
- Bellman equation (Optimal):
  - $V(s) = \max_a \left(\mathbb{E}\left(R_a(s,s') + \gamma V(s')\right)\right)$
  - $\gamma = 1, s = s_t, s' = s_{t+1}$
- Using Bellman equation iteratively
  - $V_{k+1}(s) = \max_a \left(\mathbb{E}\left(R_a(s,s') + \gamma V_k(s')\right)\right)$
  - Fifth Iteration

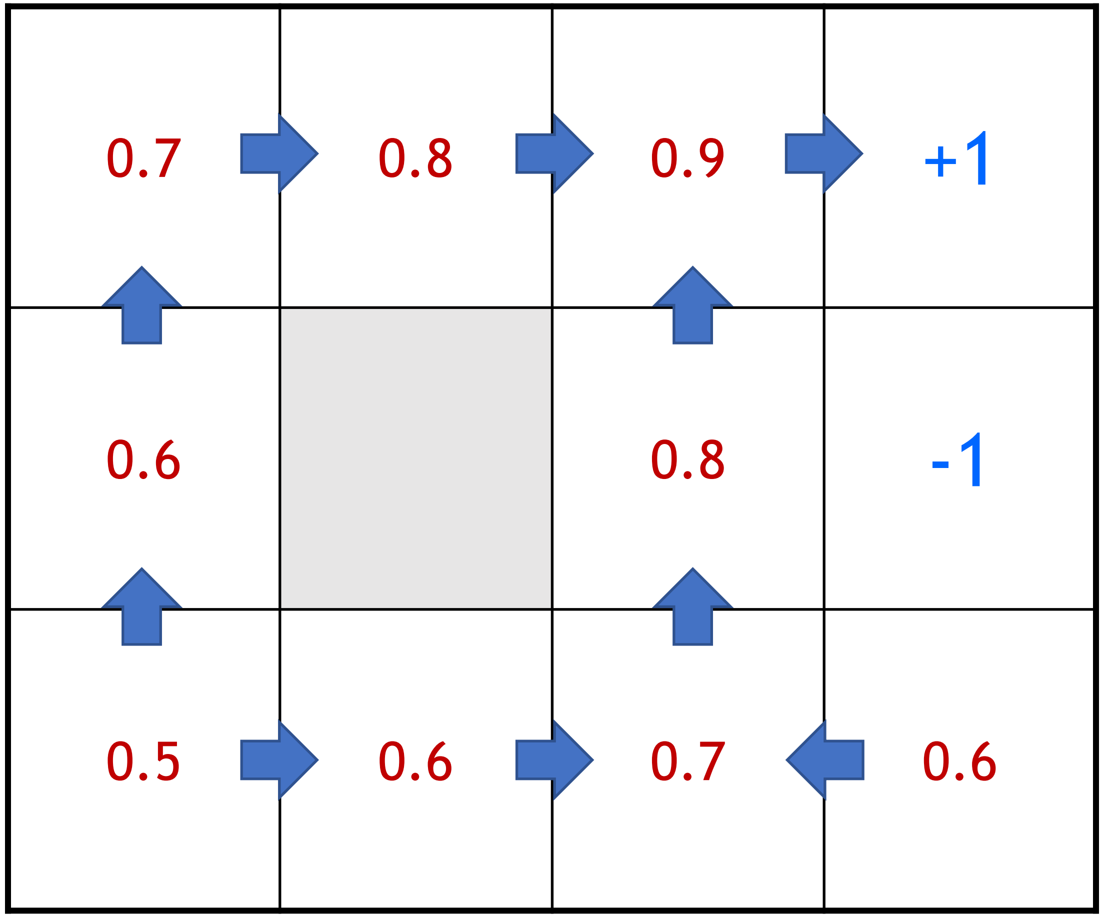
Robot in a room: State Value Function

0.7	0.8	0.9	+1
0.6		0.8	-1
0.5	0.6	0.7	0.6

Reward -0.1 for each step

- State Value Function:
  - $V(s)$
  - Maximum expected reward accumulated when starting from a given state
- Bellman equation (Optimal):
  - $V(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V(s')\right)\right)$
  - $\gamma = 1, s = s_t, s' = s_{t+1}$
- Using Bellman equation iteratively
  - $V_{k+1}(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V_k(s')\right)\right)$
  - Converged

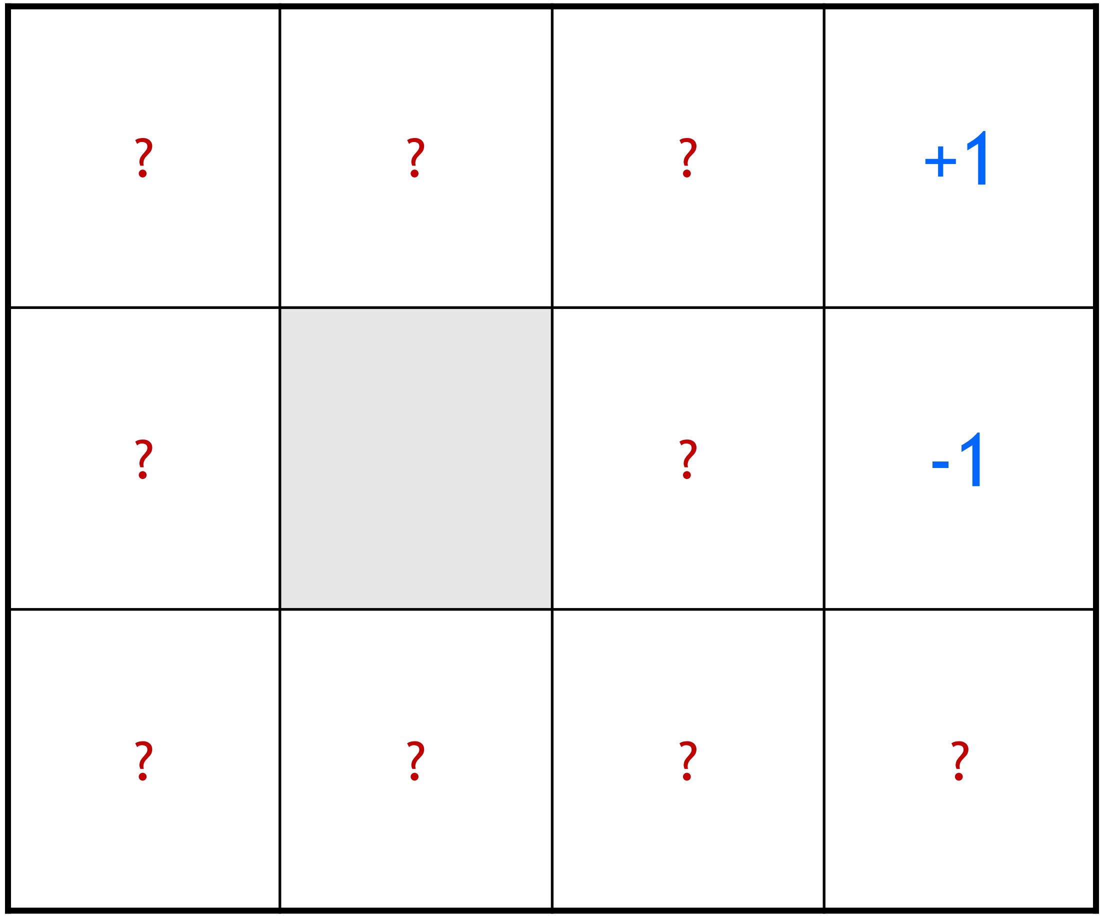
Robot in a room: State Value Function



Reward -0.1 for each step

- State Value Function:
  - $V(s)$
  - Maximum expected reward accumulated when starting from a given state
- Bellman equation (Optimal):
  - $V(s) = \max_a \left(\mathbb{E}\left(R_a(s, s') + \gamma V(s')\right)\right)$
  - $\gamma = 1, s = s_t, s' = s_{t+1}$
- Policy:
  - $\pi(s) = \operatorname{argmax}_a \sum P_a(s, s') V(s')$

Robot in a room:

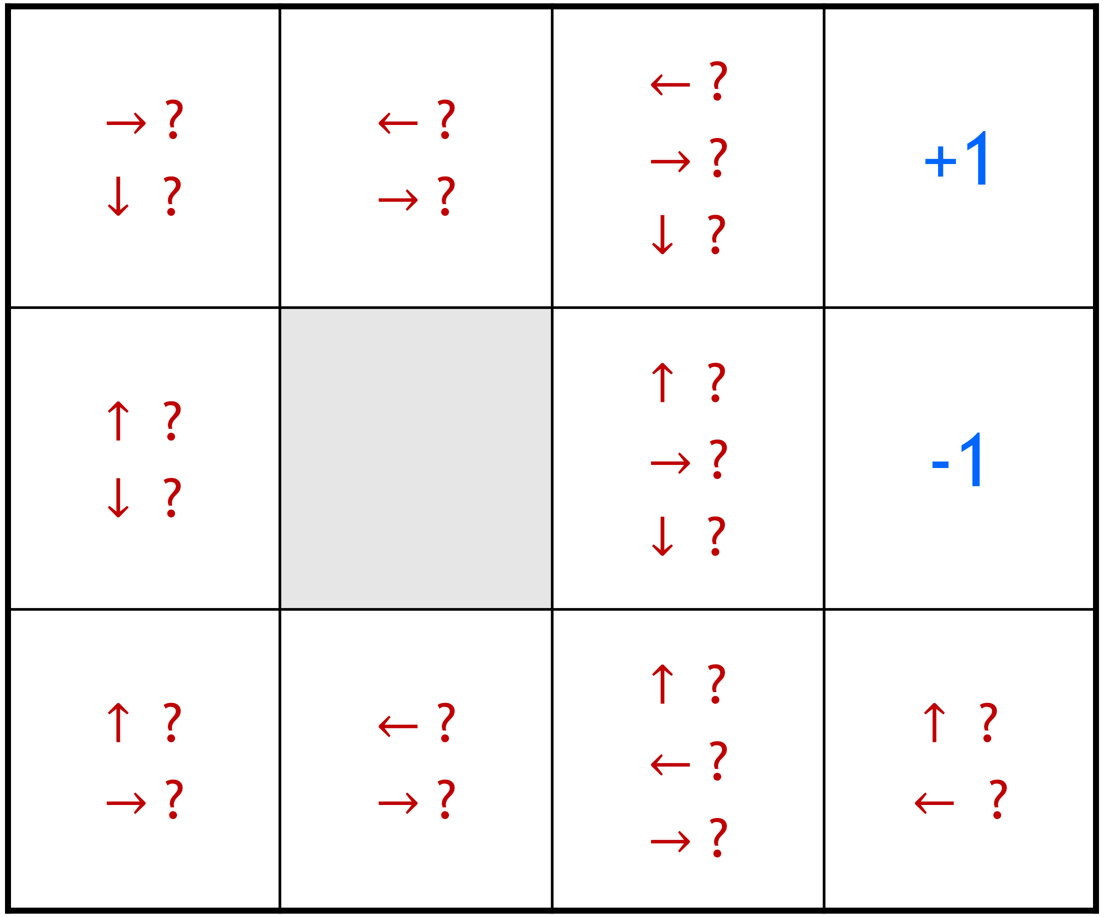


Reward -0.1 for each step

What if the robot is not functioning properly?

- State transitions are stochastic
  - An action may not lead to intended state
- Rewards/Costs are stochastic

Robot in a room: State-Action Value Function



Reward -0.1 for each step

- State-Action Value Function:
  - $Q(s, a)$
  - Maximum expected reward accumulated when starting from a given state and choosing a given action.
  - $V(s) = \max_a Q(s, a)$
- Bellman equation (Optimal):

$$Q(s,a) = \mathbb{E}\left(R_a(s,s') + \gamma \max_{a'} Q(s',a')\right)$$

$$\gamma = 1 \ , \ s = s_t,\ s' = s_{t+1},\ a' = a_{t+1}$$

Robot in a room: State-Action Value Function

→ 0 ↓ 0	← 0 → 0	← 0 → 0 ↓ 0	+1
↑ 0 ↓ 0		↑ 0 → 0 ↓ 0	-1
↑ 0 → 0	← 0 → 0	↑ 0 ← 0 → 0	↑ 0 ← 0

Reward -0.1 for each step

- State-Action Value Function:
  - $Q(s,a)$
  - Maximum expected reward accumulated when starting from a given state and choosing a given action.
  - $V(s) = \max_a Q(s,a)$
- Bellman equation (Optimal):

$$Q(s,a) = \mathbb{E}\left(R_a(s,s') + \gamma \max_{a'} Q(s',a')\right)$$

$$\gamma = 1 \ , \ s = s_t, \ s' = s_{t+1}, \ a' = a_{t+1}$$

- Value Iteration: Initialization

Robot in a room: State-Action Value Function

→ -0.1 ↓ -0.1	← -0.1 → -0.1	← -0.1 → 0.9 ↓ -0.1	+1
↑ -0.1 ↓ -0.1		↑ -0.1 → -1.1 ↓ -0.1	-1
↑ -0.1 → -0.1	← -0.1 → -0.1	↑ -0.1 ← -0.1 → -0.1	↑ -1.1 ← -0.1

Reward -0.1 for each step

- State-Action Value Function:
  - $Q(s, a)$
  - Maximum expected reward accumulated when starting from a given state and choosing a given action.
  - $V(s) = \max_a Q(s, a)$
- Bellman equation (Optimal):

$$Q(s, a) = \mathbb{E}\left(R_a(s, s') + \gamma \max_{a'} Q(s', a')\right)$$

$$\gamma = 1 \; , \; s = s_t, \; s' = s_{t+1}, \; a' = a_{t+1}$$

- Value Iteration: First Iteration

Robot in a room: State-Action Value Function

→ -0.2 ↓ -0.2	← -0.2 → 0.8	← -0.2 → 0.9 ↓ -0.2	+1
↑ -0.2 ↓ -0.2		↑ 0.8 → -1.1 ↓ -0.2	-1
↑ -0.2 → -0.2	← -0.2 → -0.2	↑ -0.2 ← -0.2 → -0.2	↑ -1.1 ← -0.2

Reward -0.1 for each step

- State-Action Value Function:
  - $Q(s,a)$
  - Maximum expected reward accumulated when starting from a given state and choosing a given action.
  - $V(s) = \max_a Q(s,a)$
- Bellman equation (Optimal):

  $$Q(s,a) = \mathbb{E}\left(R_a(s,s') + \gamma \max_{a'} Q(s',a')\right)$$

  $$\gamma = 1\ ,\ s = s_t,\ s' = s_{t+1},\ a' = a_{t+1}$$
- Value Iteration: Second Iteration

Robot in a room: State-Action Value Function

→ 0.7 ↓ -0.3	← -0.3 → 0.8	← 0.7 → 0.9 ↓ 0.7	+1
↑ -0.3 ↓ -0.3		↑ 0.8 → -1.1 ↓ -0.3	-1
↑ -0.3 → -0.3	← -0.3 → -0.3	↑ 0.7 ← -0.3 → -0.3	↑ -1.1 ← -0.3

Reward -0.1 for each step

- State-Action Value Function:
  - $Q(s,a)$
  - Maximum expected reward accumulated when starting from a given state and choosing a given action.
  - $V(s) = \max_a Q(s,a)$
- Bellman equation (Optimal):

  $$Q(s,a) = \mathbb{E}\left(R_a(s,s') + \gamma \max_{a'} Q(s',a')\right)$$

  $\gamma = 1$, $s = s_t$, $s' = s_{t+1}$, $a' = a_{t+1}$
- Value Iteration: Third Iteration

Robot in a room: State-Action Value Function

→ 0.7 ↓ 0.5	← 0.6 → 0.8	← 0.7 → 0.9 ↓ 0.7	+1
↑ 0.6 ↓ 0.4		↑ 0.8 → -1.1 ↓ 0.6	-1
↑ 0.5 → 0.5	← 0.4 → 0.6	↑ 0.7 ← 0.5 → 0.5	↑ -1.1 ← 0.6

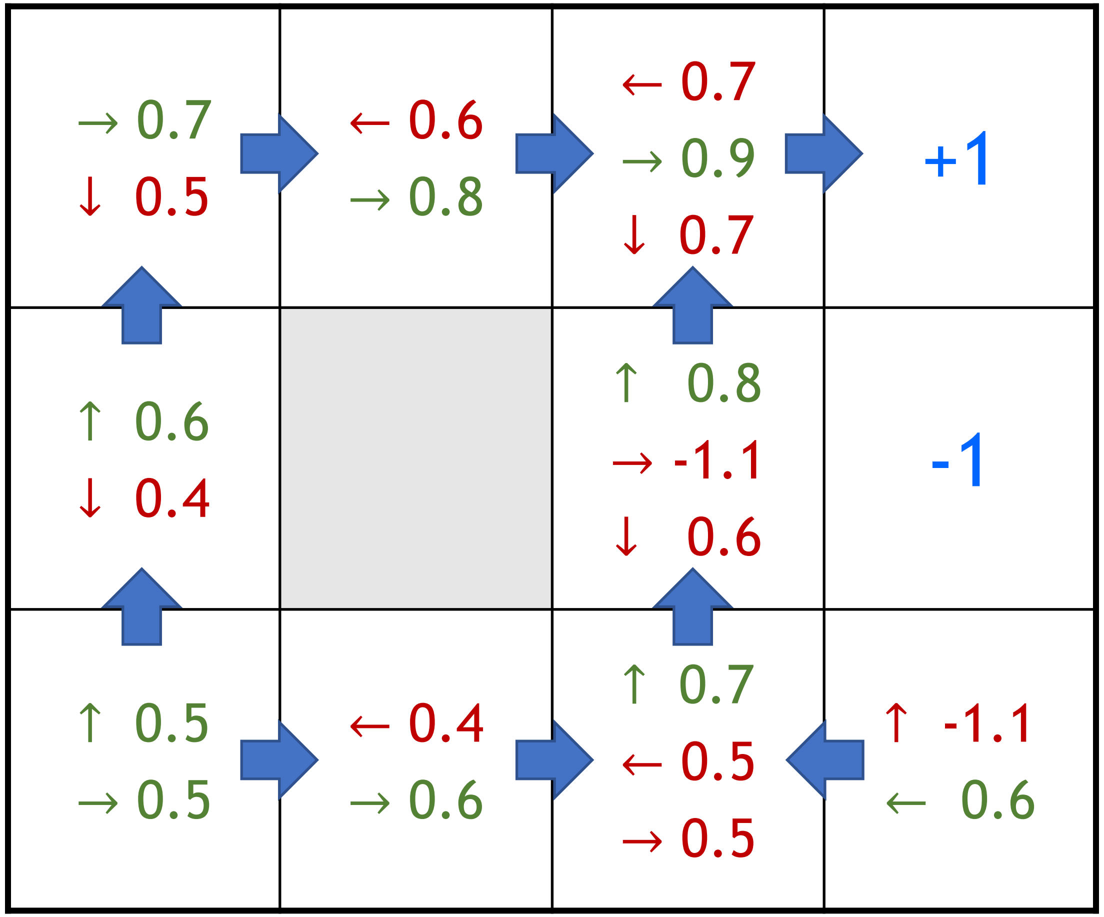
Reward -0.1 for each step

- State-Action Value Function:
  - $Q(s,a)$
  - Maximum expected reward accumulated when starting from a given state and choosing a given action.
  - $V(s) = \max_a Q(s,a)$
- Bellman equation (Optimal):

$$Q(s,a) = \mathbb{E}\left(R_a(s,s') + \gamma \max_{a'} Q(s',a')\right)$$

$$\gamma = 1\ ,\ s = s_t,\ s' = s_{t+1},\ a' = a_{t+1}$$

Robot in a room: State-Action Value Function

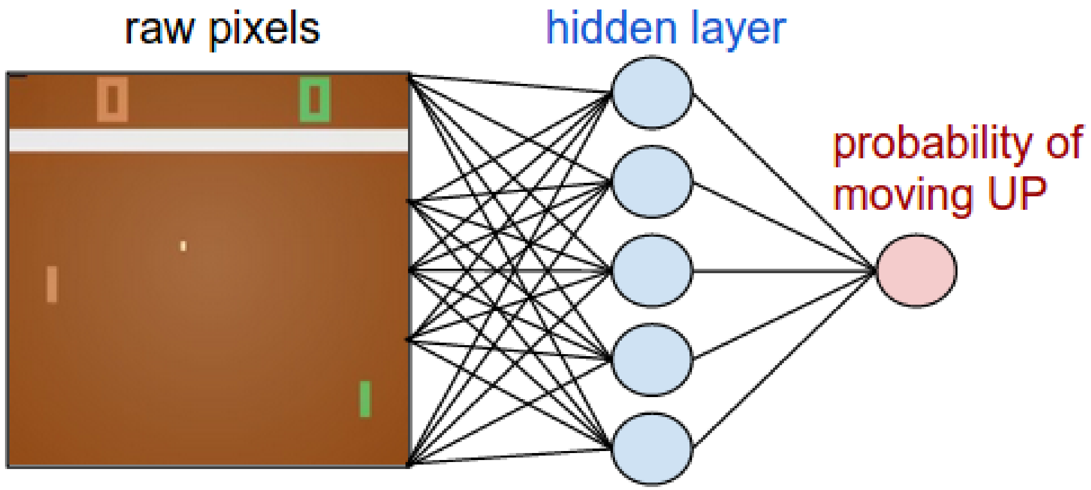


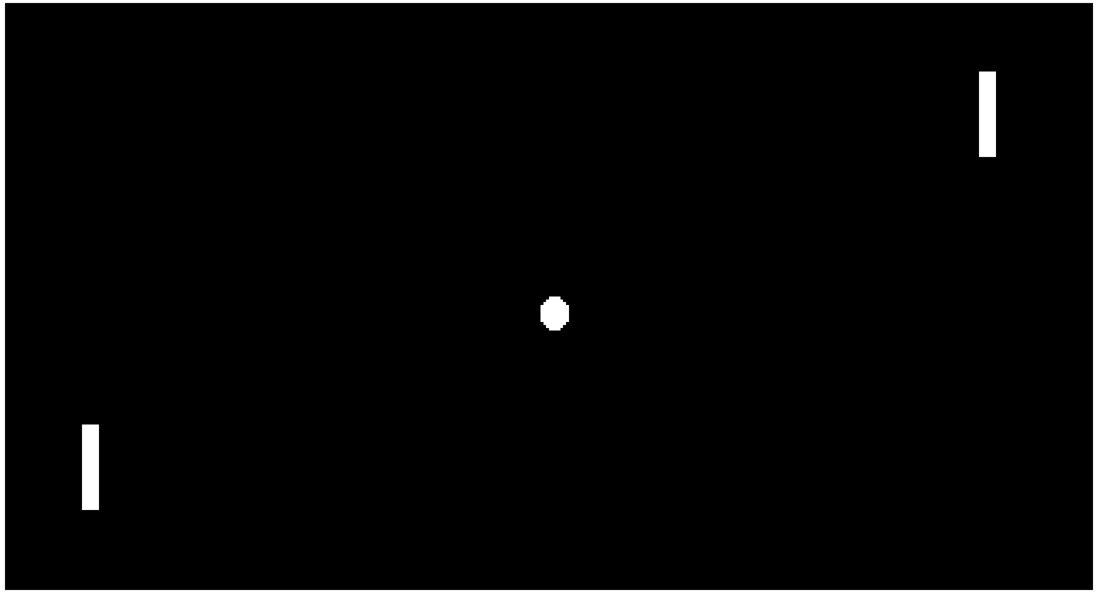
Reward -0.1 for each step

- State-Action Value Function:
  - $Q(s, a)$
  - Maximum expected reward accumulated when starting from a given state and choosing a given action.
  - $V(s) = \max_a Q(s, a)$
- Policy:
  - $\pi(s) = argmax_a Q(s, a)$

Policy Gradient Method

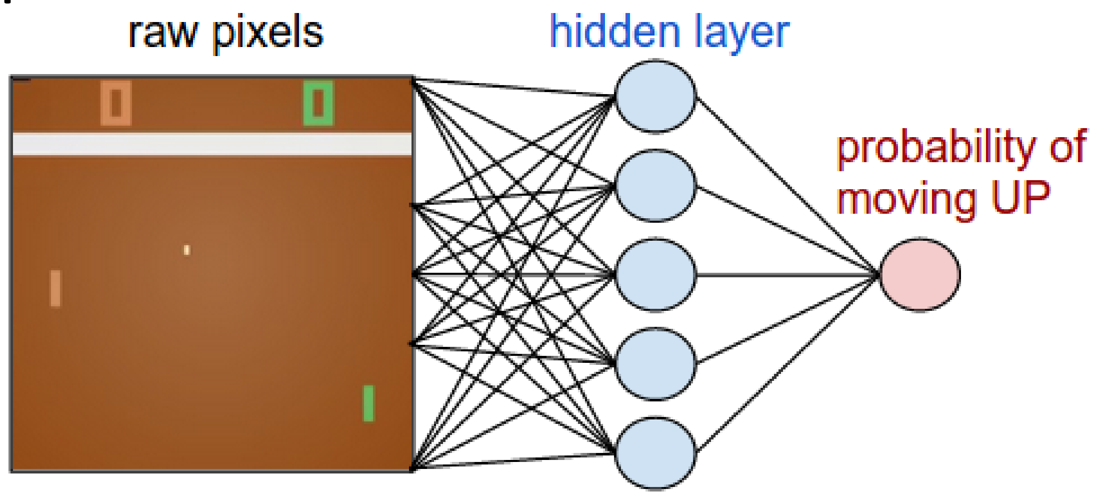
- Policy Gradient:
  - learn policy directly $\pi(a|s)$





Example from: https://www.youtube.com/watch?v=tqrcjHuNdmQ

Policy Gradient Method



- Policy Gradient:
  - learn policy directly $\pi(a|s)$

- REINFORCE Algorithm

Initialize θ arbitrarily

for each episode $\{s_0, a_0, r_0, \ldots, s_T, a_T, r_T\} \sim \pi(\cdot, \cdot; \theta)$

do $\begin{cases} \textbf{for } t \leftarrow 0 \textbf{ to } T \\ \textbf{do} \begin{cases} G \leftarrow \sum_{k=t}^{T} \gamma^{k-t} \cdot r_k \\ \theta \leftarrow \theta + \alpha \cdot \gamma^t \cdot \nabla_\theta \log \pi(s_t, a_t; \theta) \cdot G \end{cases} \end{cases}$

return (θ)

References

- "What is reinforcement learning?," What Is Reinforcement Learning? - MATLAB & Simulink, https://www.mathworks.com/discovery/reinforcement-learning.html (accessed Jul. 10, 2023).
- S. Brown, "Machine Learning, explained," MIT Sloan, https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained (accessed Jul. 10, 2023).
- "Cluster analysis," Wikipedia, https://en.wikipedia.org/wiki/Cluster_analysis (accessed Jul. 10, 2023).
- "Regression vs classification in Machine Learning - Javatpoint," www.javatpoint.com, https://www.javatpoint.com/regression-vs-classification-in-machine-learning (accessed Jul. 10, 2023).
- "Reinforcement learning," Wikipedia, https://en.wikipedia.org/wiki/Reinforcement_learning (accessed Jul. 10, 2023).
- G. Hulten, "CSEP546: Machine learning," University of Washington, https://courses.cs.washington.edu/courses/csep546/18au/ (accessed Jul. 10, 2023).

References

- K. Fragkiadaki and T. Mitchell, “CMU 10703: Deep RL and Control,” Carnegie Mellon University, https://www.andrew.cmu.edu/course/10-703/ (accessed Jul. 10, 2023).
- “Markov decision process,” Wikipedia, https://en.wikipedia.org/wiki/Markov_decision_process (accessed Jul. 10, 2023).
- P. Bodik, “Practical machine learning lecture: Reinforcement learning,” University of California, Berkeley, http://people.eecs.berkeley.edu/~jordan/courses/294-fall09/lectures/reinforcement/ (accessed Jul. 10, 2023).
- A. Karpathy et al., “Deep RL Bootcamp,” Deep RL Bootcamp - Berkeley CA, https://sites.google.com/view/deep-rl-bootcamp/lectures (accessed Jul. 10, 2023).
- R. S. Sutton and A. Barto, Reinforcement Learning: An Introduction. Cambridge, Massachusetts ; London, England: The MIT Press, 2020.

Thankyou

Appendices

A1. Optimal State-Value Function

- Value function for arbitrary π

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t | S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big]
\end{aligned}
$$

- Optimal value function

$$
\begin{aligned}
v_*(s) &\doteq \max_\pi v_\pi(s) \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_*(s')\Big]
\end{aligned}
$$

- Return

$$
\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\
&= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

A2. Optimal (State, Action)-Value Function

- Q function for arbitrary π

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(a',s')\Big]$$

- Optimal Q function

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a)$$
$$= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') | S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big]$$

- Return

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$$
$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$
$$= R_{t+1} + \gamma G_{t+1}$$

A3. Value iteration

Params: θ - a small positive threshold determining the accuracy of the estimation

Initialize V(s), for all $s \in \mathcal{S}^+$ arbitrarily, except V(terminal)

$\Delta \leftarrow 0$

while $\Delta \geq \theta$ **do**

  foreach $s \in S$ **do**

    $v \leftarrow V(s)$

    $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$

    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

  end

end

output: Deterministic policy $\pi \approx \pi_*$ such that

$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$