ME5406 Deep Learning for Robotics

Part I
Tabular Methods for Reinforcement Learning

Dr. Peter C. Y. CHEN
Associate Professor
Department of Mechanical Engineering
College of Design and Engineering
National University of Singapore
Email: mpechenp@nus.edu.sg
https://sites.google.com/view/peter-chen/home

Part 1
Conceptual Foundation (Tabular Methods)

Lecturer: Peter C.Y. Chen

1. Intro. to Reinforcement Learning
2. Markov Decision Processes
3. Dynamic Programming
4. Monte-Carlo Methods
5. Temporal Difference Learning (SARSA, Q-learning)
6. Intro. to Deep-Q-Network (MLP and backpropagation)

Part 2
Approximate Methods (Deep-RL)

Lecturer: Guillaume A. Sartoretti

1. Robotic Perception
2. Deep Q-Learning and Variants
3. Policy Gradient Methods
4. Imitation Learning
5. Motion and Path Planning
6. Articulated Robotics
7. Multi-Agent Reinforcement Learning

For Part 1

- This set of lecture slides
- A set of lecture notes

For Part 2

- Please consult lecturer

References

- Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, 2nd ed., 2018
- Artificial Intelligence: A Modern Approach, Stuart Russell and Peter Norvig, 4th ed., 2020
- Other references will be announced in class or on CANVAS

- **Understand different RL approaches and applicability**
  - When the environment is/isn't known
  - When learning from complete/incomplete rollouts
  - When using exact/approximate representations
- **Discover challenges associated with use of RL**
  for a variety of textbooks/real-world (robotics) problems
- **Learn to implement your own RL solutions**
  - to tune and understand the relevant hyperparameters
  - to test the learning performance (using Python3 and tensorflow/keras, etc.)

100% Continuous Assessment

Two projects:

- One for each part
- 50% each for final grade
- Require Python coding
- Graded based on submitted code and individual project report

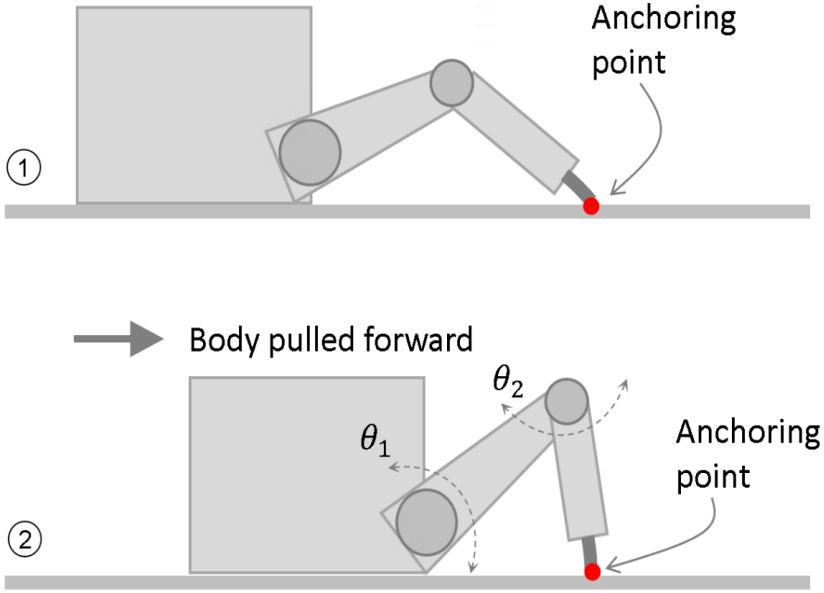
Students are expected to gain Python skills by independent study

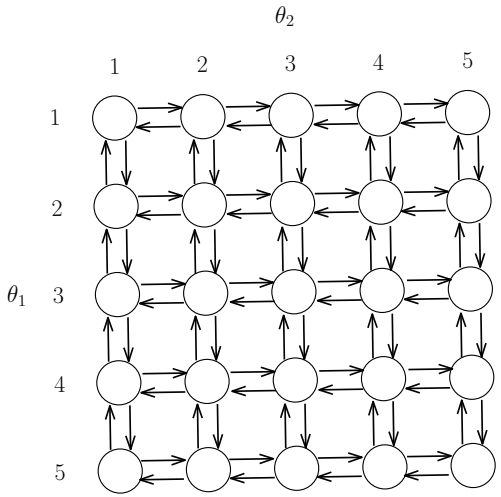
Example: Single-Agent Path Planning (SIPP) DRL
https://github.com/marmotlab/ME5406_exampleSAPP

Lecture	*Week*	*Date*	*Topic*
1	1	12/Jan	Ch 1: Introduction
2	2	19/Jan	Ch 2: Markov Decision Processes
3	3	26/Jan	Ch 3: Dynamic Programming
4	4	02/Feb	Ch 4: Monte-Carlo Methods
5	5	09/Feb	Ch 5: Temporal Difference Methods
–	6	16/Feb	No lecture (Chinese New Year's Eve)
–	6	**20/Feb**	**Project 1 due**
6	7	02/Mar	Ch 6: Intro. to Deep Reinforcement Learning

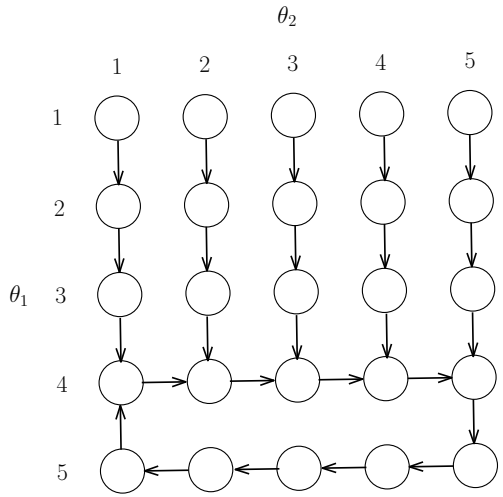
What is Reinforcement Learning

Tokic, M., Ertel, W., and Fessler, J., The crawler, a class room demonstrator for reinforcement learning. Proc. of the 22 International Florida Artificial Intelligence Research Society Conference, 160-165, 2009. Video link on YouTube: https://www.youtube.com/watch?v=2VjNTQ5cGzM



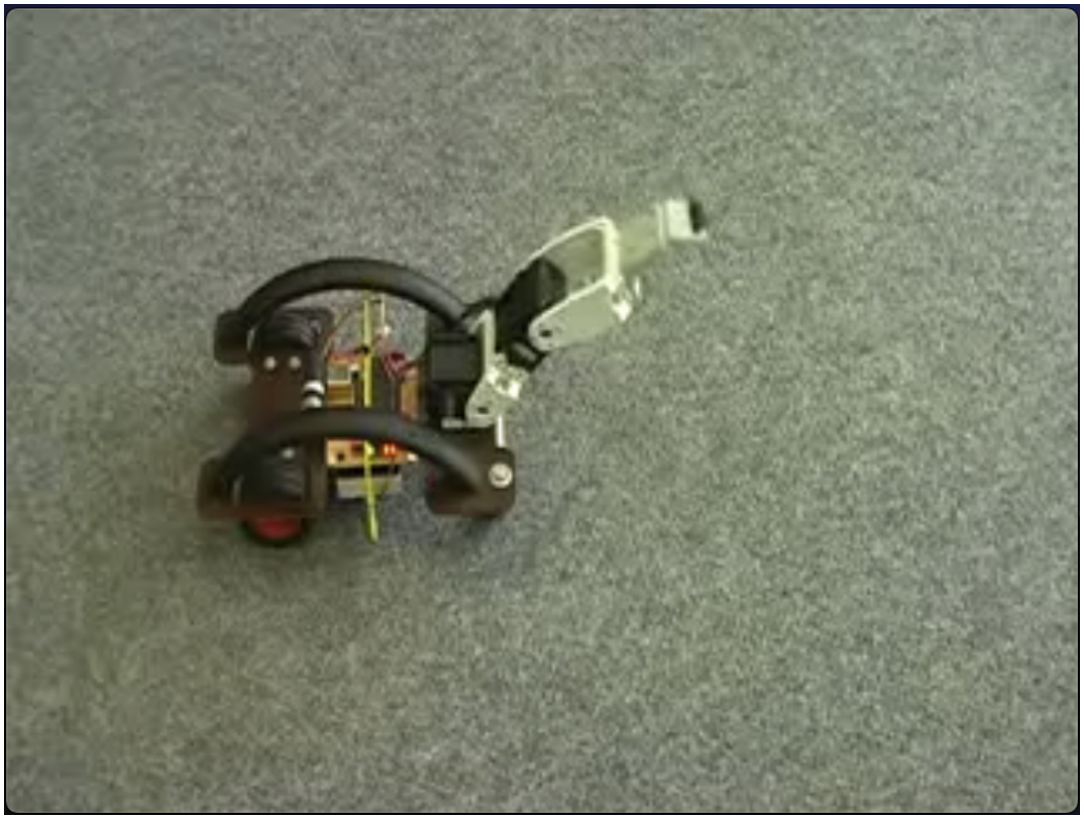


Start of learning



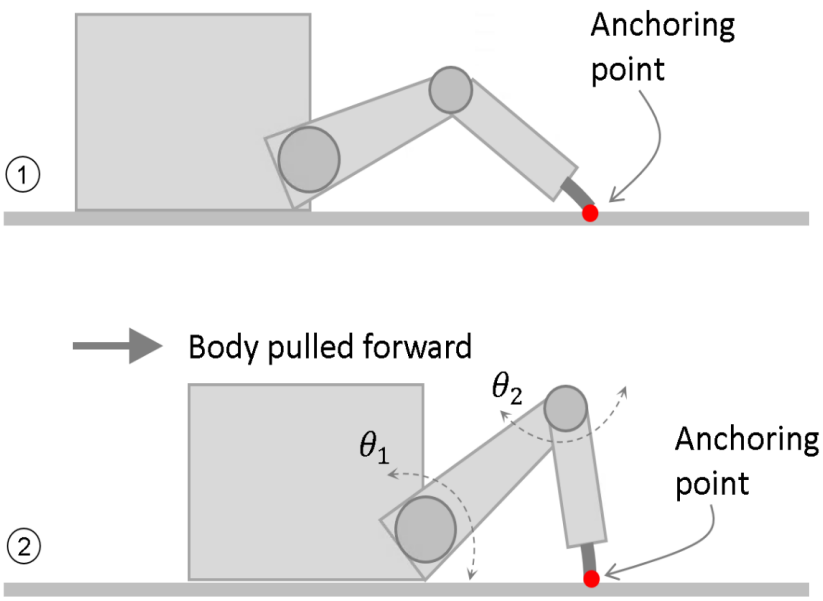
End of learning

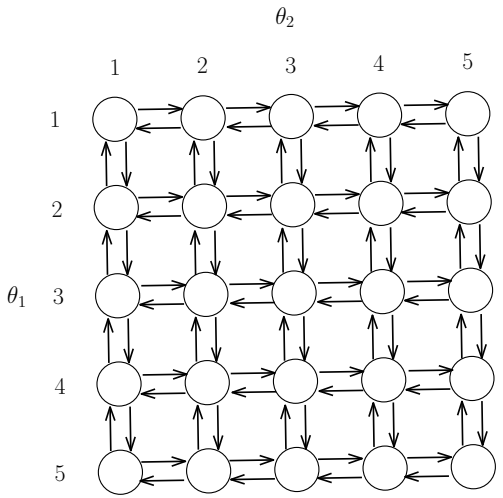
Video



https://www.youtube.com/watch?v=2VjNTQ5cGzM

Tokic, M., Ertel, W., and Fessler, J., The crawler, a class room demonstrator for reinforcement learning. Proc. of the 22 International Florida Artificial Intelligence Research Society Conference, 160-165, 2009. Video link on YouTube: https://www.youtube.com/watch?v=2VjNTQ5cGzM



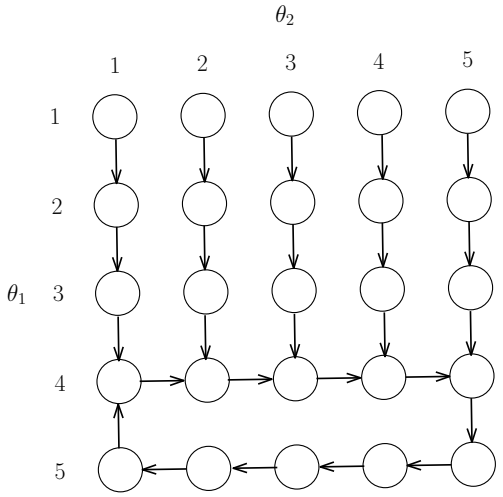


Start of learning

- $(\theta_1, \theta_2) \Rightarrow$ State s
- Change θ_1 or $\theta_2 \Rightarrow$ Action a

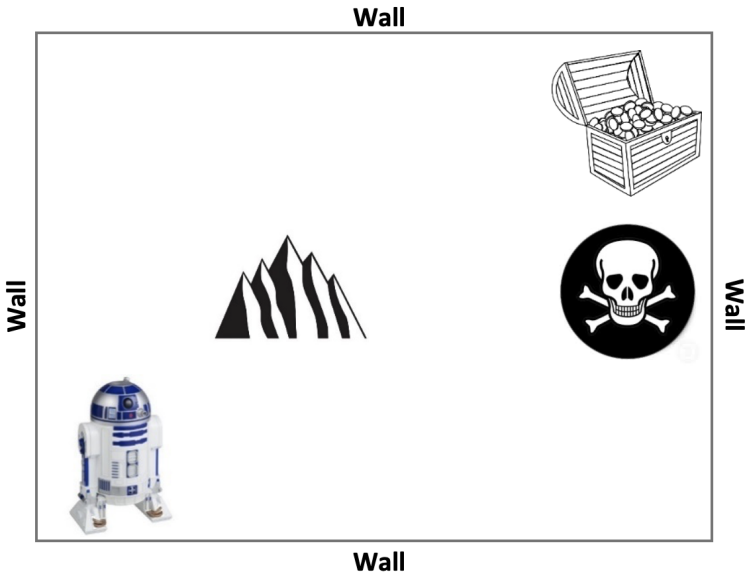
Learning

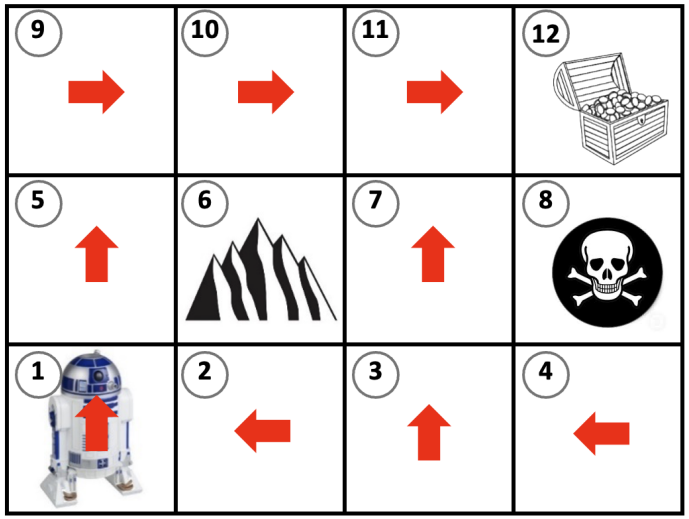
- **Decide** on an action $\Rightarrow$ Policy π
- Measure speed $\Rightarrow$ Reward r
- **Evaluate** decision made at s
- Repeat



End of learning

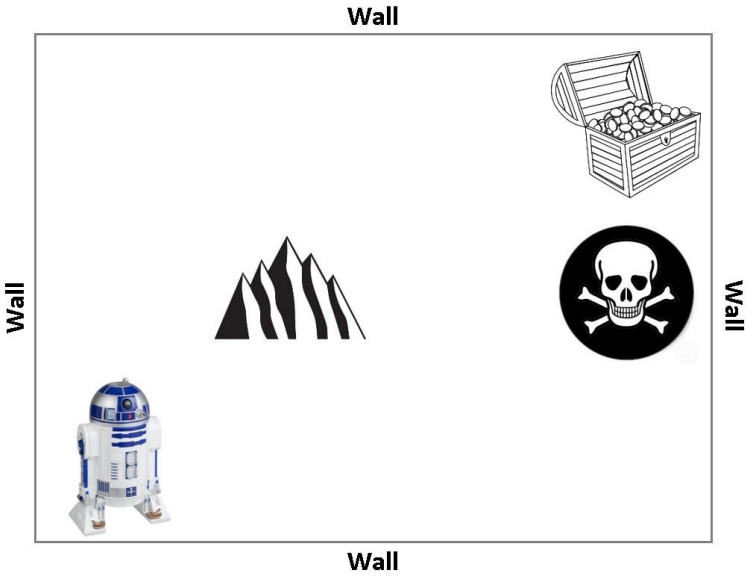
Robot to learn the "best" action to take at each state

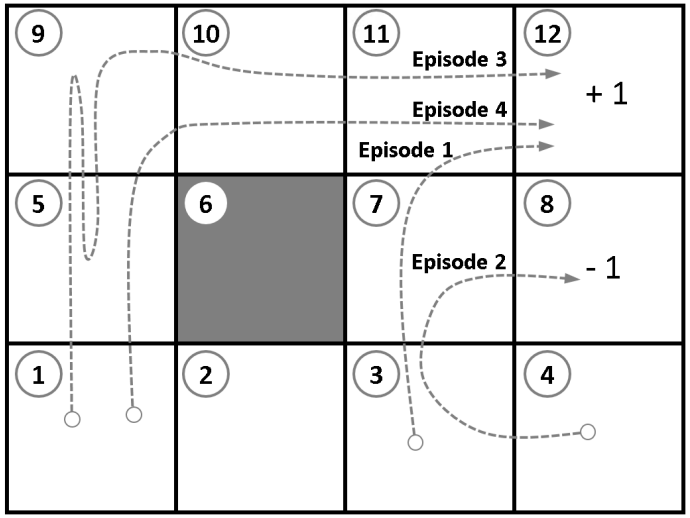




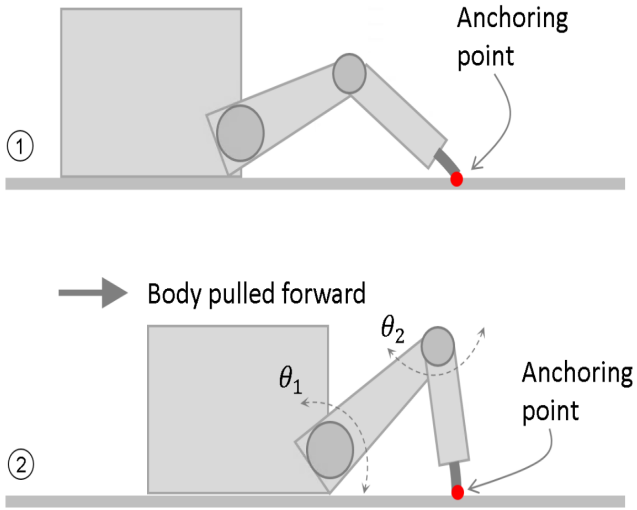
This is called an *optimal policy*

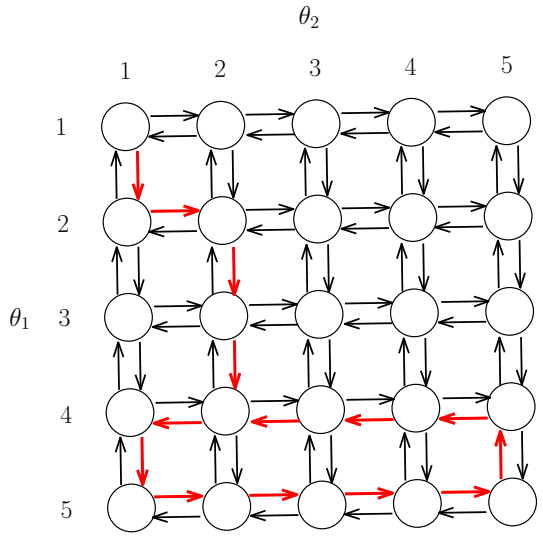
Episodic

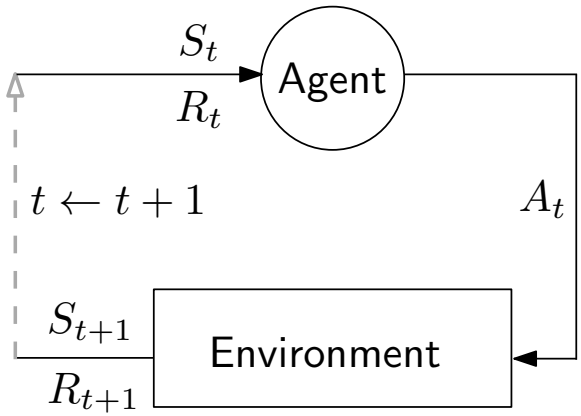




Continuing



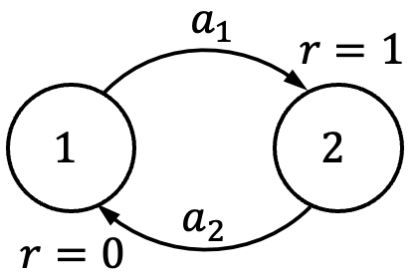




t	Time step
S_t, s	State where agent is in
A_t, a	Action taken by agent
R_t, r	Reward received by agent

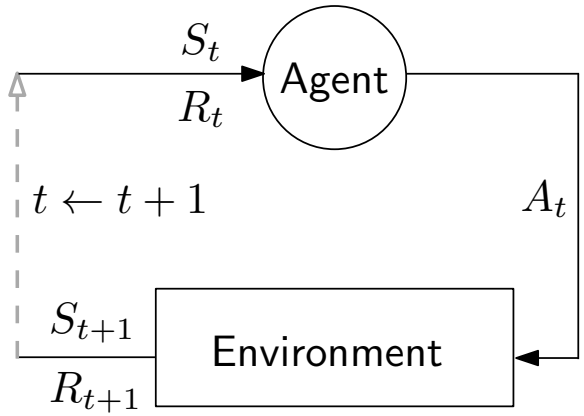
Markov Decision Processes (MDP)

- A set of states $\mathcal{S}$
- A set of actions $\mathcal{A}$
- A transition function, e.g., $f(s,a) = s'$
- A reward function, e.g., $\rho(s,a,s') = r$



Example $\mathcal{S} = \{1,2\}$ $\quad f(1,a_1) = 2$ $\quad \rho(1,a_1,2) = 1$

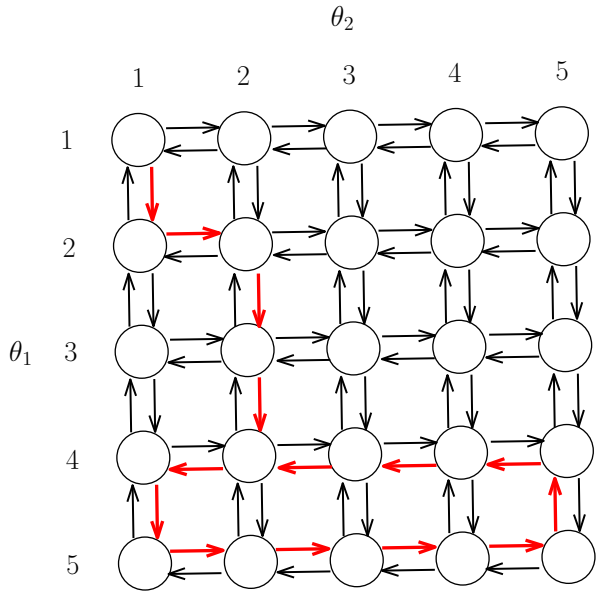
$\mathcal{A} = \{a_1, a_2\}$ $\quad f(2,a_2) = 1$ $\quad \rho(2,a_2,1) = 0$



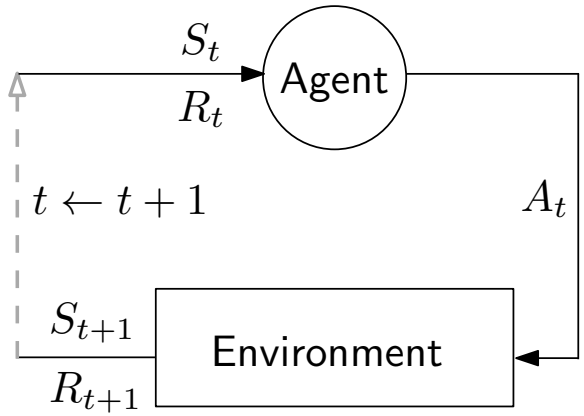
t	Time step
S_t, s	State where agent is in
A_t, a	Action taken by agent
R_t, r	Reward received by agent

Dynamics: At time step t, agent

- reaches S_t
- receives R_t
- takes action A_t
- $t \leftarrow t+1$



How does agent decide which action to take? $\Rightarrow$ Policy

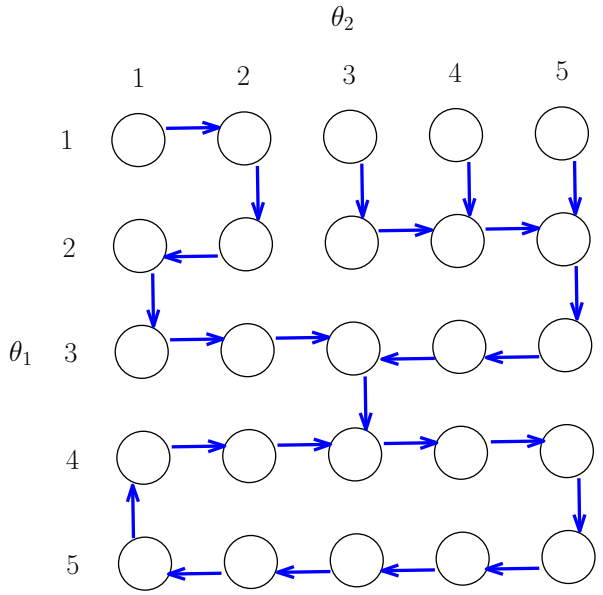


t	Time step
S_t, s	State where agent is in
A_t, a	Action taken by agent
R_t, r	Reward received by agent

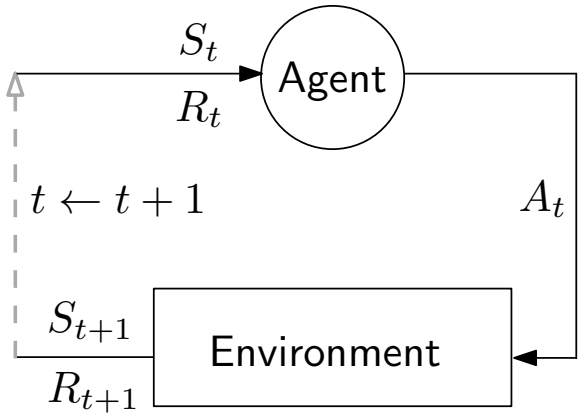
Policy π

A policy is a rule that decides the action the agent is to take at any given state

$$\pi(s) = a$$



A policy for robot-crawler problem



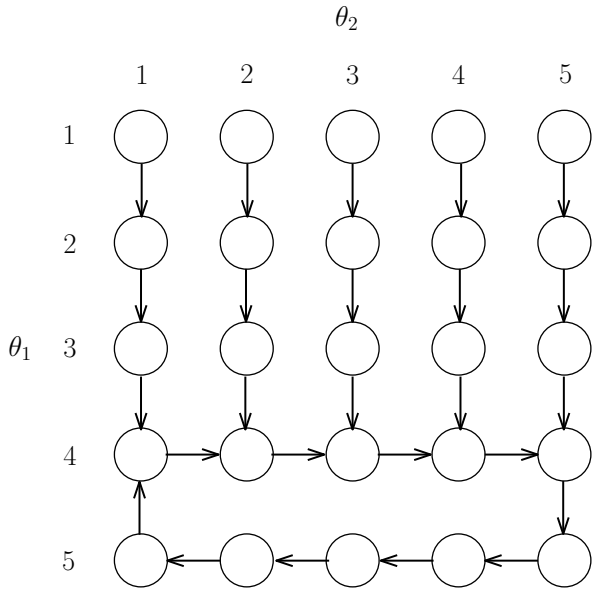
t	Time step
S_t, s	State where agent is in
A_t, a	Action taken by agent
R_t, r	Reward received by agent

After each time step, agent receives a reward

Reinforcement Learning Problem

Find policy $\pi = \pi_*$ that maximizes the 'total reward' (discussed later)

π_* is called the *optimal policy*



Optimal policy for robot-crawler problem

Solving RL problem requires notion of *value function*

For a given reward function $\rho(s, a, s') = r$, each state s or state-action pair (s, a) has a 'worth' calculated based on the rewards

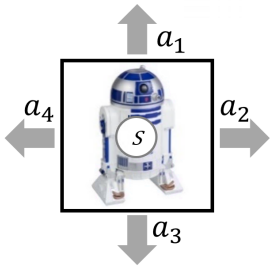
State value function

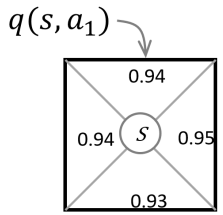
- Denoted by $v(s)$
- 'Worth' of state s

Action value function

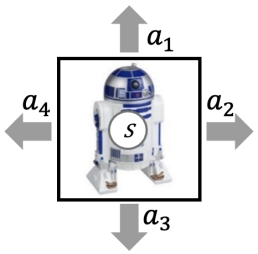
- Denoted by $q(s, a)$
- 'Worth' of taking a at s

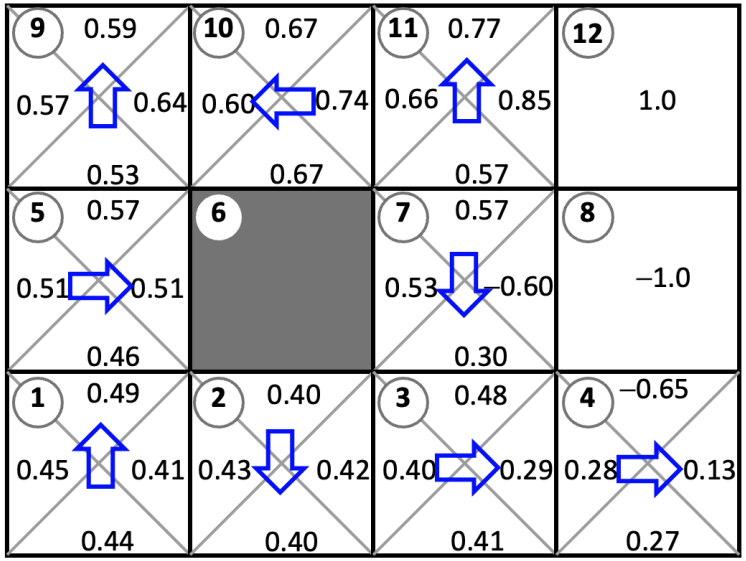
$q(s, a)$ and $v(s)$ are related to 'total reward' (discussed later)





Each π is associated with a set of $v_\pi(s)$ or $q_\pi(s,a)$ values





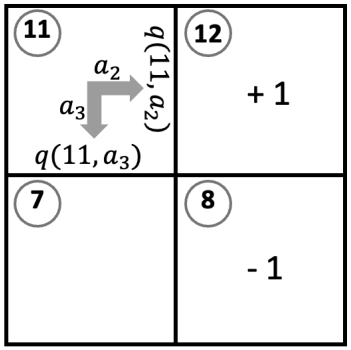
Blue arrows indicate policy π

$$q_\pi(1, a_1) = 0.49$$
$$q_\pi(2, a_3) = 0.40$$
$$q_\pi(3, a_2) = 0.29$$
$$q_\pi(4, a_2) = 0.13$$
$$q_\pi(5, a_2) = 0.51$$
$$q_\pi(6, \_) = \_$$
$$q_\pi(7, a_3) = 0.30$$
$$q_\pi(8, \_) = 0$$
$$q_\pi(9, a_1) = 0.59$$
$$q_\pi(10, a_4) = 0.60$$
$$q_\pi(11, a_1) = 0.77$$
$$q_\pi(12, \_) = 0$$

Each π is associated with a set of $v_\pi(s)$ or $q_\pi(s,a)$ values

$$\pi_1 \Rightarrow \begin{cases} \pi_1(1) = \cdots \\ \pi_1(2) = \cdots \\ \vdots \\ \pi_1(11) = a_2 \\ \vdots \\ \pi_1(12) = \cdots \end{cases} \Rightarrow \begin{cases} q_{\pi_1}(1,\_) = \cdots \\ q_{\pi_1}(2,\_) = \cdots \\ \vdots \\ q_{\pi_1}(11,a_2) = 0.85 \\ \vdots \\ q_{\pi_1}(12,\_) = \cdots \end{cases}$$

$$\pi_2 \Rightarrow \begin{cases} \pi_2(1) = \cdots \\ \pi_2(2) = \cdots \\ \vdots \\ \pi_2(11) = a_3 \\ \vdots \\ \pi_2(12) = \cdots \end{cases} \Rightarrow \begin{cases} q_{\pi_2}(1,\_) = \cdots \\ q_{\pi_2}(2,\_) = \cdots \\ \vdots \\ q_{\pi_2}(11,a_3) = 0.57 \\ \vdots \\ q_{\pi_2}(12,\_) = \cdots \end{cases}$$



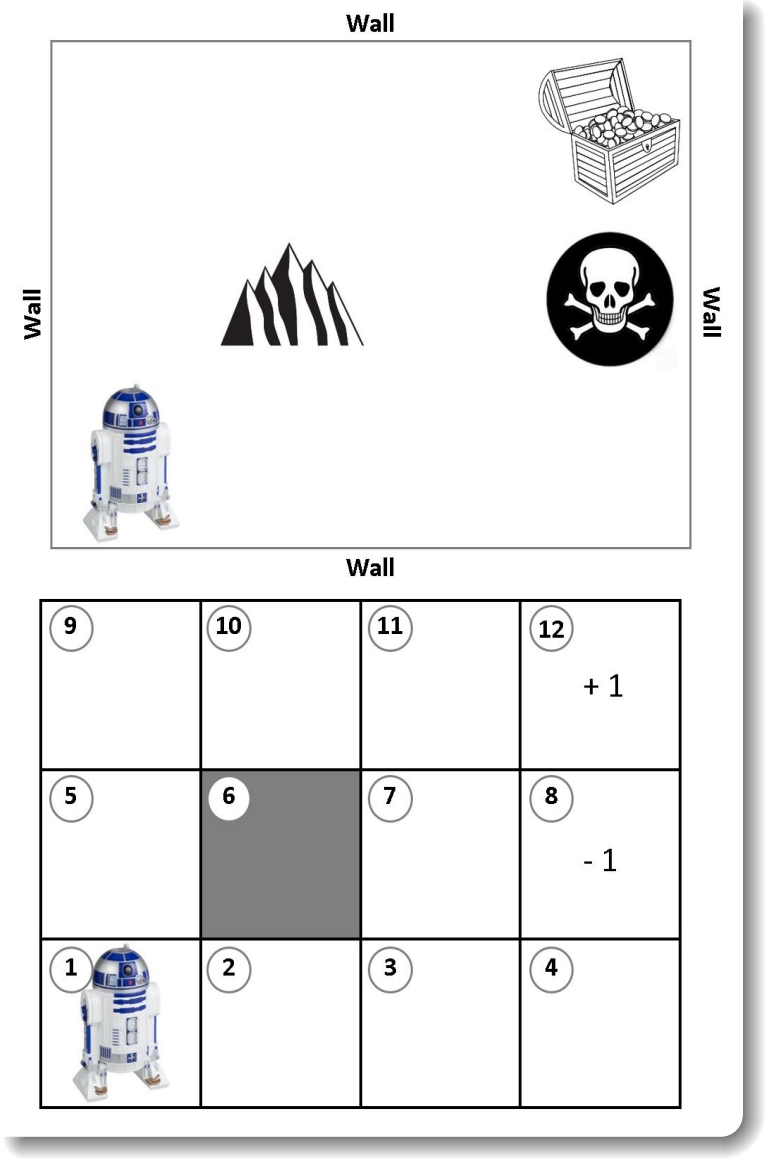
At (11), more desirable to take a_2 than a_3

All else identical, π_1 is "better" then π_2, i.e.,

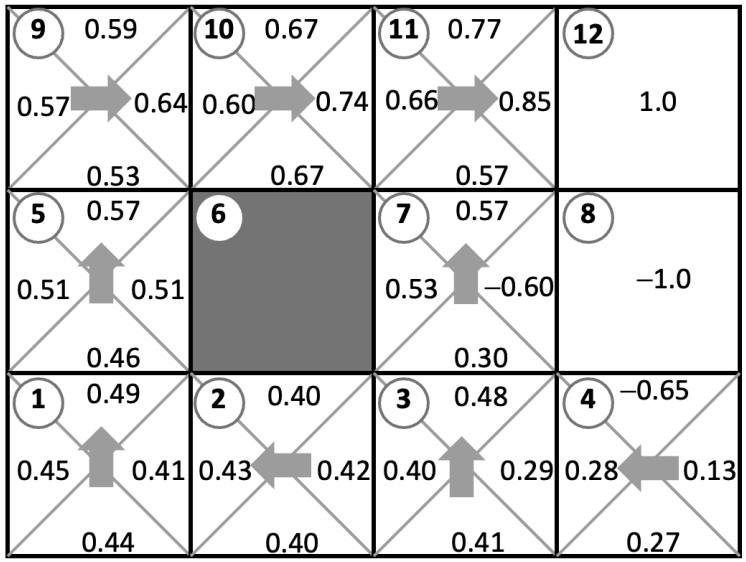
$$\pi_1 > \pi_2$$

since

$$q_{\pi_1}(11,a_2) > q_{\pi_2}(11,a_3)$$



If we know the action value $q(s, a)$ for all (s, a) pairs, then we can find π_* by selecting the action a at a state s that gives highest $q(s, a) \triangleq q_*(s, a)$



Model construction

Formulate problem as MDP

- Model-based methods only

Value function estimation

Determine $q_*(s, a)$ or $v_*(s)$

- This is the 'heart' of RL

Policy extraction

Obtain π_* from $q_*(s, a)$ or $v_*(s)$

- Solution to RL Problem

RL research is mainly about Step 2

Model-based: Dynamic Programming

- Exact method (not covered) - curse of dimensionality

  Need to deal with every state-action pair (s, a)

  Size of $\mathcal{S} \times \mathcal{A}$ too large for solution to be tractable

- Approximate methods

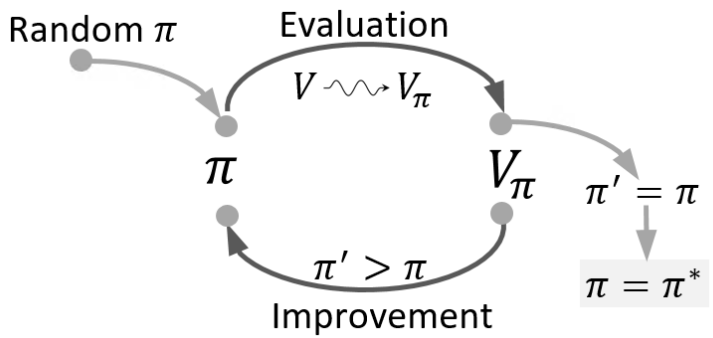
  Policy iteration
  Value iteration

Model-free (Just need actions $\mathcal{A}$; all methods are approximate)

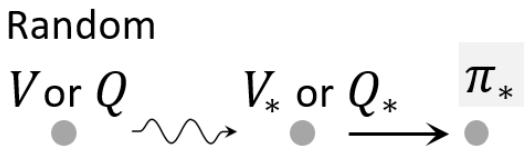
- Monte-Carlo
- Temporal Difference

  TD(0) prediction
  Q-learning
  SARSA

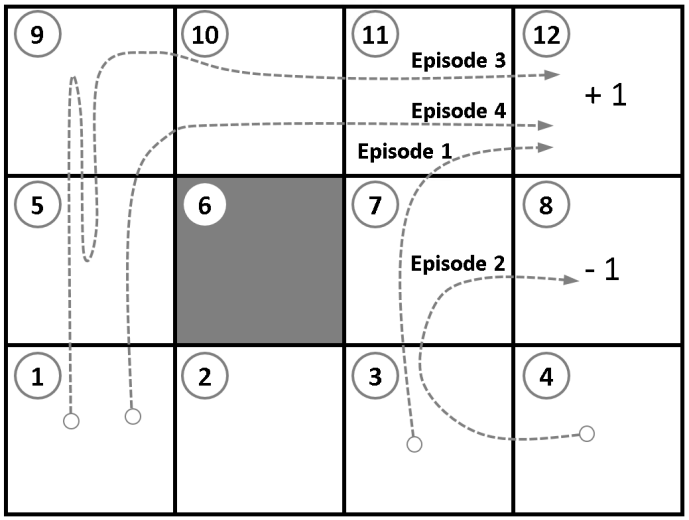
Policy iteration

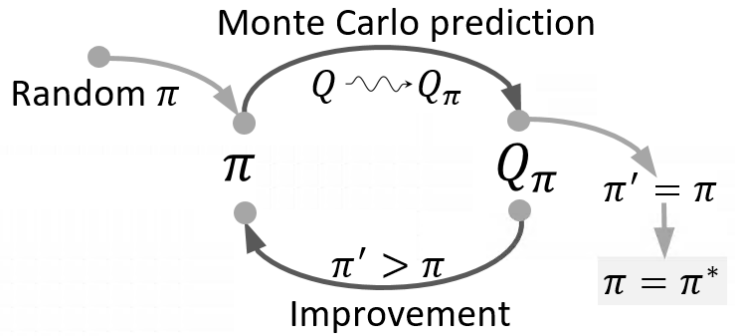


Value iteration



- Agent executes episodes following some π to estimate V_π
- Modify π to obtain "better" policy π'
- Set $\pi = \pi'$. Repeat till no "better" policy is found





Estimate V or Q values iteratively using update rule

$$\text{updated value} \leftarrow \text{current value} + \text{learning rate} \times \underbrace{(\text{estimated target value} - \text{current value})}_{\text{temporal difference error}}$$

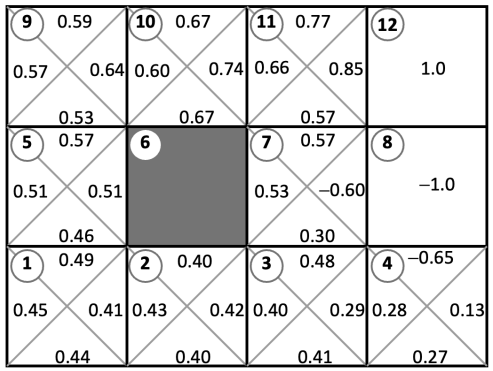
- The well-known Q-learning technique is a temporal difference method

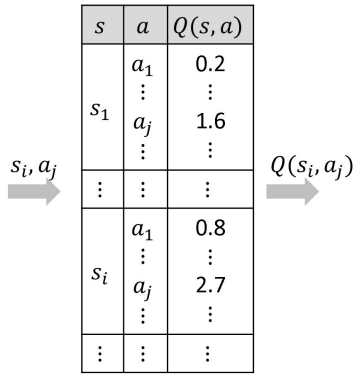
Problem

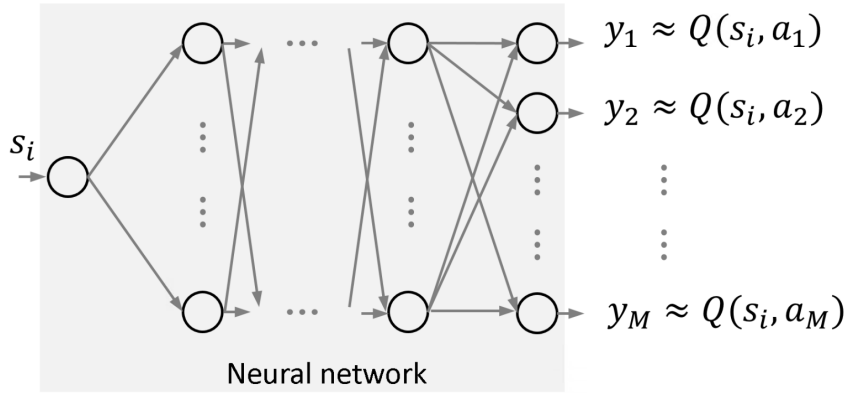
- During learning, need to keep table of $V(s)$ or $Q(s,a)$ values
- To update values, need to search table for current $V(s)$ or $Q(s,a)$
- Table can be huge, e.g., $\sim 2 \cdot 10^{170}$ states $\times$ 250 actions for GO
- Searching through huge table is intractable

Solution

- Use neural network to approximate 'true' but huge table
- This makes Q-learning 'deep' (Deep Q-Network, DQN)







Google AlphaGO

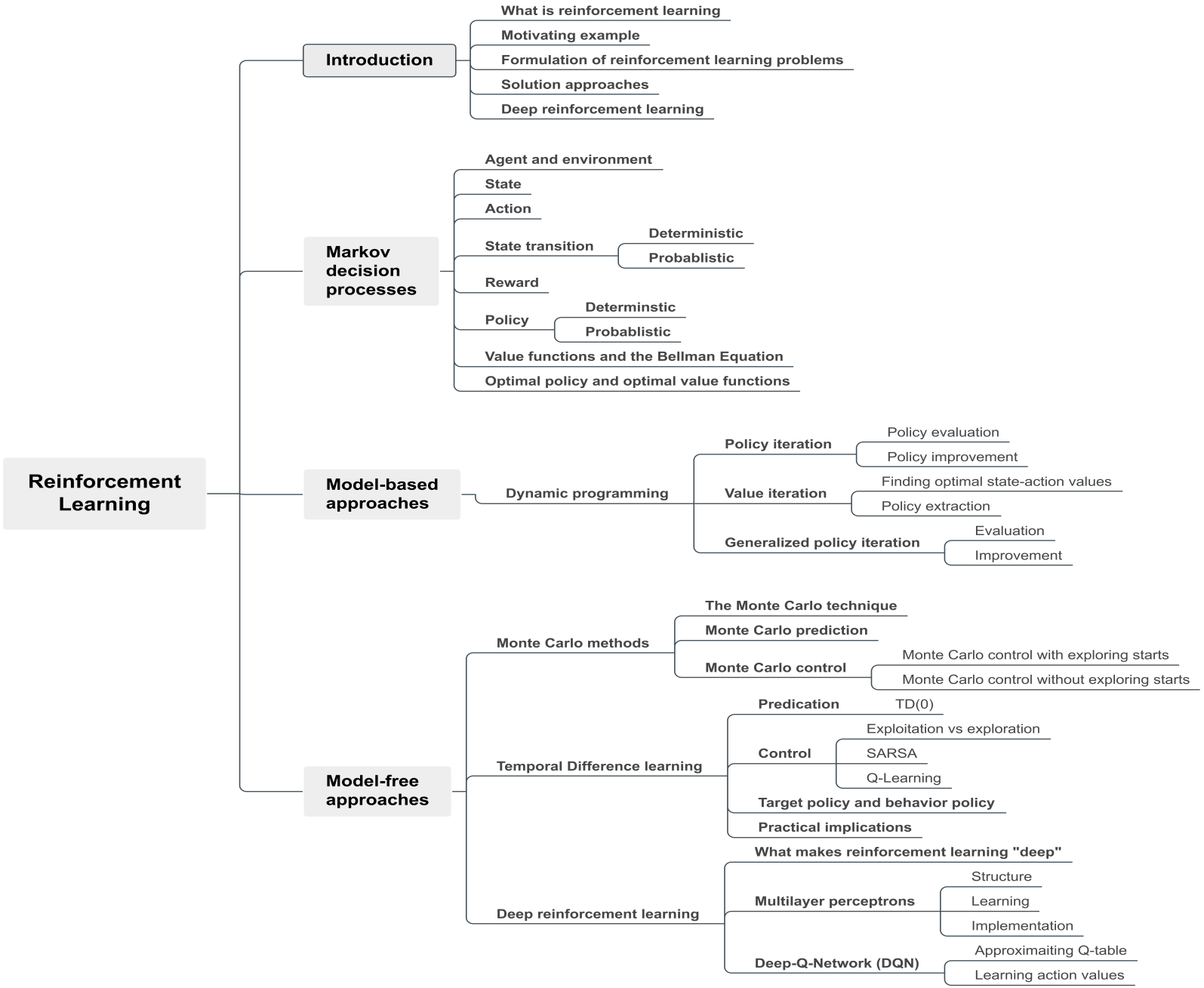
NN with $< 10^6$ processing units approximates Q-table with $\sim 500 \times 10^{170}$ entries

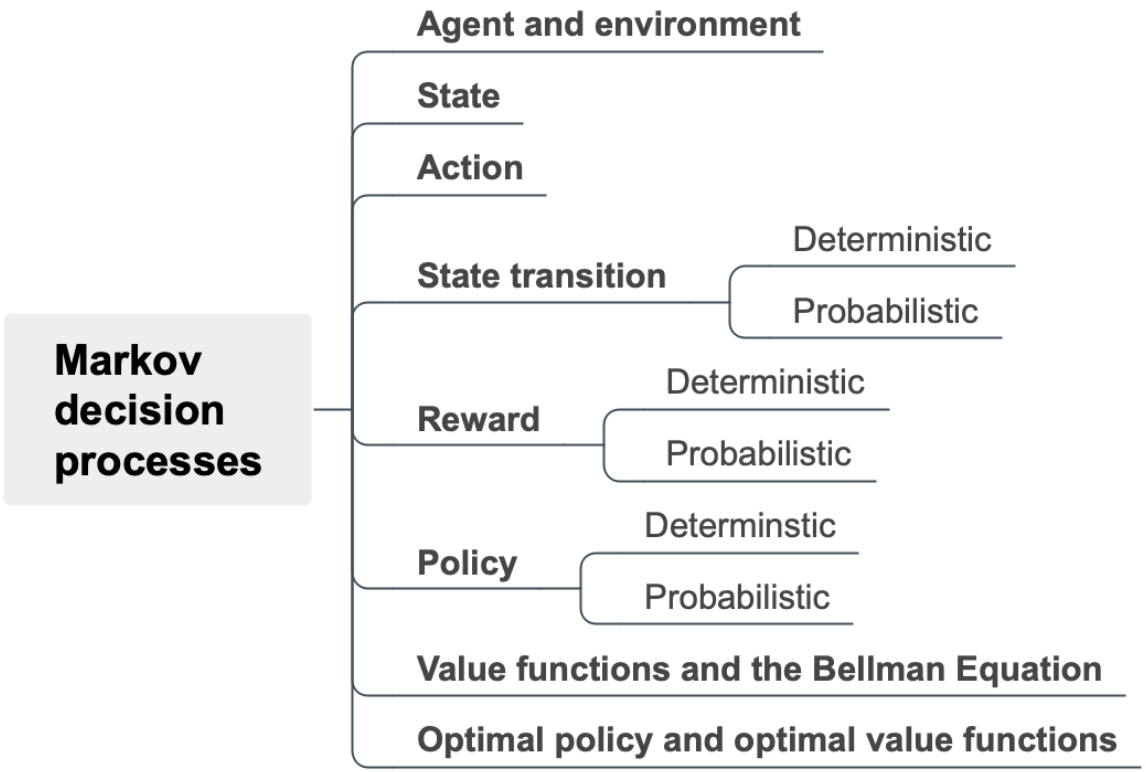
Layer Structure Signals flow forward through connected processing units arranged in layers

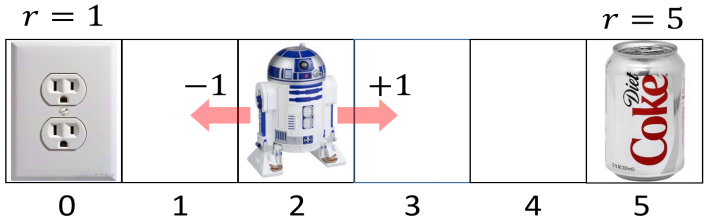
Parametric Representation Connection strength represented by parameter called weight

Universal Approximator With enough units network can proximate any function to any degree of accuracy

Overview of Topics







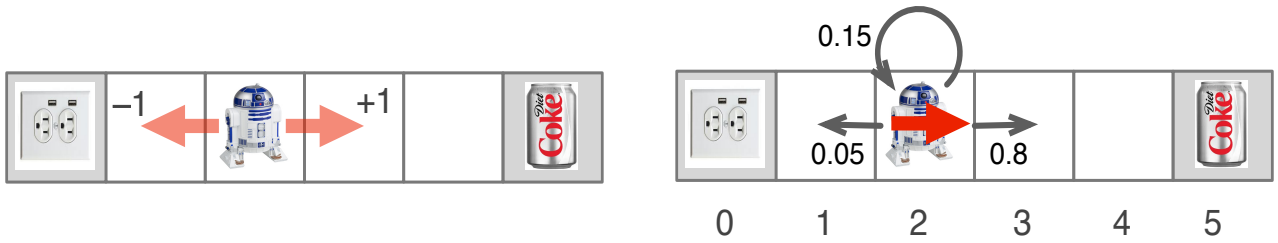
- A set of states: $\mathcal{S} = \{s\} = \{0, 1, 2, 3, 4, 5\}$
- A set of actions: $\mathcal{A} = \{a\} = \{-1, +1\}$
- A transition function (deterministic): $\bar{f}(s, a) = s'$

$$\bar{f}(0, \pm 1) = 0 \quad \bar{f}(1, +1) = 2 \quad \bar{f}(1, -1) = 0 \quad \bar{f}(2, +1) = 3 \quad \bar{f}(2, -1) = 1$$
$$\bar{f}(3, +1) = 4 \quad \bar{f}(3, -1) = 2 \quad \bar{f}(4, +1) = 5 \quad \bar{f}(4, -1) = 3 \quad \bar{f}(5, \pm 1) = 5$$

- A reward function (deterministic): $\bar{\rho}(s, a, s') = r$

$$\bar{\rho}(1, -1, 0) = 1, \quad \bar{\rho}(4, +1, 5) = 5, \quad \bar{\rho}(s, a, s') = 0 \text{ for all other } (s, a, s')$$

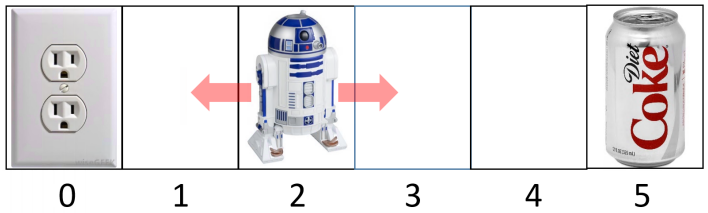
$$f\left(s,a,s'\right)=\mathbb{P}\left(S_{t+1}=s'|\,S_t=s,A_t=a\right)\equiv P_{ss'}^{a}\in[0,1]$$



$$\begin{cases} f(2,1,3)=0.8 \\ f(2,1,2)=0.15 \\ f(2,1,1)=0.05 \end{cases}$$

- Probability of receiving reward r when in state s

$$\rho(r|s) \in [0, 1]$$



# of cans	r	$\rho(r \mid 5)$
0	0	0.4
1	1	0.2
3	3	0.3
5	5	0.1

- Probability of receiving reward r when in state s and taking action a

$$\rho(r|s, a) \in [0, 1]$$

Suppose that a random variable X can take value x_1 with probability p_1, value x_2 with probability p_2, and so on, up to value x_k with probability p_k, and $\sum_k p_k = 1$. Then the expectation of X is defined as

$$\mathbb{E}[X] = x_1 p_1 + x_2 p_2 + \cdots x_k p_k$$

$\mathbb{E}$ is also called the *expectation operator*

$$\begin{aligned} \mathbb{E}[X + Y] &= \mathbb{E}[X] + \mathbb{E}[Y] \\ \mathbb{E}[X + c] &= \mathbb{E}[X] + c \\ \mathbb{E}[cX] &= c\mathbb{E}[X] \end{aligned}$$

where c is a real constant.

- Probability of agent receiving a reward r after taking action a at state s and transitioning to state s'

$$p(s', r \,|\, s, a) = \underbrace{f(s, a, s')}_{\text{transition function}} \cdot \underbrace{\rho(r \,|\, s')}_{\text{reward function}}$$

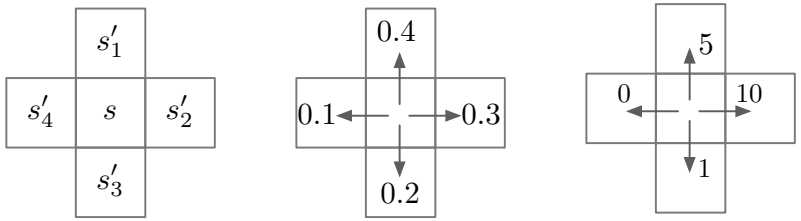
- Expected reward at state s

$$r(s) = \mathbb{E}\big[\, R_t \big|\, S_t = s \,\big] = \sum_{r \in \mathcal{R}} r\, \rho(r \,|\, s)$$

- Expected reward at state s and taking action a

$$r(s, a) = \mathbb{E}\big[\, R_t \big|\, S_{t-1} = s,\, A_{t-1} = a \,\big] = \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} r p(s', r \,|\, s, a)$$

Expected Reward: Probabilistic Transitions and Deterministic Rewards



$$
\begin{aligned}
R_{t+1}|_{s_{t+1}=s'_1} &= \bar{\rho}\left(s,a,s'_1\right) = 5 \qquad & f\left(s,a,s'_1\right) &\equiv P^a_{ss'_1} = 0.4 \\
R_{t+1}|_{s_{t+1}=s'_2} &= \bar{\rho}\left(s,a,s'_2\right) = 10 \qquad & f\left(s,a,s'_2\right) &\equiv P^a_{ss'_2} = 0.3 \\
R_{t+1}|_{s_{t+1}=s'_3} &= \bar{\rho}\left(s,a,s'_3\right) = 1 \qquad & f\left(s,a,s'_3\right) &\equiv P^a_{ss'_3} = 0.2 \\
R_{t+1}|_{s_{t+1}=s'_4} &= \bar{\rho}\left(s,a,s'_4\right) = 0 \qquad & f\left(s,a,s'_4\right) &\equiv P^a_{ss'_4} = 0.1
\end{aligned}
$$

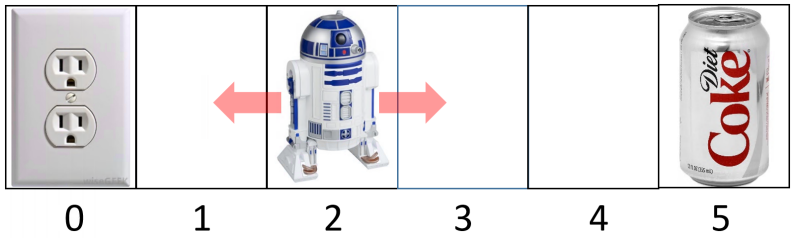
$$
\begin{aligned}
r(s,a) &= \mathbb{E}\left[R_{t+1}\right] \\
&= \sum_{s'}\left(P^a_{ss'},\ R_{t+1}\right) = \sum_{s'} P^a_{ss'}\,\bar{\rho}\left(s,a,s'\right) \\
&= P^a_{ss'_1}\,\bar{\rho}\left(s,a,s'_1\right) + P^a_{ss'_2}\,\bar{\rho}\left(s,a,s'_2\right) \\
&\quad + P^a_{ss'_3}\,\bar{\rho}\left(s,a,s'_3\right) + P^a_{ss'_4}\,\bar{\rho}\left(s,a,s'_4\right) \\
&= 0.4\times 5 + 0.3\times 10 + 0.2\times 1 + 0.1\times 0 \\
&= 5.2
\end{aligned}
$$

Probability of receiving reward r when in state s

$$\rho(r|s) \in [0, 1]$$

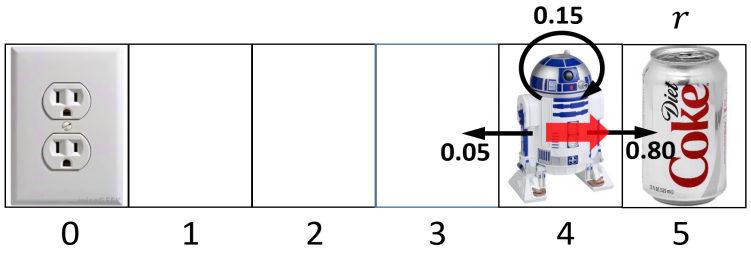
Expected reward when at state s

$$\begin{aligned} r(s) &= \mathbb{E}\big[\, R_t | \, S_t = s \,\big] \\ &= \sum_{r \in \mathcal{R}} r \, \rho(r \,|\, s) \end{aligned}$$



# of cans	r	$\rho(r \,\vert\, 5)$
0	0	0.4
1	1	0.2
3	3	0.3
5	5	0.1

$$\begin{aligned} r(5) &= \mathbb{E}\big[\, R_t | \, S_t = 5 \,\big] = \sum_{r} r \, \rho(r \,|\, 5) \\ &= 0 \cdot 0.4 + 1 \cdot 0.2 + 3 \cdot 0.3 + 5 \cdot 0.1 = 1.6 \end{aligned}$$



s	# of cans	r	$\rho(r \mid s)$
3	0	0	1
4	0	0	1
5	0	0	0.4
	1	1	0.2
	3	3	0.3
	5	5	0.1

$$
\begin{aligned}
& r(s,a) \\
= \quad & r(4,+1) = \mathbb{E}\big[\, R_t \,|\, S_{t-1}=4,\, A_{t-1}=+1 \,\big] \\
= \quad & \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} r \cdot p(s', r \,|\, s, a) = \sum_{r\in\mathcal{R}} \sum_{s'\in\mathcal{S}} r \cdot f(s,a,s') \cdot \rho(r \,|\, s') \\
= \quad & \sum_{r\in\mathcal{R}} r\, f(4,+1,5)\, \rho(r \,|\, 5) + r\, f(4,+1,4)\, \rho(r \,|\, 4) + r\, f(4,+1,3)\, \rho(r \,|\, 3) \\
= \quad & 0 \cdot f(4,+1,5)\, \rho(0 \,|\, 5) + 0 \cdot f(4,+1,4)\, \rho(0 \,|\, 4) + 0 \cdot f(4,+1,3)\, \rho(0 \,|\, 3) + \\
& 1 \cdot f(4,+1,5)\, \rho(1 \,|\, 5) + 1 \cdot f(4,+1,4)\, \rho(1 \,|\, 4) + 1 \cdot f(4,+1,3)\, \rho(1 \,|\, 3) + \\
& 3 \cdot f(4,+1,5)\, \rho(3 \,|\, 5) + 3 \cdot f(4,+1,4)\, \rho(3 \,|\, 4) + 3 \cdot f(4,+1,3)\, \rho(3 \,|\, 3) + \\
& 5 \cdot f(4,+1,5)\, \rho(5 \,|\, 5) + 5 \cdot f(4,+1,4)\, \rho(5 \,|\, 4) + 5 \cdot f(4,+1,3)\, \rho(5 \,|\, 3) \\
= \quad & 1 \times 0.8 \times 0.2 + 3 \times 0.8 \times 0.3 + 5 \times 0.8 \times 0.1 = 1.28
\end{aligned}
$$

A state S_t is said to be *Markov* if and only if

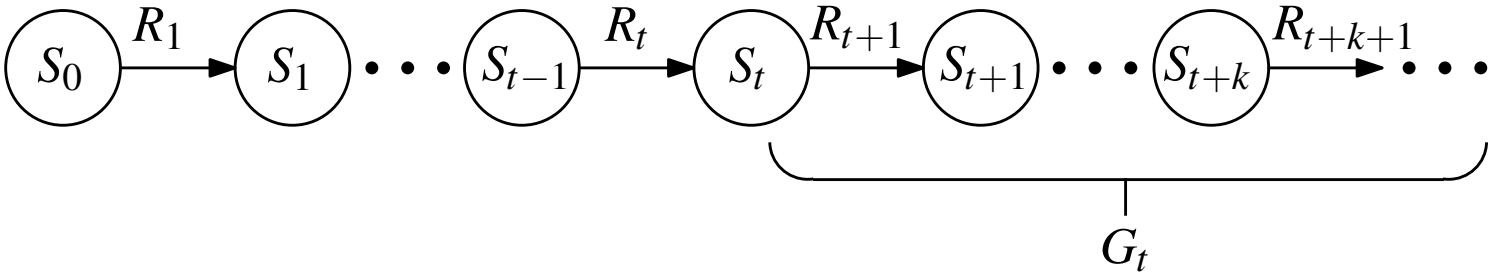
$$\mathbb{P}(S_{t+1}|S_t) = \mathbb{P}(S_{t+1}|S_t, S_{t-1}, \ldots, S_1)$$

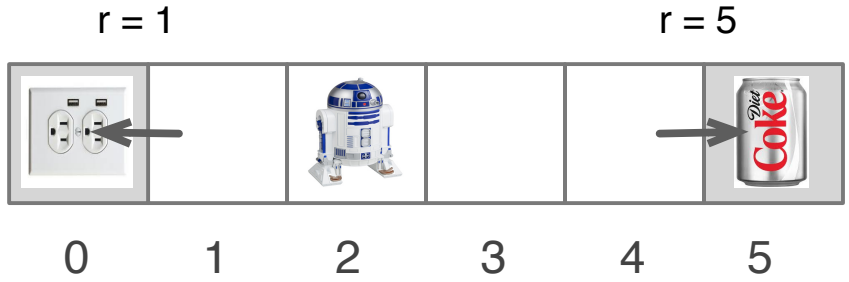
- Next state is completely determined by current state
- Infomation about past states are not needed

- Agent reaches some state s after t time steps
- Total reward agent can obtain *from this point onward*:

$$G_t \triangleq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

  - $\gamma \in [0, 1]$ is called the *discount rate*



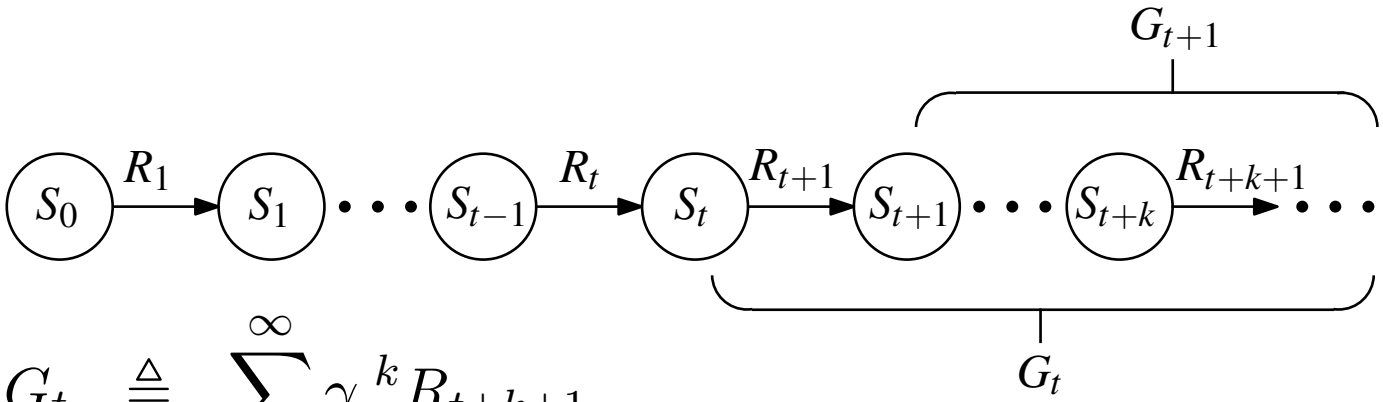


$$\bar{f}(s,a) = \begin{cases} s+a & 1 \le s \le 4 \\ s & s = 0 \text{ or } s = 5 \end{cases} \qquad \bar{\rho}(s,a,s') = \begin{cases} 5 & s \neq 5 \text{ and } s' = 5 \\ 1 & s \neq 0 \text{ and } s' = 0 \\ 0 & \text{otherwise} \end{cases}$$

Robot at state 2 moves to 5 without changing direction:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \\ &= \bar{\rho}(2,1,3) + 0.5 \cdot \bar{\rho}(3,1,4) + 0.5^2 \cdot \bar{\rho}(4,1,5) \\ &= 0 + 0 + 0.25 \cdot 5 \\ &= 1.25 \end{aligned}$$

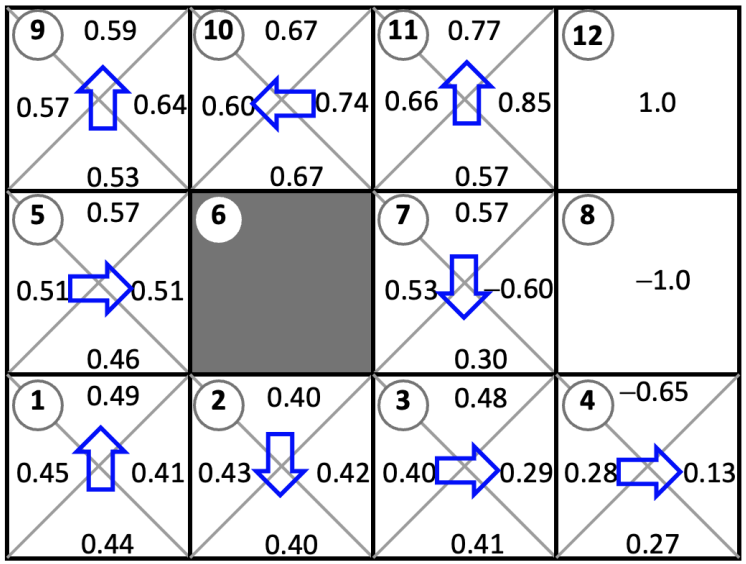
Return at time step t can be expressed in terms of return at $t+1$

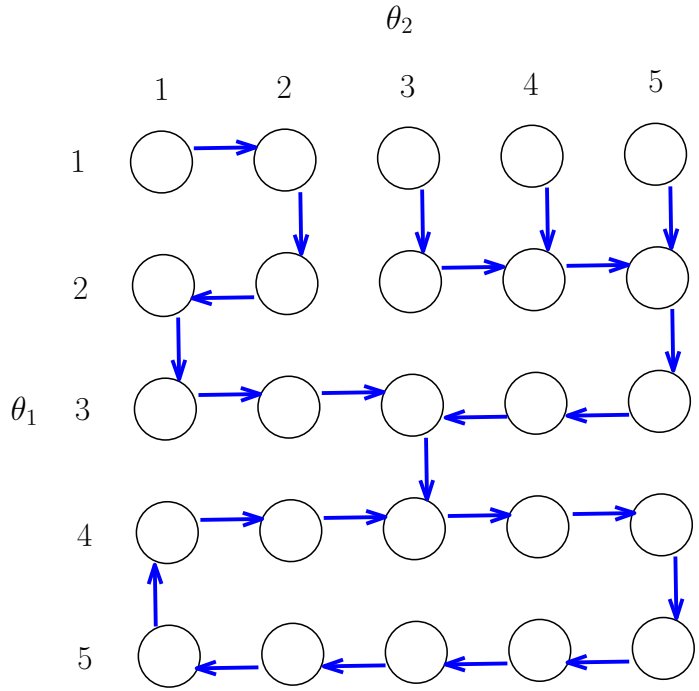


$$
\begin{aligned}
G_t &\triangleq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \\
&= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \cdots \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \cdots \right) \\
&= R_{t+1} + \gamma \Bigg(\underbrace{\sum_{k=0}^{\infty} \gamma^k R_{t+k+2}}_{G_{t+1}} \Bigg) = R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

- This relationship will be used in Monte Carlo method

Policy specifies the action to be taken by agent at each state

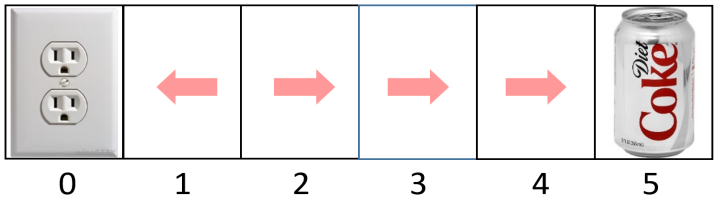




Policy specifies the action to be taken by agent at each state

- Deterministic: $\pi(s) = a \in \mathcal{A}$

Example: $a = -1$ (left); $a = +1$ (right)

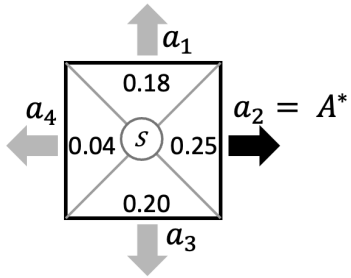


$$\begin{cases} \pi(1) = -1 \\ \pi(s) = \pm 1 & \text{if } s = 0, 5 \\ \pi(s) = 1 & \text{if } s = 2, 3, 4 \end{cases}$$

- Probabilistic: $\pi(a \,|\, s) = \mathbb{P}\{A_t = a \,|\, S_t = s\} \in [0, 1]$

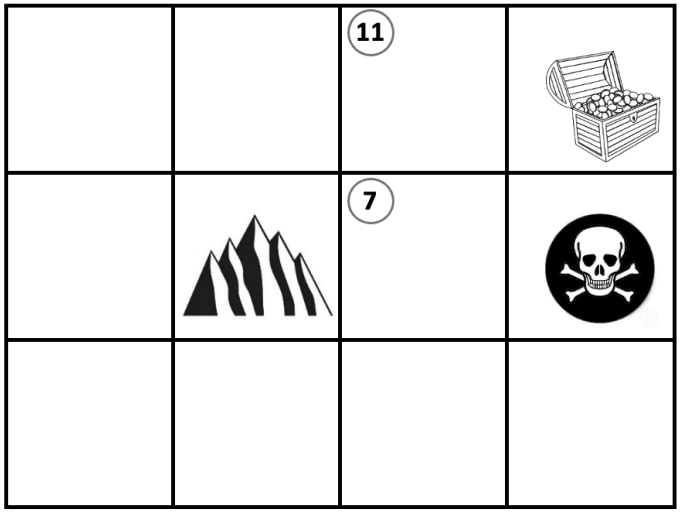
Example: ϵ-soft policy (commonly used in Q-learning; discussed later)

$$\pi(a \,|\, s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{if } a = A^* \quad \text{(Usually 1 action)} \\ \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{if } a \neq A^* \quad (|\mathcal{A}(s)| - 1 \text{ actions}) \end{cases}$$

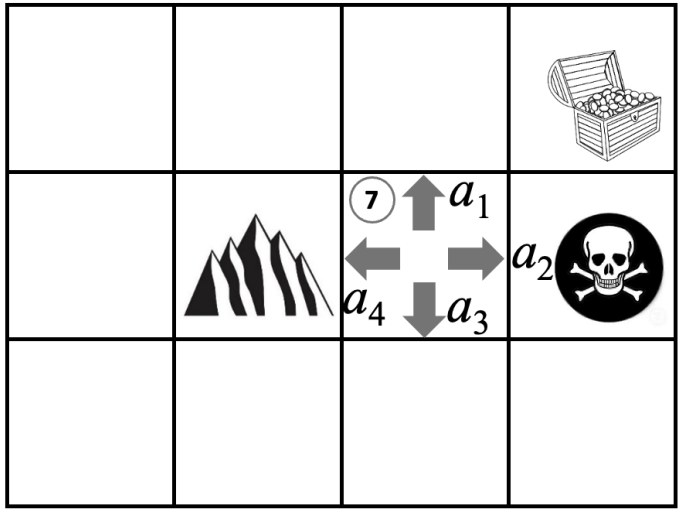


where A^* is the 'best' action to take at s and $|\mathcal{A}(s)|$ is the number of actions available for agent to choose at s. Note that all action probabilities sum to 1

Given two states 7 and 11, which one do you prefer?



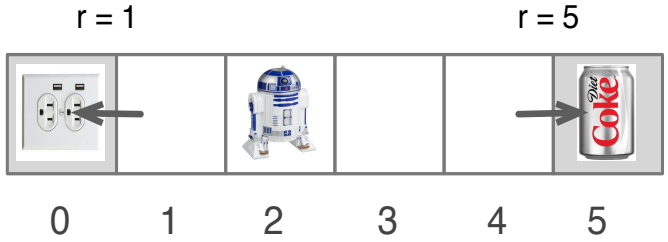
Given four actions at state 7, which one do you prefer?



- Some states, or state-action pairs, are 'worth' more than others
- Such 'worth' depends on the chosen policy π
  - What if $\pi_1(7) = a_1$, $\pi_1(11) = a_2$, $\pi_2(11) = a_3$, $\pi_2(7) = a_2$?
  - What if $\pi_1(7) = a_1$ and $\pi_2(7) = a_3$?

$$v_\pi(s) \triangleq \mathbb{E}_\pi\Big[G_t \,|\, S_t = s\Big] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

If transitions and policy are deterministic, then $v_\pi(s) = G_t\,\big|_{S_t=s}$



$\pi(1) = -1;\ \pi(s) = +1,\ s \in \{2,3,4\}$

$$\bar{f}(s,a) = \begin{cases} s+a & 1 \le s \le 4 \\ s & s = 0 \text{ or } s = 5 \end{cases}$$

$$\bar{\rho}(s,a,s') = \begin{cases} 5 & s \neq 5 \text{ and } s' = 5 \\ 1 & s \neq 0 \text{ and } s' = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} v_\pi(2) &= \mathbb{E}_\pi\Big[G_t \,|\, S_t = s\Big] \\ &= G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \\ &= \bar{\rho}(2,1,3) + 0.5 \cdot \bar{\rho}(3,1,4) + 0.5^2 \cdot \bar{\rho}(4,1,5) \\ &= 0 + 0 + 0.25 \cdot 5 = 1.25 \end{aligned}$$

Action Value Function (*aka* state-action value function)

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi\Big[G_t \Big| S_t = s, A_t = a\Big] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right] \end{aligned}$$

- More about $q_\pi(s,a)$ when discussing the Bellman Equation

For a deterministic policy

$$v_\pi(s) = q_\pi(s, \pi(s))$$

For a terminal (e.g., goal) state s

$$v_\pi(s) = 0 \text{ and } q_\pi(s,a) = 0 \text{ for all } a \in \mathcal{A}(s) \text{ and any } \pi$$

π_i is said to be "better" than π_j, written as $\pi_i > \pi_j$, if

$$v_{\pi_i}(s) > v_{\pi_j}(s) \quad \text{for all } s \in \mathcal{S}$$

Objective of reinforcement learning is to find "best" policy π_*

Since with finite states and actions, total number of polices is finite, there must be at least one policy π_i such that

$$v_{\pi_i}(s) = v_{\pi_*}(s) \equiv v_*(s) \triangleq \max_\pi v_\pi(s)$$

To compare policies we need to evaluate $v_\pi(s)$ for given π

$\Rightarrow$ Solve Bellman Equation for $v_\pi(s)$ if $p(s', r|s, a)$ is known

The Bellman Equation expresses the relationship between

$v_\pi(s)$ and $v_\pi(s')$:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma \, v_\pi(s') \Big]$$

or $q_\pi(s, a)$ and $q_\pi(s', a')$:

$$q_\pi(s, a) = \sum_{s', r} p(s', \, r \mid s, a) \Big[r + \gamma \, \pi(a' \mid s') \, q_\pi(s', a') \Big]$$

where s' is the successor of s, and a' is the action taken at s'

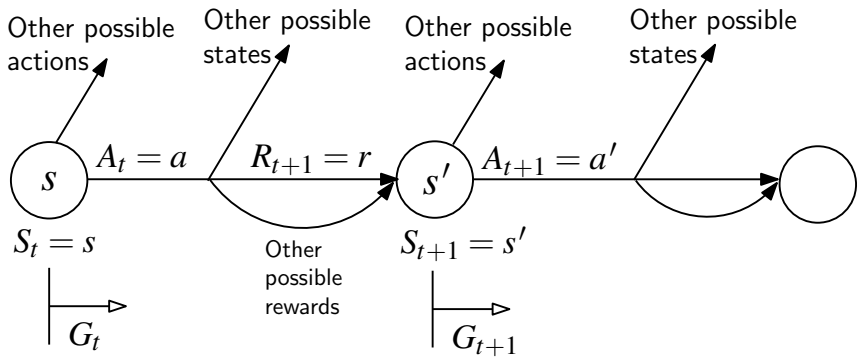
$$
\begin{aligned}
& v_\pi(s) \\
\triangleq\ & \mathbb{E}_\pi\Big[G_t \,\Big|\, S_t = s\Big] \quad \text{(from the definition of } v_\pi\text{)} \\
=\ & \mathbb{E}_\pi\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \,\Big|\, S_t = s\Big] \\
=\ & \mathbb{E}_\pi\Big[R_{t+1} \,\Big|\, S_t = s\Big] + \mathbb{E}_\pi\Big[\gamma\, R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \,\Big|\, S_t = s\Big] \\
=\ & \mathbb{E}_\pi\Big[R_{t+1} \,\Big|\, S_t = s\Big] + \gamma\, \mathbb{E}_\pi\Big[\underbrace{\big(R_{t+2} + \gamma\, R_{t+3} + \gamma^2 R_{t+4} + \cdots\big)}_{G_{t+1}\ (\text{with } S_{t+1} = s')} \,\Big|\, S_t = s\Big] \\
=\ & \mathbb{E}_\pi\Big[R_{t+1} \,\Big|\, S_t = s\Big] + \gamma\, \mathbb{E}_\pi\Big[G_{t+1} \,\Big|\, S_t = s\Big]
\end{aligned}
$$

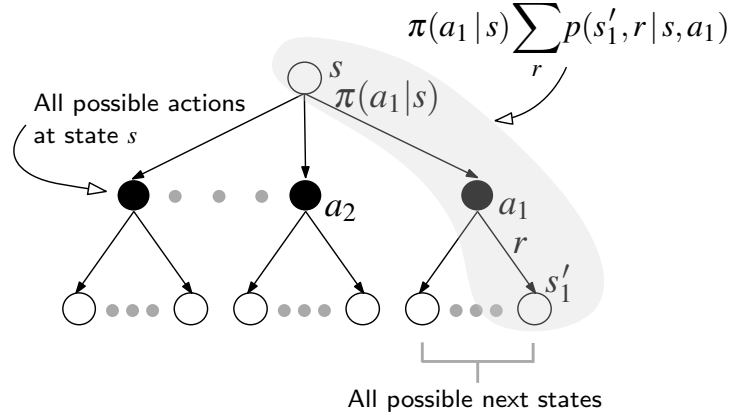
First term:

$$
\mathbb{E}_\pi\Big[R_{t+1} \,\Big|\, S_t = s\Big] = \sum_a \pi(a \,|\, s) \sum_{s'} \sum_r r \cdot p(s', r \,|\, s, a)
$$

Second term is more complicated ...

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} \mid S_t = s\right] + \gamma\, \mathbb{E}_\pi\left[G_{t+1} \mid S_t = s\right]$$





G_{t+1} is obtained from step $t+1$ onward, i.e., when the agent is at state $S_{t+1} = s'$. However, in second term of v_π expression above G_{t+1} is with reference to step t, i.e., when the agent is still at state $S_t = s$

So need to take into account probability of agent reaching s' from s, i.e., to express $\mathbb{E}_\pi\left[G_{t+1} | S_t = s\right]$ in terms of $\mathbb{E}_\pi\left[G_{t+1} | S_{t+1} = s'\right]$

① Expected return when reaching s'_1 ② Prob. of taking a_1 and reaching s'_1

$$
\begin{aligned}
& \mathbb{E}_\pi\left[G_{t+1} \middle| S_t = s\right] \\
= \quad & \underbrace{\pi(a_1|s)\left(\sum_r p(s'_1, r|s, a_1)\right)}_{②} \times \underbrace{\mathbb{E}_\pi\left[G_{t+1}|S_{t+1} = s'_1\right]}_{①} \\
+ \quad & \pi(a_1|s)\left(\sum_r p(s'_2, r|s, a_1)\right) \times \mathbb{E}_\pi\left[G_{t+1}|S_{t+1} = s'_2\right] \\
+ \quad & \pi(a_1|s)\left(\sum_r p(s'_3, r|s, a_1)\right) \times \mathbb{E}_\pi\left[G_{t+1}|S_{t+1} = s'_3\right] \\
+ \quad & \cdots \\
+ \quad & \pi(a_2|s)\left(\sum_r p(s'_1, r|s, a_2)\right) \times \mathbb{E}_\pi\left[G_{t+1}|S_{t+1} = s'_1\right] \\
+ \quad & \cdots \cdots \\
= \quad & \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a) \underbrace{\mathbb{E}_\pi\left[G_{t+1}|S_{t+1} = s'\right]}_{v_\pi(s')} \\
= \quad & \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)\, v_\pi(s')
\end{aligned}
$$

$$\mathbb{E}_\pi\Big[R_{t+1}\,\Big|\,S_t=s\Big]=\sum_a \pi(a\,|\,s)\sum_{s'}\sum_r r\cdot p(s',r\,|\,s,a)$$

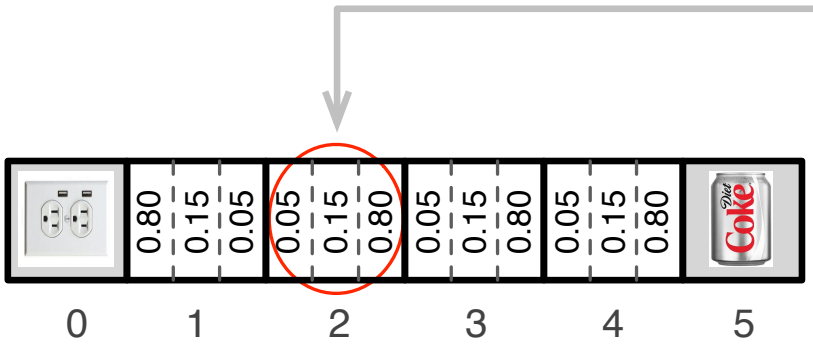
$$\mathbb{E}_\pi\Big[G_{t+1}\,\Big|\,S_t=s\Big]=\sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r\,|\,s,a)\,v_\pi(s')$$

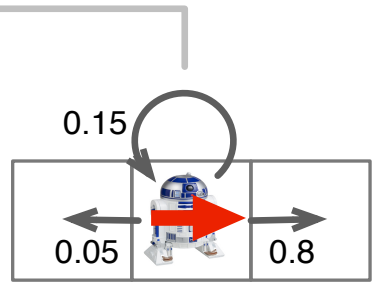
$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\Big[R_{t+1}\,\Big|\,S_t=s\Big]+\gamma\,\mathbb{E}_\pi\Big[G_{t+1}\,\Big|\,S_t=s\Big]\\
&= \sum_a \pi(a\,|\,s)\sum_{s'}\sum_r r\cdot p(s',r\,|\,s,a)\\
&+ \gamma\sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r\,|\,s,a)\,v_\pi(s')\\
&= \sum_a \pi(a\,|\,s)\sum_{s',r} p(s',r\,|\,s,a)\big[r+\gamma\,v_\pi(s')\big]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

$$
\begin{aligned}
& q_\pi(s,a) \\
= \;& \mathbb{E}_\pi\Big[R_{t+1} \,\Big|\, S_t=s, A_t=a\Big] + \gamma\, \mathbb{E}_\pi\Big[G_{t+1} \,\Big|\, S_t=s, A_t=a\Big] \\
= \;& \sum_{s'}\sum_{r} r \cdot p(s', r \,|\, s, a) \\
+ \;& \gamma \sum_{s',a'}\sum_{r} p(s', r \,|\, s, a)\, \pi(a' \,|\, s')\, q_\pi(s', a') \\
= \;& \sum_{s',r} p(s', r \,|\, s, a)\Big[r + \gamma \sum_{a'} \pi(a' \,|\, s')\, q_\pi(s', a')\Big] \quad \leftarrow \text{Bellman Eqn for } q_\pi \\
= \;& \sum_{s',r} p(s', r \,|\, s, a)\Big[r + \gamma\, v_\pi(s')\Big] \quad \leftarrow v_\pi(s') \text{ is related to } q_\pi(s', a')
\end{aligned}
$$

$$
\begin{aligned}
& \textstyle\sum_a \pi(a \,|\, s)\, q_\pi(s,a) \\
= \;& \textstyle\sum_a \pi(a \,|\, s) \Big(\mathbb{E}_\pi\left[R_{t+1} \,\middle|\, S_t=s, A_t=a\right] + \gamma\, \mathbb{E}_\pi\left[G_{t+1} \,\middle|\, S_t=s, A_t=a\right]\Big) \\
= \;& \textstyle\sum_a \pi(a|s) \Big(\mathbb{E}_\pi\left[R_{t+1} \middle| S_t=s, A_t=a\right]\Big) + \\
& \textstyle\sum_a \pi(a|s) \Big(\gamma \mathbb{E}_\pi\left[G_{t+1} \middle| S_t=s, A_t=a\right]\Big) \\
= \;& \mathbb{E}_\pi\left[R_{t+1} \,\middle|\, S_t=s\right] + \gamma\, \mathbb{E}_\pi\left[G_{t+1} \,\middle|\, S_t=s\right] = v_\pi(s)
\end{aligned}
$$

Example: Evaluating q_π





$$\bar{\rho}(s,a,s') = \begin{cases} 5 & \text{if } s \neq 5 \text{ and } s' = 5 \\ 1 & \text{if } s \neq 0 \text{ and } s' = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{cases} \pi(1) = -1 \\ \pi(s) = 1 \quad \text{if} \quad s = 2,3,4 \\ \pi(s) = \pm 1 \quad \text{if} \quad s = 0,5 \end{cases}$$

$$f(1,-1,0) = 0.80 \equiv P_{10}^{-1} \qquad f(2,1,1) = 0.05 \equiv P_{21}^{1}$$
$$f(1,-1,1) = 0.15 \equiv P_{11}^{-1} \qquad f(2,1,2) = 0.15 \equiv P_{22}^{1}$$
$$f(1,-1,2) = 0.05 \equiv P_{12}^{-1} \qquad f(2,1,3) = 0.80 \equiv P_{23}^{1}$$

$$f(3,1,2) = 0.05 \equiv P_{32}^{1} \qquad f(4,1,3) = 0.05 \equiv P_{43}^{1}$$
$$f(3,1,3) = 0.15 \equiv P_{33}^{1} \qquad f(4,1,4) = 0.15 \equiv P_{44}^{1}$$
$$f(3,1,4) = 0.80 \equiv P_{34}^{1} \qquad f(4,1,5) = 0.80 \equiv P_{45}^{1}$$
$$f(0,\pm 1,0) = 1 \equiv P_{00}^{\pm 1} \qquad f(5,\pm 1,5) = 1 \equiv P_{55}^{\pm 1}$$

$$
\begin{aligned}
& q_\pi(2,1) \\
= \quad & \sum_{s'} P^a_{ss'} \Big(\bar{\rho}(s,a,s') + \gamma q_\pi\left(s',a'\right) \Big) = \sum_{j=1}^{3} P^1_{2j} \Big(\bar{\rho}(2,1,j) + \gamma q_\pi\left(j,\pi(j)\right) \Big) \\
= \quad & P^1_{21} \Big(\bar{\rho}(2,1,1) + \gamma q_\pi\left(1,-1\right) \Big) \quad \text{(Note: } j=1 \text{ and } \pi(1)=-1\text{)} \\
+ \quad & P^1_{22} \Big(\bar{\rho}(2,1,2) + \gamma q_\pi\left(2,1\right) \Big) \quad \text{(Note: } j=2 \text{ and } \pi(2)=1\text{)} \\
+ \quad & P^1_{23} \Big(\bar{\rho}(2,1,3) + \gamma q_\pi\left(3,1\right) \Big) \quad \text{(Note: } j=3 \text{ and } \pi(3)=1\text{)} \\
= \quad & 0.05\Big(0 + 0.5 q_\pi\left(1,-1\right)\Big) + 0.15\Big(0 + 0.5 q_\pi\left(2,1\right)\Big) + 0.80\Big(0 + 0.5 q_\pi\left(3,1\right)\Big) \\
= \quad & 0.025 q_\pi(1,-1) + 0.075 q_\pi(2,1) + 0.4 q_\pi(3,1)
\end{aligned}
$$

$$
\begin{aligned}
q_\pi(2,1) &= 0.025 q_\pi(1,-1) + 0.075 q_\pi(2,1) + 0.4 q_\pi(3,1) \\
q_\pi(1,-1) &= 0.8 + 0.075 q_\pi(1,-1) + 0.025 q_\pi(2,1) \\
q_\pi(3,1) &= 0.025 q_\pi(2,1) + 0.075 q_\pi(3,1) + 0.4 q_\pi(4,1) \\
q_\pi(4,1) &= 0.025 q_\pi(3,1) + 0.075 q_\pi(4,1) + 4
\end{aligned}
$$

Solution: $q_\pi(1,-1) = 0.89,\ q_\pi(2,1) = 0.85,\ q_\pi(3,1) = 1.92,\ q_\pi(4,1) = 4.38$

For reinforcement learning problem with finite states and actions, total number of polices n is finite

Consider two policies π_i and π_j, with $i, j \in \{1, 2, \ldots, n\}$.
We say that $\pi_i > \pi_j$ if $v_{\pi_i}(s) > v_{\pi_j}(s)$ for all $s \in \mathcal{S}$

Objective is to find 'best' (i.e., optimal) policy

Optimal Policy π_*

A policy is *optimal* if it is better than any other policy

$$\pi_* = \left\{\pi_i \,|\, v_{\pi_i}(s) \equiv v_*(s) > v_{\pi_j}(s) \text{ for all } j \neq i\right\}$$

π^* may not be unique

Optimal Policy π_*

A policy is *optimal* if it is better than any other policy

$$\pi_* = \left\{ \pi_i \,|\, v_{\pi_i}(s) \equiv v_*(s) > v_{\pi_j}(s) \text{ for all } j \neq i \right\}$$

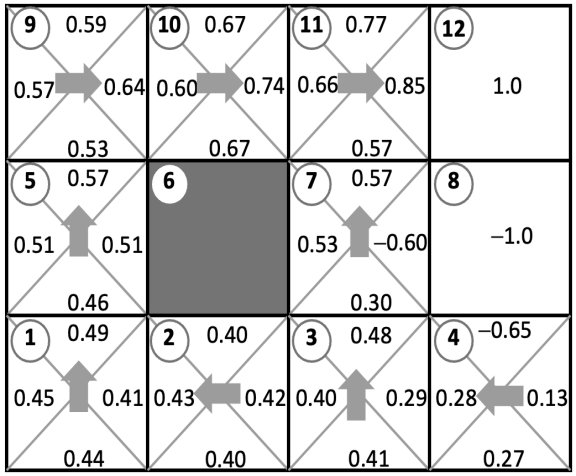
The $v_\pi(s)$ and $q_\pi(s)$ values corresponding to $\pi = \pi_*$, denoted by $v_*(s)$ and $q_*(s)$ respectively, are called the *optimal* state and action values, i.e.,

$$\begin{aligned} v_*(s) &\triangleq \max_\pi v_\pi(s) \\ q_*(s,a) &\triangleq \max_\pi q_\pi(s,a) \end{aligned}$$

$$v_*(s) \triangleq \max_\pi v_\pi(s)$$

$$q_*(s,a) \triangleq \max_\pi q_\pi(s,a)$$

In practice, prefer deterministic π_*, i.e., $\pi_*(a \mid s) = 1$ or $\pi_*(s) = a$

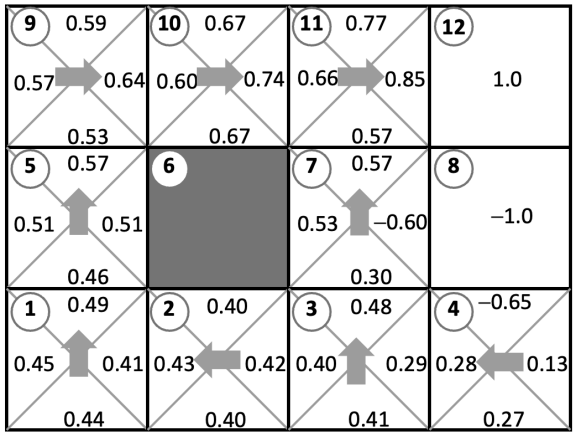


If $v_*(s)$ or $q_*(s,a)$ are known, we can 'extract' a deterministic optimal policy $\pi_*(s) = a$ by picking at state s the action a based on $v_*(s)$ or corresponding to largest $q_*(s,a)$:

$$a \in \operatorname*{argmax}_\sigma q_*(s,\sigma) \quad \text{(i.e., all } \sigma\text{'s such that } q_*(s,\sigma) \text{ is max.)}$$

$$= \operatorname*{argmax}_\sigma \left(\sum_{s',r} p(s',r \mid s,\sigma) \Big[r + \gamma\, v_*(s') \Big] \right)$$

Note that we can do so using $v_*(s)$ only if we know $p(s',r \mid s,a)$



For deterministic policy

$$\pi_*(a \mid s) = 1, \text{ where } a \in \operatorname{argmax}_\sigma \; q_*(s, \sigma)$$

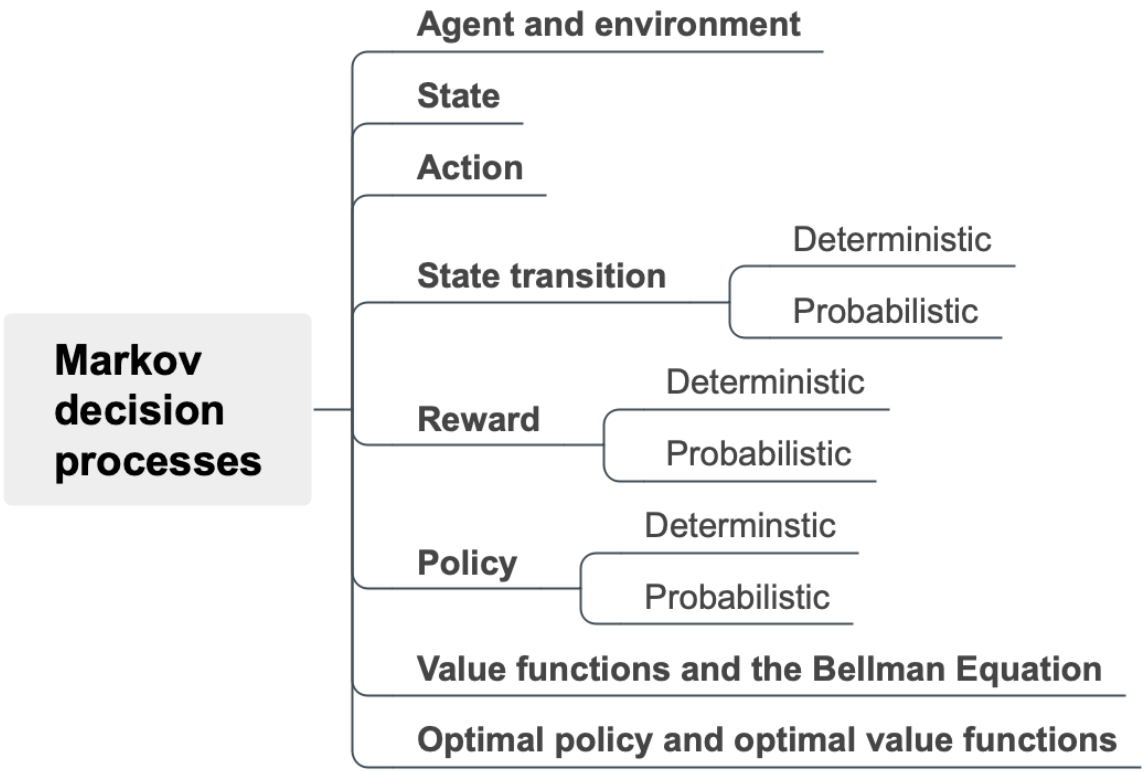
we have

$$\begin{aligned} v_*(s) &= \sum_a \pi_*(a \mid s)\, q_*(s, a) \\ &= \max_a q_*(s, a) \end{aligned}$$

Since v_* satisfies the Bellman Equation, so

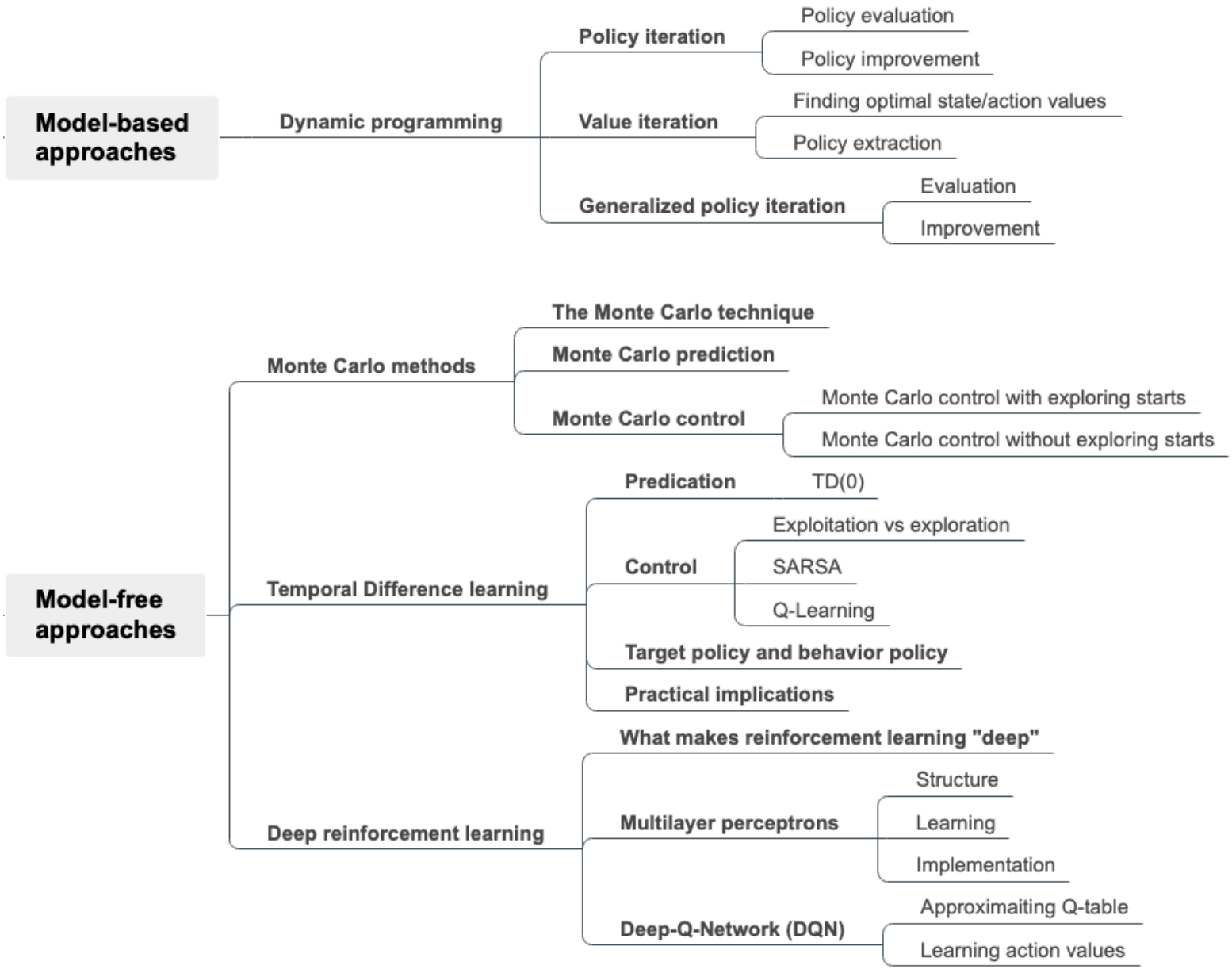
$$\begin{aligned} v_*(s) &= \max_a q_*(s, a) = \max_a \sum_{s', r} p(s', r \mid s, a) \big[r + \gamma\, v_*(s') \big] \\ q_*(s, a) &= \sum_{s', r} p(s', r \mid s, a) \big[r + \gamma\, v_*(s') \big] \\ &= \sum_{s', r} p(s', r \mid s, a) \big[r + \gamma \max_{a'} q_*(s', a') \big] \end{aligned}$$

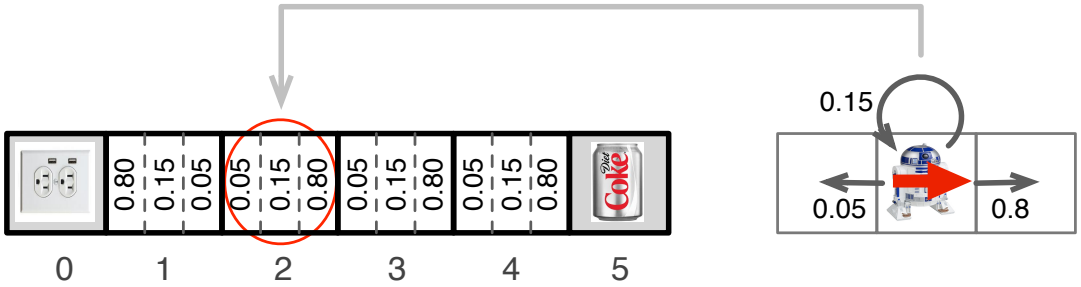
Markov Decision Process: Overview



How to find an optimal policy?

How to find an optimal policy?





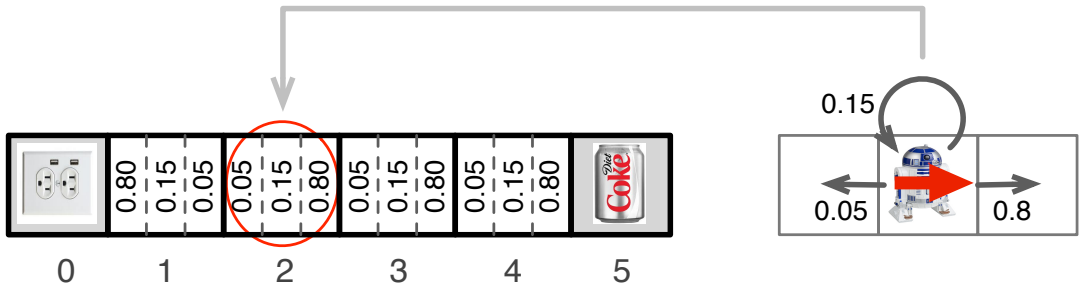
Note that

$$q_*(0, \pm 1) = 0$$
$$q_*(5, \pm 1) = 0$$

Consider $s = 2$ and $a = 1$ for example:

$$\begin{aligned}
q_*(2,1) &= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \\
&= \sum_{s'} p(s', r \mid 2, 1)\left[r + \gamma \max_{a'} q_*(s', a')\right] \\
&= \sum_{s'} P^1_{2s'}\left[r + \gamma \max_{a'} q_*(s', a')\right] \\
&= P^1_{23}\left[r + \gamma \max\{q_*(3,1), q_*(3,-1)\}\right] \\
&+ P^1_{22}\left[r + \gamma \max\{q_*(2,1), q_*(2,-1)\}\right] \\
&+ P^1_{21}\left[r + \gamma \max\{q_*(1,1), q_*(1,-1)\}\right]
\end{aligned}$$

- Overall 8 equations for 8 $q_*(s, a)$ variables (4 states $\times$ 2 actions)
- Similar situation for $v_*(s)$
- Equations nonlinear (due to $\max$); cannot be solved explicitly



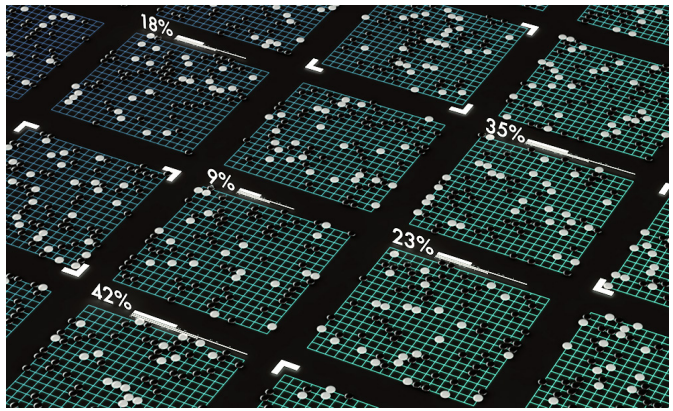
- Overall 8 equations for 8 $q_*(s, a)$ variables
- Similar situation for $v_*(s)$

- Solution becomes intractable with large $\mathcal{S}$ and $\mathcal{A}$

Estimated positions and moves in *GO*

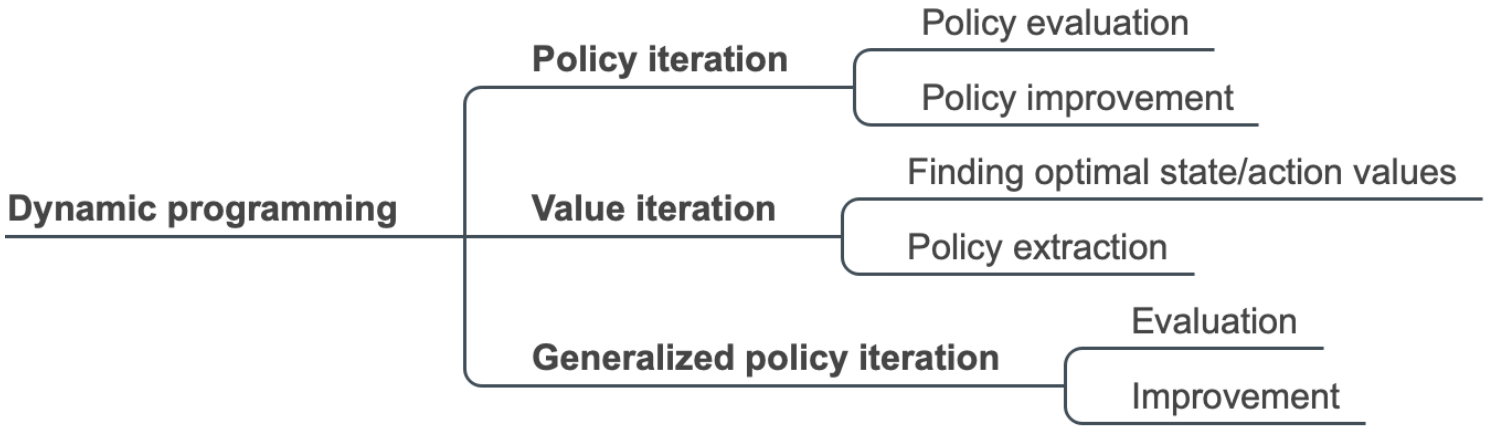
Board positions $\mathcal{S}$	Moves $\mathcal{A}$	Equations $q_*(s, a)$
2×10^{170}	250	500×10^{170}

Atoms in known observable universe $\sim 10^{80}$



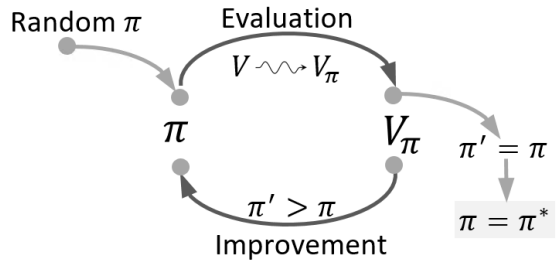
- Approximate solution approaches are needed

Image source: https://www.quantamagazine.org/is-alphago-really-such-a-big-deal-20160329/



Policy Iteration: (1) Policy Evaluation (PE); (2) Policy Improvement (PI)

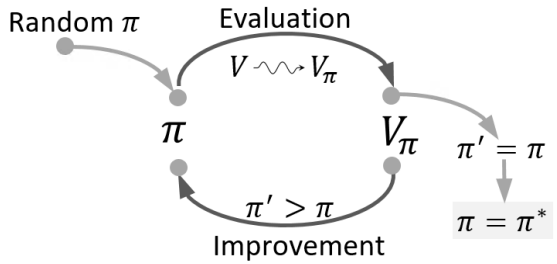
$$\pi_0 \xrightarrow{\text{PE}} v_{\pi_0} \xrightarrow{\text{PI}} \pi_1 \xrightarrow{\text{PE}} v_{\pi_1} \xrightarrow{\text{PI}} \pi_2 \xrightarrow{\text{PE}} \cdots \xrightarrow{\text{PI}} \pi_* \xrightarrow{\text{PE}} v_*$$



Value Iteration: Iterative algorithm based on Bellman Equation

$$\begin{cases} v_{k+1}(s) = \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma\, v_k(s') \right] \\ q_{k+1}(s,a) = \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_k(s', a') \right] \end{cases} \Rightarrow \begin{cases} v_* \\ q_* \end{cases}$$

$$\pi_0 \xrightarrow{\text{PE}} v_{\pi_0} \xrightarrow{\text{PI}} \pi_1 \xrightarrow{\text{PE}} v_{\pi_1} \xrightarrow{\text{PI}} \pi_2 \xrightarrow{\text{PE}} \cdots \xrightarrow{\text{PI}} \pi_* \xrightarrow{\text{PE}} v_*$$



1. Arbitrarily choose an initial policy π_0. Let $\pi_k = \pi_0$
2. *Policy Evaluation*: Evaluate v_{π_k}
3. *Policy Improvement*: Find (possibly new) $\pi_{k+1} \geq \pi_k$
4. If $\pi_{k+1} = \pi_k$, then stop with $\pi_* = \pi_k$

   If $\pi_{k+1} > \pi_k$, then set $\pi_k \leftarrow \pi_{k+1}$; go to Step 2

Solution Approach: Estimate $v_\pi(s)$ iteratively

Input

π	Policy to be evaluated
θ	A small threshold
$V_0(s) = 0$	Set arbitrary initial values for non-terminal states State values are zero for all terminal states by definition

Repeat $\Delta \leftarrow 0$

Loop through all $s \in \mathcal{S}$

$v \leftarrow V(s)$

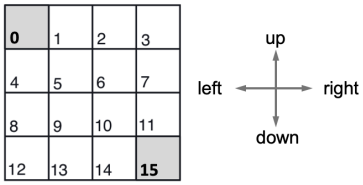
$$\underbrace{V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma V(s') \Big]}_{\text{Variation of Bellman Equation for } v_\pi}$$

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

until $\Delta < \theta$

Output $V(s)$, which is the estimated value of $v_\pi(s)$

Policy Evaluation: Given π, Determine $v_\pi(s)$ for All $s \in \mathcal{S}$



$\gamma = 1$

$k = 0$

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

Random policy:

$$\pi(\text{up/down/left/right} \mid 5) = 0.25$$

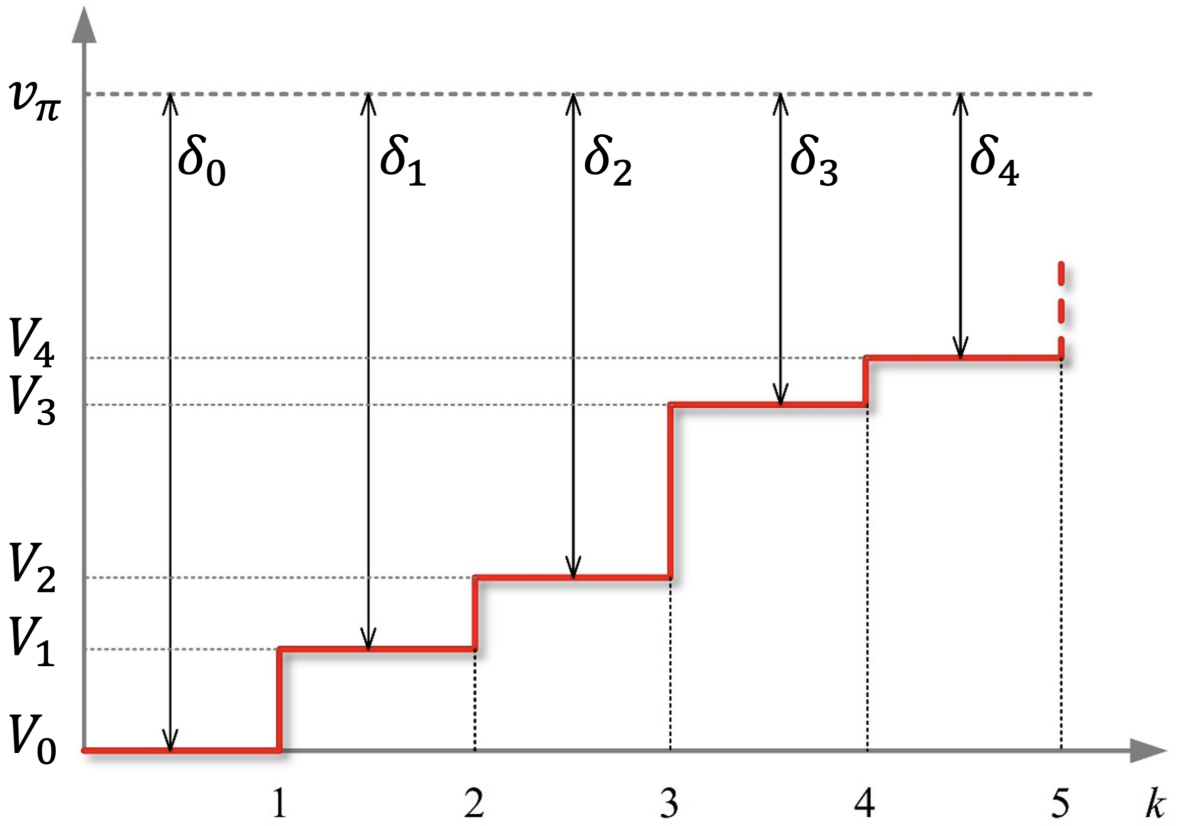
Taking any action results in $r = -1$

V_k denotes the estimate of v_π at iteration k

$$
\begin{aligned}
V_1(5) &= \sum_a \pi(a \mid s) \sum_{s',\, r} p(s', r \mid s, a)\,[r + \gamma\, V_0(s')] \\
&= \pi(\text{up} \mid 5) \cdot p(1, -1 \mid 5, \text{up}) \cdot [-1 + \gamma\, V_0(1)] \\
&+ \pi(\text{right} \mid 5) \cdot p(6, -1 \mid 5, \text{right}) \cdot [-1 + \gamma\, V_0(6)] \\
&+ \pi(\text{left} \mid 5) \cdot p(4, -1 \mid 5, \text{left}) \cdot [-1 + \gamma\, V_0(4)] \\
&+ \pi(\text{down} \mid 5) \cdot p(9, -1 \mid 5, \text{down}) \cdot [-1 + \gamma\, V_0(9)] \\
&= 0.25 \cdot 1 \cdot [-1 + 1 \cdot 0] \times 4 \\
&= -1.0
\end{aligned}
$$

$$
\begin{aligned}
V_2(5) &= 0.25 \cdot 1 \cdot [-1 + 1 \cdot V_1(1)] \\
&+ 0.25 \cdot 1 \cdot [-1 + 1 \cdot V_1(6)] \\
&+ 0.25 \cdot 1 \cdot [-1 + 1 \cdot V_1(4)] \\
&+ 0.25 \cdot 1 \cdot [-1 + 1 \cdot V_1(9)] \\
&= 0.25 \cdot 1 \cdot [-1 + 1 \cdot (-1)] \times 4 \\
&= -2.0
\end{aligned}
$$

$$
\begin{aligned}
V_3(5) &= 0.25 \cdot 1 \cdot [-1 + 1 \cdot V_2(1)] \\
&+ 0.25 \cdot 1 \cdot [-1 + 1 \cdot V_2(6)] \\
&+ 0.25 \cdot 1 \cdot [-1 + 1 \cdot V_2(4)] \\
&+ 0.25 \cdot 1 \cdot [-1 + 1 \cdot V_2(9)] \\
&= 0.25 \cdot 1 \cdot [-1 + 1 \cdot (-1.7)] \times 2 \\
&+ 0.25 \cdot 1 \cdot [-1 + 1 \cdot (-2.0)] \times 2 \\
&= -2.85 \approx -2.9
\end{aligned}
$$



$$\delta_k \triangleq \max_s |v_\pi(s) - V_k(s)|$$

$$\delta_k \to 0 \quad \text{as} \quad k \to \infty$$

$$\delta_k \triangleq \max_s |v_\pi(s) - V_k(s)|$$

$$\Rightarrow \quad |v_\pi(s) - V_k(s)| \le \delta_k$$

$$-\delta_k \le v_\pi(s) - V_k(s) \le \delta_k$$

$$\Rightarrow \quad -v_\pi(s) - \delta_k \le -V_k(s) \le -v_\pi(s) + \delta_k$$

$$\Rightarrow \quad v_\pi(s) + \delta_k \ge V_k(s) \ge v_\pi(s) - \delta_k$$

$$\Rightarrow \quad v_\pi(s) - \delta_k \le V_k(s) \le v_\pi(s) + \delta_k$$

$$\begin{aligned}
V_{k+1}(s) &= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \big[r + \gamma V_k(s') \big] \\
&\ge \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \big[r + \gamma \, \big(v_\pi(s') - \delta_k \big) \big] \\
&= \underbrace{\sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \big[r + \gamma \, v_\pi(s') \big]}_{\text{This is } v_\pi(s)} - \gamma \, \delta_k \\
&= v_\pi(s) - \gamma \, \delta_k
\end{aligned}$$

$$v_\pi(s) - \gamma \, \delta_k \le V_{k+1}(s) \qquad \text{or} \qquad v_\pi(s) - V_{k+1}(s) \le \gamma \, \delta_k$$

$$\delta_{k+1} \triangleq \max_s |v_\pi(s) - V_{k+1}(s)| \le \max_s |\gamma \, \delta_k| = \gamma \, \delta_k$$

$$\delta_{k+1} \le \gamma \, \delta_k$$

If choose new π' such that $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, then $v_{\pi'}(s) \geq v_\pi(s)$

Proof: We only consider *deterministic* policies here

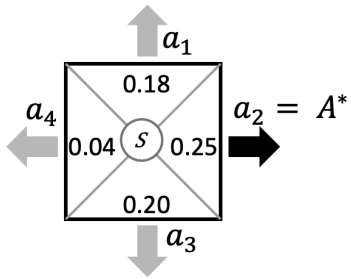
back

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}\left[R_{t+1} + \gamma\, v_\pi(S_{t+1}) \,\middle|\, S_t = s, A_t = \pi'(s)\right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma\, v_\pi(S_{t+1}) \,\middle|\, S_t = s\right] \qquad \leftarrow (\dagger) \\
&\leq \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma \underbrace{q_\pi\left(S_{t+1}, \pi'(S_{t+1})\right)}_{\text{From condition on } q_\pi \text{ and } v_\pi} \,\Big|\, S_t = s\Big] \\
&= \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma \underbrace{\mathbb{E}_{\pi'}\left[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1}\right]}_{\text{Bellman Equation for } q_\pi;\ A_{t+1} = \pi'(S_{t+1})} \,\Big|\, S_t = s\Big] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma\, R_{t+2} + \gamma^2\, v_\pi(S_{t+2}) \,\middle|\, S_t = s\right] \qquad \leftarrow (\ddagger) \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma\, R_{t+2} + \gamma^2\, R_{t+3} + \gamma^3\, v_\pi(S_{t+3}) \,\middle|\, S_t = s\right] \leftarrow \begin{array}{l}\text{repeat} \\ (\dagger) \text{ to } (\ddagger)\end{array} \\
&\ \ \vdots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma\, R_{t+2} + \gamma^2\, R_{t+3} + \gamma^3\, R_{t+4} + \cdots \,\middle|\, S_t = s\right] \\
&= v_{\pi'}(s)
\end{aligned}
$$

We can just modify π to produce a better policy π'

Simply selecting at s the action that looks best (in terms of one step look-ahead) according to the current values of $v_\pi(s)$

$$\begin{aligned}\pi'(s) &\triangleq \operatorname*{argmax}_a q_\pi(s,a) \\ &= \operatorname*{argmax}_a \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma\, v_\pi(s')\right]\end{aligned}$$



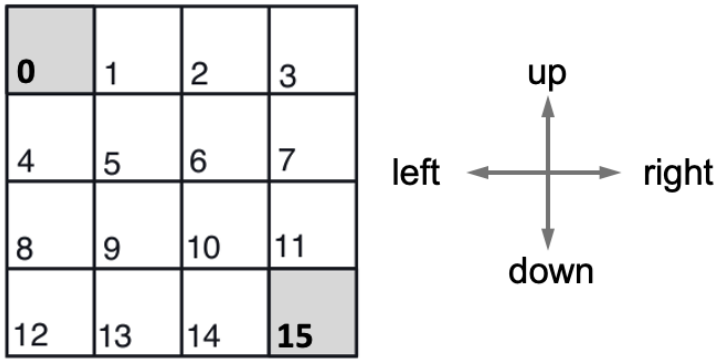
This new policy π' is called a *greedy policy*

We next check the condition: $\boxed{q_\pi\left(s,\pi'(s)\right) \geq v_\pi(s)}$

- With the way $\pi'(s)$ is chosen, we have: $q_\pi\left(s,\pi'(s)\right) \geq q_\pi\left(s,\pi(s)\right)$
- For deterministic policies: $q_\pi(s,\pi(s)) = v_\pi(s)$
- Therefore

$$q_\pi\left(s,\pi'(s)\right) \geq q_\pi(s,\pi(s)) = v_\pi(s) \quad \checkmark$$

Policy Improvement Example: Modify π to get π'



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

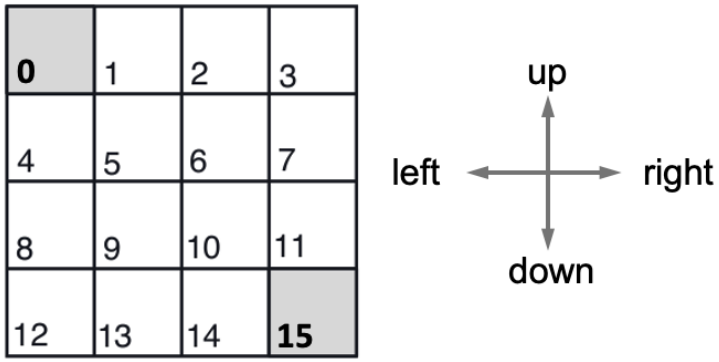
Assume:

$\gamma = 1$

$\pi(5) = \text{right}$

$$
\begin{aligned}
\pi'(s) &\triangleq \operatorname*{argmax}_a q_\pi(s,a) = \operatorname*{argmax}_a \sum_{s',r} p(s',r \mid s,a)\big[r + \gamma\, v_\pi(s')\big] \\
&= \operatorname*{argmax}_a \sum_{s'} p(s',-1 \mid 5,a)\big[-1 + 1 \cdot v_\pi(s')\big] \\
&= \operatorname*{argmax}_a \Big[\overline{f}(5,a,s')\,\big[-1 + 1 \cdot v_\pi(s')\big]\Big]_{s'=1,4,6,9} \\
&= \{\text{up}, \text{left}\}
\end{aligned}
$$

Policy Improvement Example: Checking Condition $q_\pi(s, \pi'(s)) \geq v_\pi(s)$



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

Assume:

$\gamma = 1$

$\pi(5) =$ right

$$
\begin{aligned}
q_{\pi'}(s,a) &= q_{\pi'}(5, \pi'(5)) = q_{\pi'}(5, \text{up}) = \sum_{s',\, r} p(s', r \,|\, s, a)\big[r + \gamma\, v_\pi(s')\big] \\
&= p(1, -1 \,|\, 5, \text{up}) \times \big[-1 + 1 \cdot v_\pi(1)\big] = 100\% \times \big[-1 - 6.1\big] \\
&= -7.1 > -7.7 = v_\pi(5) \qquad \text{(Same for } \pi'(5) = \text{left)}
\end{aligned}
$$

- Since π' is deterministic, we have $q_{\pi'}(5, \text{up}) = v_{\pi'}(5)$
- So $v_{\pi'}(5) > v_\pi(5)$, i.e., π' is better than π
- The decision to execute *up* or *left* can be made arbitrarily

$$\pi_0 \xrightarrow{\text{PE}} v_{\pi_0} \xrightarrow{\text{PI}} \pi_1 \xrightarrow{\text{PE}} v_{\pi_1} \xrightarrow{\text{PI}} \pi_2 \xrightarrow{\text{PE}} \cdots \xrightarrow{\text{PI}} \pi_* \xrightarrow{\text{PE}} v_*$$

- After each PE-PI cycle, we have $\pi'(s) \geq \pi(s)$
- If we reach a point where $\pi'(s) = \pi(s)$, then $v_\pi(s) = v_{\pi'}(s)$
- At each s we always pick 'greedy action' according to v_π, i.e.,

$$\pi'(s) \triangleq \operatorname*{argmax}_a q_\pi(s,a) = \operatorname*{argmax}_a \sum_{s',r} p(s',r \mid s,a)\big[r + \gamma\, v_\pi(s')\big]$$

- With $\pi'(s) = \pi(s)$ we have

$$v_{\pi'} = \max_a \sum_{s',r} p(s',r \mid s,a)\big[r + \gamma\, v_{\pi'}(s')\big]$$

which is the Bellman Optimality Equation with $v_* = v_{\pi'}$

- So $\pi' = \pi_*$, i.e., policy iteration always yields an optimal policy

Policy Evaluation (for deterministic policy)

$$V_{k+1}(s) \leftarrow \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma V_k(s') \Big]$$

Policy Improvement

$$v_{\pi'} = \max_a \sum_{s',\, r} p(s', r \mid s, a) \big[r + \gamma v_{\pi'}(s') \big]$$

Value Iteration: One time-step PE immediately followed by PI

$$V_{k+1}(s) = \max_a \sum_{s',\, r} p\left(s', r \mid s, a\right) \Big[r + \gamma V_k(s') \Big]$$

Bellman Equation for $v(s)$ as update rule

Policy Extraction

$$\pi_*(s) = \operatorname*{argmax}_a \sum_{s',\, r} p\left(s', r \mid s, a\right) \Big[r + \gamma v_*(s') \Big]$$

Input

θ a small threshold

$Q(s, a) = 0$ set arbitrary initial values for all non-terminal states

action values are zero for all terminal states by definition

Repeat $\Delta \leftarrow 0$

Loop through all pairs $(s, a) \in \mathcal{S} \times \mathcal{A}(s)$

$q \leftarrow Q(s, a)$

$$\underbrace{Q(s, a) \leftarrow \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma \max_{a'} Q(s', a') \Big]}_{\text{Bellman Equation for } q(s,a) \text{ as update rule}}$$

$\Delta \leftarrow \max \big(\Delta, |q - Q(s, a)| \big)$

until $\Delta < \theta$

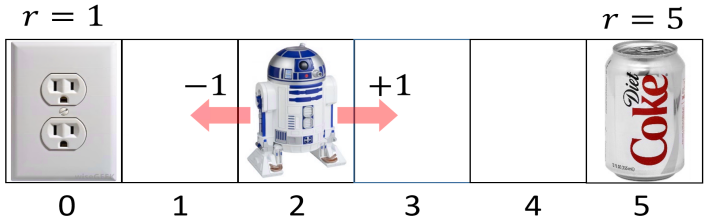
Output $Q(s, a)$, which the estimated value of $q_*(s, a)$

Policy $\pi_*(s) = \underset{a}{\operatorname{argmax}}\ q_*(s, a) = \underset{a}{\operatorname{argmax}}\ Q_k(s, a)$

Proof of convergence is provided in supplementary notes

Value Iteration on $q(s,a)$: Example

More details in supplementary notes



$$\gamma = 0.5$$

$$\bar{f}(s,a) = \begin{cases} s+a & \text{if } 1 \le s \le 4 \\ s & \text{if } s = 0 \text{ or } s = 5 \end{cases}$$

$$\rho(s,a,s') = \begin{cases} 5 & \text{if } s \neq 5 \text{ and } s' = 5 \\ 1 & \text{if } s \neq 0 \text{ and } s' = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} Q(s,a) &= \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma \max_{a'} Q(s',a')\right] \\ &= \rho(s,a,s') + \gamma \max_{a'} Q\left(\bar{f}(s,a),a'\right) \end{aligned}$$

$\boxed{k=1}$ (We do not index Q here because in this example Q update is asynchronous – see next slide)

$$\begin{aligned} Q(0,1) &= \rho(0,1,0) + 0.5 \max_{a'} Q\left(\bar{f}(0,1),a'\right) \quad \left(\text{Note: } \bar{f}(0,1) = 0\right) \\ &= \rho(0,1,0) + 0.5 \max\left[Q(0,-1), Q(0,1)\right] \\ &= 0 + 0.5 \cdot \max\left[0,0\right] = 0 \\ &\vdots \\ Q(1,-1) &= \rho(1,-1,0) + 0.5 \max_{a'} Q\left(\bar{f}(1,-1),a'\right) \quad \left(\text{Note: } \bar{f}(1,-1) = 0\right) \\ &= \rho(1,-1,0) + 0.5 \max\left[Q(0,-1), Q(0,1)\right] \\ &= 1 + 0.5 \cdot \max\left[0,0\right] = 1 \end{aligned}$$

$k = 1$ (Continued)

$$
\begin{aligned}
&\vdots \\
Q(1,1) &= \cdots \\
&= 0 \\
Q(2,-1) &= \rho(2,-1,1) + 0.5 \max_{a'} Q\left(\bar{f}(2,-1), a'\right) \quad \left(\text{Note: } \bar{f}(2,-1) = 1\right) \\
&= \rho(2,-1,1) + 0.5 \max\left[Q(1,-1), Q(1,1)\right] \\
&= 0 + 0.5 \cdot \max\left[1, 0\right] = 0.5 \\
Q(3,1) &= \cdots \\
&\vdots
\end{aligned}
$$

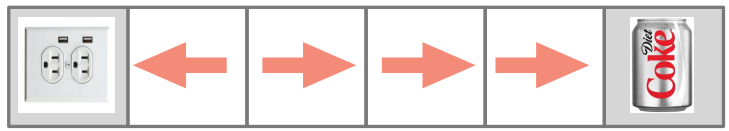
$k = 2$

$$
\begin{aligned}
Q(0,1) &= \cdots \\
Q(1,1) &= \rho(1,1,2) + 0.5 \max_{a'} Q\left(\bar{f}(1,1), a'\right) \\
&= \rho(1,1,2) + 0.5 \max\left[Q(2,-1), Q(2,1)\right] \\
&= 0 + 0.5 \cdot \max\left[0.5, 0\right] = 0.25
\end{aligned}
$$

Value Iteration on $q(s, a)$: Example (with deterministic transitions)

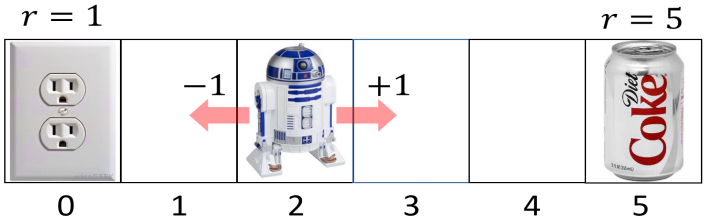
More details in supplementary notes

Action Value	State 0	1	2	3	4	5
Q_0	0.000|0.000	0.000|0.000	0.000|0.000	0.000|0.000	0.000|0.000	0.000|0.000
Q_1	0.000|0.000	1.000|0.000	0.500|0.000	0.250|0.000	0.125|5.000	0.000|0.000
Q_2	0.000|0.000	1.000|0.250	0.500|0.125	0.250|0.250	1.250|5.000	0.000|0.000
Q_3	0.000|0.000	1.000|0.250	0.500|1.250	0.625|2.500	1.250|5.000	0.000|0.000
Q_4	0.000|0.000	1.000|0.625	0.500|1.250	0.625|2.500	1.250|5.000	0.000|0.000
Q_5	0.000|0.000	1.000|0.625	0.500|1.250	0.625|2.500	1.250|5.000	0.000|0.000



Value Iteration on $q(s,a)$: Example (with probabilistic transitions)

More details in supplementary notes



$$\rho(s,a,s') = \begin{cases} 5 & \text{if } s \neq 5 \text{ and } s' = 5 \\ 1 & \text{if } s \neq 0 \text{ and } s' = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma = 0.5$$

(s,a)	$f(s,a,0)$	$f(s,a,1)$	$f(s,a,2)$	$f(s,a,3)$	$f(s,a,4)$	$f(s,a,5)$
$(0,-1)$	1	0	0	0	0	0
$(1,-1)$	0.8	0.15	0.05	0	0	0
$(2,-1)$	0	0.8	0.15	0.05	0	0
$(3,-1)$	0	0	0.8	0.15	0.05	0
$(4,-1)$	0	0	0	0.8	0.15	0.05
$(5,-1)$	0	0	0	0	0	1
$(0,1)$	1	0	0	0	0	0
$(1,1)$	0.05	0.15	0.8	0	0	0
$(2,1)$	0	0.05	0.15	0.8	0	0
$(3,1)$	0	0	0.05	0.15	0.8	0
$(4,1)$	0	0	0	0.05	0.15	0.8
$(5,1)$	0	0	0	0	0	1

$$\begin{aligned} Q(s,a) &= \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma \max_{a'} Q(s',a')\right] \\ &= \rho(s,a,s') + \gamma \max_{a'} Q\left(\bar{f}(s,a),a'\right) \end{aligned}$$

Value Iteration on $q(s,a)$: Example (with probabilistic transitions)

$\boxed{k=1}$ (We do not index Q here because in this example Q update is asynchronous – see next slide)

$$
\begin{aligned}
Q(0,-1) &= \cdots = 0 \\
Q(1,-1) &= \sum_{s'} P_{1s'}^{-1} \left(\rho(1,-1,s') + \gamma \max_{a'} Q\left(s',a'\right)\right) \\
&= \sum_{j=0}^{2} P_{1j}^{-1} \left(\rho(1,-1,j) + \gamma \max_{a'} Q\left(j,a'\right)\right) \\
&= P_{10}^{-1} \left(\rho(1,-1,0) + \gamma \max_{a'} Q\left(0,a'\right)\right) \quad (\text{Note: } j=0) \\
&+ P_{11}^{-1} \left(\rho(1,-1,1) + \gamma \max_{a'} Q\left(1,a'\right)\right) \quad (\text{Note: } j=1) \\
&+ P_{12}^{-1} \left(\rho(1,-1,2) + \gamma \max_{a'} Q\left(2,a'\right)\right) \quad (\text{Note: } j=2) \\
&= 0.8\big(1 + 0.5 \max\left[Q(0,-1), Q(0,1)\right]\big) \quad (\text{Note: } \rho(1,-1,0)=1\ ) \\
&+ 0.15\big(0 + 0.5 \max\left[Q(1,-1), Q(1,1)\right]\big) \\
&+ 0.05\big(0 + 0.5 \max\left[Q(2,-1), Q(2,1)\right]\big) \\
&= 0.8\big(1 + 0.5 \max\left[0,0\right]\big) + 0.15 \times 0.5 \max\left[0,0\right] \\
&+ 0.05 \times 0.5 \max\left[0,0\right] \\
&= 0.8
\end{aligned}
$$

Value Iteration on $q(s,a)$: Example (with probabilistic transitions)

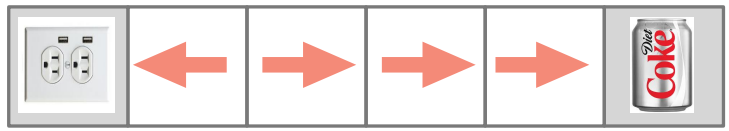
$k = 1$ (continued)

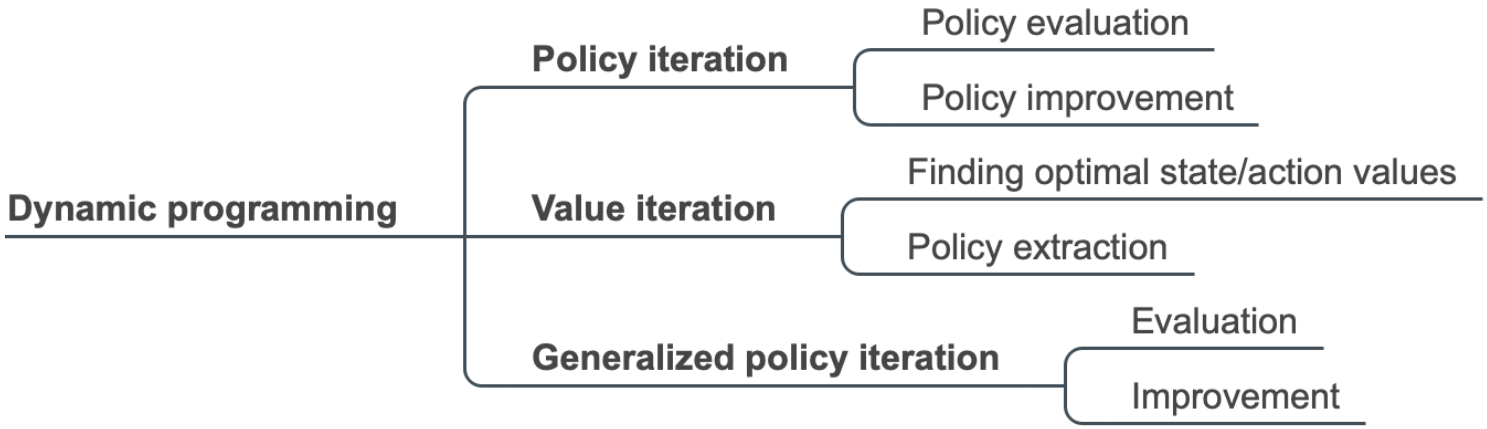
$$
\begin{aligned}
Q(0,-1) &= \cdots = 0 \\
Q(1,-1) &= 0.8 \\
Q(1,1) &= \sum_{s'} P^1_{1s'} \left(\rho(1,1,s') + \gamma \max_{a'} Q\left(s',a'\right)\right) \\
&= \sum_{j=0}^{2} P^1_{1j} \left(\rho(1,1,j) + \gamma \max_{a'} Q\left(j,a'\right)\right) \\
&= P^1_{10} \left(\rho(1,1,0) + \gamma \max_{a'} Q\left(0,a'\right)\right) \quad (\text{Note: } j=0) \\
&+ P^1_{11} \left(\rho(1,1,1) + \gamma \max_{a'} Q\left(1,a'\right)\right) \quad (\text{Note: } j=1) \\
&+ P^1_{12} \left(\rho(1,1,2) + \gamma \max_{a'} Q\left(2,a'\right)\right) \quad (\text{Note: } j=2) \\
&= 0.05 \cdot \left(1 + 0.5 \cdot \max\left[Q(0,-1), Q(0,1)\right]\right) \quad (\text{Note: } \rho(1,1,0)=1) \\
&+ 0.15 \cdot \left(0 + 0.5 \cdot \max\left[Q(1,-1), Q(1,1)\right]\right) \quad (\text{Note: } Q(1,-1)=0.8) \\
&+ 0.8 \cdot \left(0 + 0.5 \cdot \max\left[Q(2,-1), Q(2,1)\right]\right) \\
&= 0.05\left(1 + 0.5 \cdot \max\left[0,0\right]\right) + 0.15 \cdot 0.5 \cdot \max\left[0.8,0\right] \\
&+ 0.8 \cdot 0.5 \cdot \max\left[0,0\right] \\
&= 0.11
\end{aligned}
$$

Value Iteration on $q(s, a)$: Example (with probabilistic transitions)

More details in supplementary notes

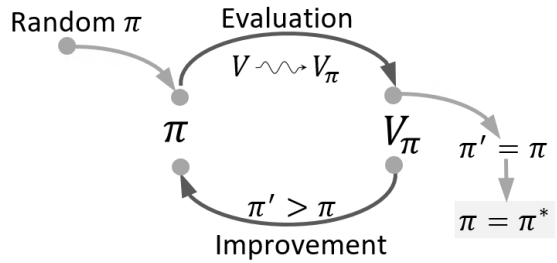
State	0	1	2	3	4	5
Q_0	0.000|0.000	0.000|0.000	0.000|0.000	0.000|0.000	0.000|0.000	0.000|0.000
Q_1	0.000|0.000	0.800|0.110	0.320|0.044	0.128|0.018	0.301|4.026	0.000|0.000
Q_2	0.000|0.000	8.868|0.243	0.374|0.101	0.260|1.639	1.208|4.343	0.000|0.000
Q_3	0.000|0.000	0.874|0.265	0.419|0.709	0.515|1.878	1.327|4.373	0.000|0.000
Q_4	0.000|0.000	0.883|0.400	0.453|0.826	0.581|1.911	1.342|4.376	0.000|0.000
...	...	...	...	...	...	...
Q_{12}	0.000|0.000	0.888|0.458	0.467|0.852	0.594|1.915	1.344|4.376	0.000|0.000
...	...	...	...	...	...	...
Q_{22}	0.000|0.000	0.888|0.458	0.467|0.852	0.594|1.915	1.344|4.376	0.000|0.000





Policy Iteration: (1) Policy Evaluation (PE); (2) Policy Improvement (PI)

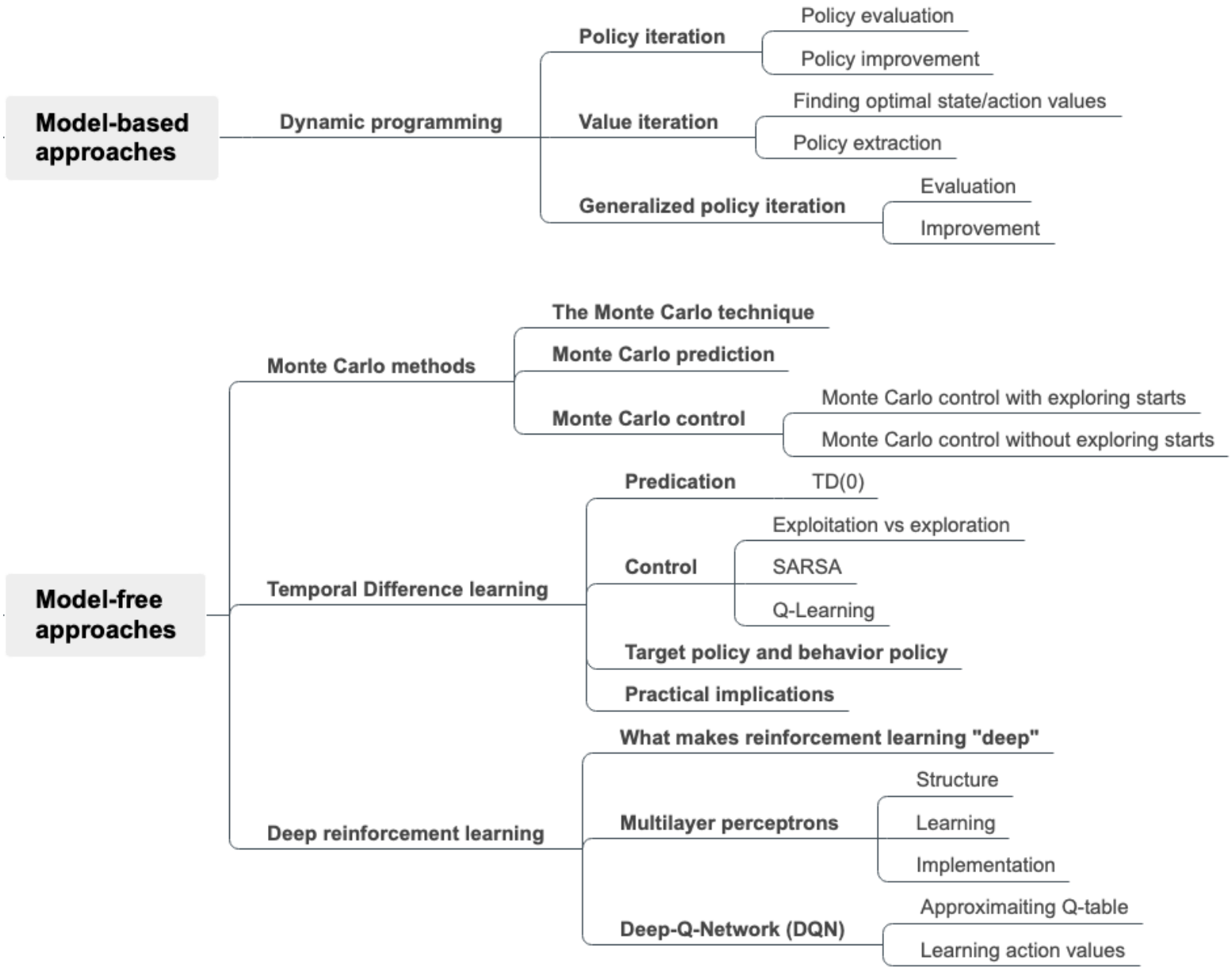
$$\pi_0 \xrightarrow{\text{PE}} v_{\pi_0} \xrightarrow{\text{PI}} \pi_1 \xrightarrow{\text{PE}} v_{\pi_1} \xrightarrow{\text{PI}} \pi_2 \xrightarrow{\text{PE}} \cdots \xrightarrow{\text{PI}} \pi_* \xrightarrow{\text{PE}} v_*$$



Value Iteration: Iterative algorithm based on Bellman Equation

$$\begin{cases} V_{k+1}(s) = \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V_k(s') \right] \\ Q_{k+1}(s,a) = \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} Q_k(s', a') \right] \end{cases} \Rightarrow \begin{cases} v_* \\ q_* \end{cases}$$

Monte Carlo Methods



Monte Carlo technique

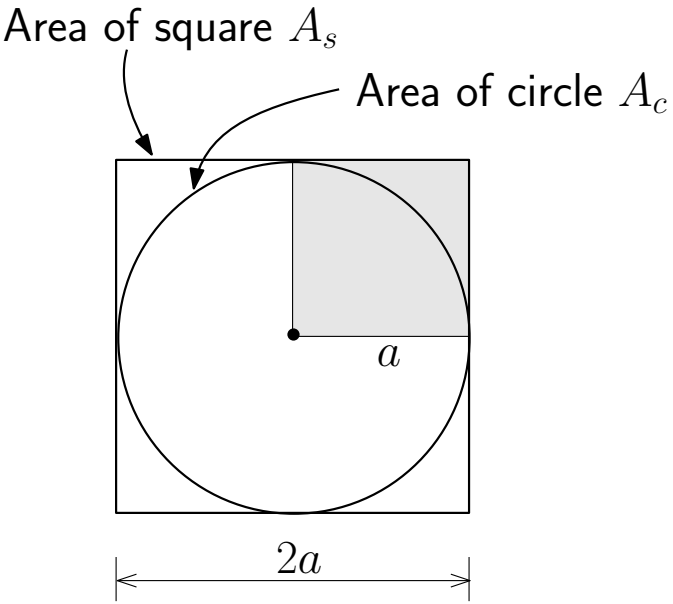
Mathematical technique that uses random sampling to generate numerical results

Monte Carlo methods

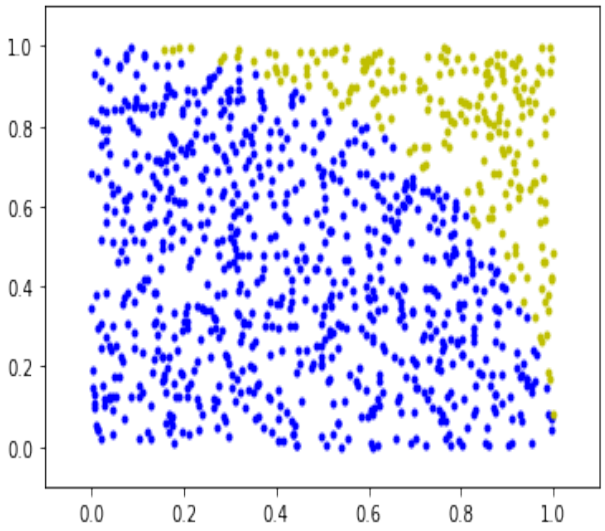
Methods used for reinforcement learning that are based on the Monte Carlo technique

Estimate π (the constant, not a policy) by throwing darts at shaded area n times and count m hits in quadrant

$$\frac{A_s}{A_c} = \frac{(2a)^2}{\pi a^2} = \frac{4}{\pi} \quad \Rightarrow \quad \pi \approx 4\left(\frac{m}{n}\right)$$



Circle enclosed by smallest square



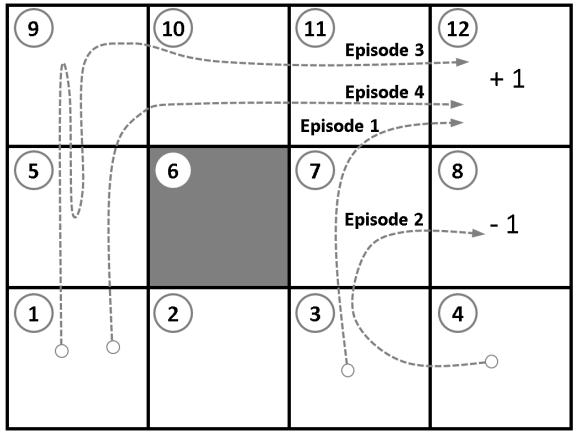
Points in shaded square yielding $\pi = \frac{4(790)}{1000} = 3.16$

$\mathcal{S}$ and $\mathcal{A}$ are known, but no knowledge about $p(s', r|s, a)$

Use sampled data from *experience* to estimate $v(s)$ or $q(s, a)$

Experience

Agent follows (possibly random) policies to move from some states to terminal state while collecting rewards along the way



Monte Carlo prediction

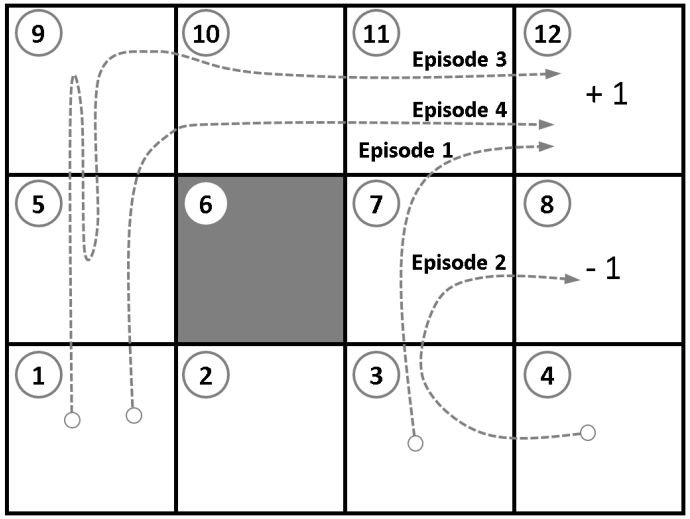
Find v_π or $q_\pi(s, a)$ for given π

(similar to Policy Evaluation)

Monte Carlo control

Find an optimal policy π_*

(similar to Policy Iteration)



Sequence of steps, each described by {state, action, reward}

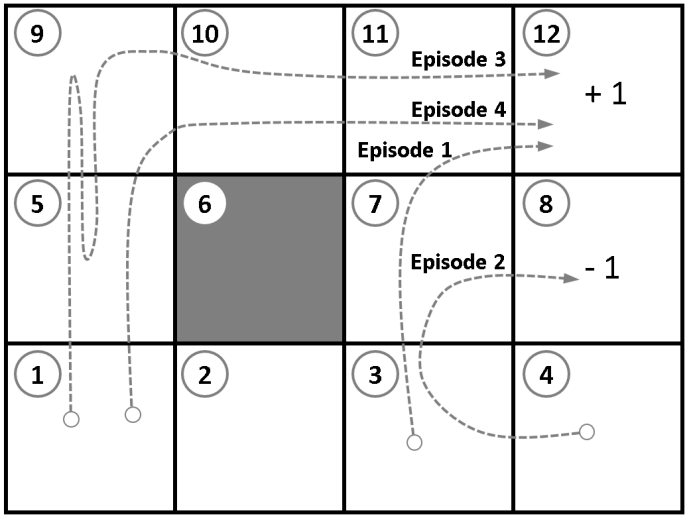
t^{th} step: $\{S_t, A_t, R_{t+1}\}$

Episode with T steps starting from S_0 and terminates at S_{T-1}

$$\left\{S_t, A_t, R_{t+1}\right\}_{t=0}^{T-1} = \left\{S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T\right\}$$

Example: Assume reward $r = -0.04$ for each step except when reaching terminal state. $T = 7$ in Episode 3

$$\left\{S_t, A_t, R_{t+1}\right\}_{t=0}^{6} = \{1, \text{up}, -0.04,\ 5, \text{up}, -0.04,\ 9, \text{down}, -0.04,$$
$$5, \text{up}, -0.04, 9, \text{right}, -0.04,\ 10, \text{right}, -0.04,$$
$$11, \text{right}, +1\}$$

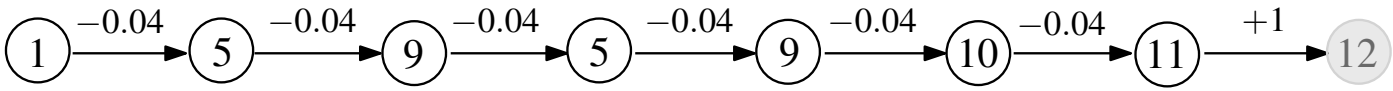


When episode terminates, "look back" and calculate return G_t for each state $S_t = s$ visited by the agent during episode, i.e.,

$$G_t = \sum_{k=0}^{T-t-1} \left(\gamma^k R_{t+k+1} \,\middle|\, S_t = s \right)$$

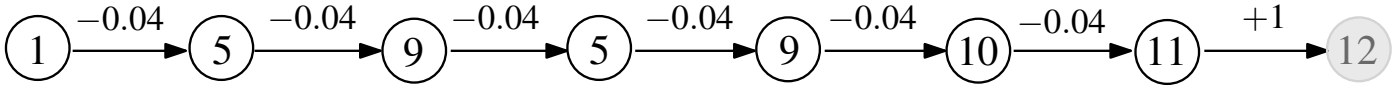
Example: Episode 3, $\gamma = 0.9$, $S_t = 1$, $t = 0$

$$G_t|_{S_t=1} = \left(\sum_{k=0}^{7-0-1} \gamma^k R_{0+k+1} \,\middle|\, S_t = 1 \right) = \sum_{k=0}^{6} \gamma^k R_{k+1}$$



$$G_t\big|_{S_t=1} = 0.9^0 \cdot (-0.04) + 0.9^1 \cdot (-0.04) + 0.9^2 \cdot (-0.04) + 0.9^3 \cdot (-0.04) + 0.9^4 \cdot (-0.04) + 0.9^5 \cdot (-0.04) + 0.9^6 \cdot (+1)$$

Calculating Return of State in Episode



$$G_t\big|_{S_t=1} = 0.9^0\cdot(-0.04)+0.9^1\cdot(-0.04)+0.9^2\cdot(-0.04)+0.9^3\cdot(-0.04)+0.9^4\cdot(-0.04)+0.9^5\cdot(-0.04)+0.9^6\cdot(+1)$$

$$G_t = \sum_{k=0}^{T-t-1}\left(\gamma^k R_{t+k+1}\,\middle|\,S_t=s\right) = R_{t+1}+\gamma\,G_{t+1}$$

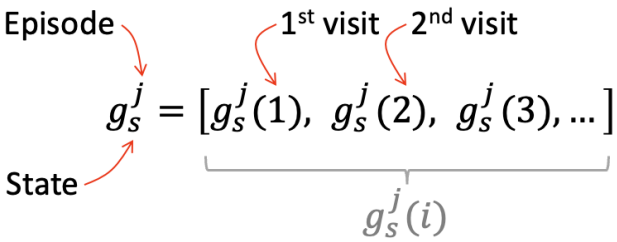
$$\begin{aligned}
G_6\,|_{S_6=11} &= R_7+\gamma\,G_7 = +1+0.9\times 0 = +1\\
G_5\,|_{S_5=10} &= R_6+\gamma\,G_6 = -0.04+0.9\times 1 = 0.86\\
G_4\,|_{S_4=9} &= R_5+\gamma\,G_5 = -0.04+0.9\times 0.86 = 0.82\\
G_3\,|_{S_3=5} &= R_4+\gamma\,G_4 = -0.04+0.9\times 0.82 = 0.70\\
G_2\,|_{S_2=9} &= R_3+\gamma\,G_3 = -0.04+0.9\times 0.70 = 0.59\\
G_1\,|_{S_1=5} &= R_2+\gamma\,G_2 = -0.04+0.9\times 0.59 = 0.49\\
G_0\,|_{S_0=1} &= R_1+\gamma\,G_1 = -0.04+0.9\times 0.49 = 0.40
\end{aligned}$$

Estimation of $v_\pi(s)$ over m Episodes

A state s may be visited multiple times during some or all episodes

For visit i occurring at t during episode j, return for s is $G_t|_{S_t=s}$, which is stored as i^{th} element of list g_s^j, i.e., $g_s^j(i) = G_t|_{S_t=s}$

For example, in episode 3, state 5 is visited twice. So



$$\left.\begin{aligned} g_5^3(1) &= G_1|_{S_1=5} = 0.49 \\ g_5^3(2) &= G_3|_{S_3=5} = 0.70 \end{aligned}\right\} \Rightarrow \quad g_5^3 = [0.49, 0.70]$$

For m episodes, append non-empty g_s^j to form g_s

Append operation: $[\![(\cdot),(\cdot)]\!]$

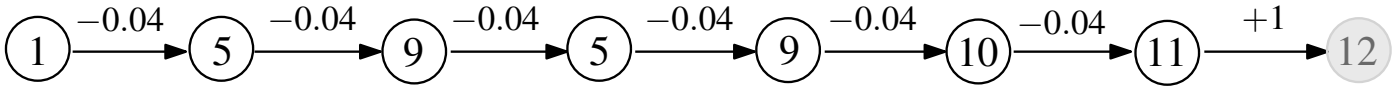
If $g_s^1 = [a]$ and $g_s^2 = [b]$, then

$$[\![g_s^1, g_s^2]\!] = [a, b]$$

$$\begin{aligned} V_\pi(s) &= \frac{\text{total return starting from } s \text{ and following } \pi \text{ over } m \text{ episodes}}{\text{number of visits to } s \text{ over } m \text{ episodes}} \\ &= \frac{\sum_{i=1}^{|g_s|} g_s(i)}{|g_s|} \qquad (\text{where } |g_s| \text{ is the number of elements in } g_s) \end{aligned}$$

By *Law of Large Numbers*, $V_\pi(s) \to v_\pi(s) \triangleq \mathbb{E}_\pi\left[G_t \,|\, S_t = s\right]$ as $m \to \infty$

A state s may be visited multiple times during an episode j



$$G_t\big|_{S_t=1} = 0.9^0\cdot(-0.04)+0.9^1\cdot(-0.04)+0.9^2\cdot(-0.04)+0.9^3\cdot(-0.04)+0.9^4\cdot(-0.04)+0.9^5\cdot(-0.04)+0.9^6\cdot(+1)$$

Many-visit

$$g_5^3 = [G_1, G_3] = [0.49, 0.7]$$

First-visit

$$g_5^3 = [G_1] = [0.49]$$

$$g_5 = [\![\ldots, g_5^3, \ldots]\!]$$

$$V_\pi(5) = \frac{\sum_{i=1}^{|g_5|} g_5(i)}{|g_5|}$$

Both versions result in $V_\pi(s) \to v_\pi(s)$ as $m \to \infty$

Input π, the policy to be evaluated

Initialize $V(s) \in \mathbb{R}$ to any arbitrary values, for all $s \in \mathcal{S}$

Returns(s) as an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode)

Generate an episode with T steps following π:

$S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

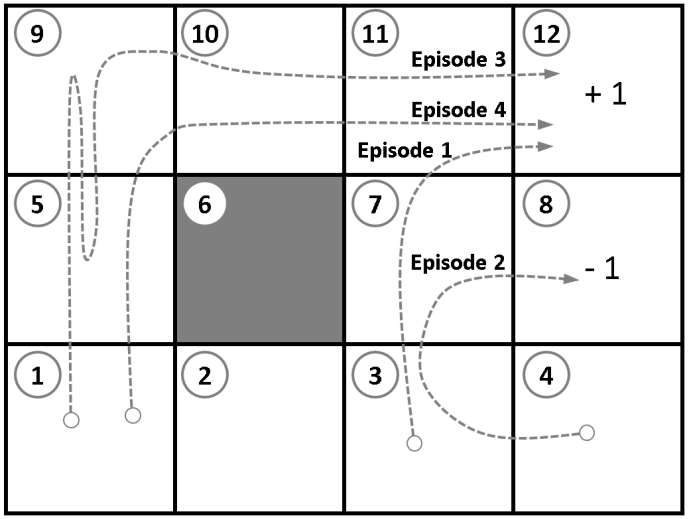
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in state set of episode $\{S_0, S_1, \ldots, S_{t-1}\}$

Append G to Return(S_t)

$V(S_t) \leftarrow$ average(Return(S_t))



Determine the values of $V_\pi(s)$ after completing Episodes 1 and 2

No state is visited more than once in either episode. So every-visit and first-visit are equivalent in this case

Let $[\,]$ = empty list, and $\gamma = 0.9$

Initialization: $V = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

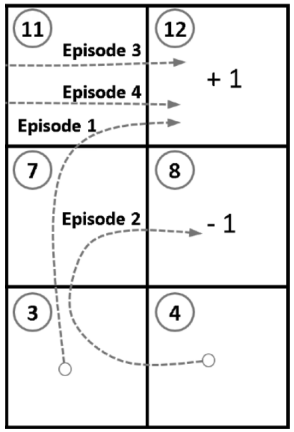
Episode 1

$$\begin{array}{lll} S_0 = 3 & A_0 = \text{up} & R_1 = -0.04 \\ S_1 = 7 & A_1 = \text{up} & R_2 = -0.04 \\ S_2 = 11 & A_2 = \text{right} & R_3 = +1 \end{array}$$

$$\begin{aligned} G_2|_{S_2=11} &= R_3 + \gamma\, G_3 = +1 + 0.9 \times 0 = +1 \\ G_1|_{S_1=7} &= R_2 + \gamma\, G_2 = -0.04 + 0.9 \times (+1) = 0.86 \\ G_0|_{S_0=3} &= = R_1 + \gamma G_1 = -0.04 + 0.9 \times 0.86 = 0.73 \end{aligned}$$

$g_3^1 = [0.73]$, $g_7^1 = [0.86]$, $g_{11}^1 = [1]$, and $g_s^1 = [\,]$ for all other states

$$V = \begin{bmatrix} 0 & 0 & 0.73 & 0 & 0 & 0 & -0.86 & 0 & 0 & 0 & 1.00 & 0 \end{bmatrix}$$



Episode 2: $S_0 = 4$ $A_0 = \text{left}$ $R_1 = -0.04$
$S_1 = 3$ $A_1 = \text{up}$ $R_2 = -0.04$
$S_2 = 7$ $A_2 = \text{right}$ $R_3 = -1$

$$
\begin{aligned}
G_2|_{S_2=7} &= R_3 + \gamma G_3 = -1 + 0.9 \times 0 = -1 \\
G_1|_{S_1=3} &= R_2 + \gamma G_2 = -0.04 + 0.9 \times (-1) = -0.90 \\
G_0|_{S_0=4} &= R_1 + \gamma G_1 = -0.04 + 0.9 \times (-0.9) = -0.85
\end{aligned}
$$

$g_3^2 = [-0.90],\ g_4^2 = [-0.85],\ g_7^2 = [-1],$ and $g_s^2 = [\,]$ for all other states

States 4 and 11 appear in Episode 1 only, but states 3 and 7 in both. So

$$g_3 = [\![g_3^1,\ g_3^2]\!] = [0.73,\ -0.90]\,, \qquad g_7 = [\![g_7^1,\ g_7^2]\!] = [0.86,\ -1]$$

$$
\begin{aligned}
V(3) &= = \frac{\sum_{i=1}^{|g_3|} g_3(i)}{|g_3|} = \frac{0.73 - 0.90}{2} = -0.09 \\
V(7) &= = \frac{\sum_{i=1}^{|g_7|} g_7(i)}{|g_7|} = \frac{0.86 - 1}{2} = -0.07
\end{aligned}
$$

Hence, after Episodes 1 and 2, we have

$$V = \begin{bmatrix} 0 & 0 & -0.09 & -0.85 & 0 & 0 & -0.07 & 0 & 0 & 0 & 1.00 & 0 \end{bmatrix}$$

Model-Based Policy Iteration

- Start with initial π and evaluate $v_\pi(s)$
- Improve π to obtain new policy $\pi' \geq \pi$

$$\pi'(s) \triangleq \underset{a}{\operatorname{argmax}} \sum_{s', r} p(s', r \,|\, s, a) \big[r + \gamma \, v_\pi(s') \big]$$

This requires $p(s', r \,|\, s, a)$

Monte Carlo Method for Model-Free Policy Iteration

1. Use Monte Carlo technique to estimate transition and reward functions — Intractable for large $\mathcal{S} \times \mathcal{A}$
2. Apply Monte Carlo prediction to estimate $q_\pi(s, a)$ directly, then determine improved policy

$$\pi'(s) \triangleq \underset{a}{\operatorname{argmax}} \sum_{s', r} p(s', r \,|\, s, a) \big[r + \gamma \, v_\pi(s') \big] = \underset{a}{\operatorname{argmax}} \; q_\pi(s, a)$$

We need to show that $\pi' \geq \pi$

Invoke Policy Improvement Theorem (with deterministic policies): view

$$\begin{array}{ll} \text{IF} & \pi' \text{ satisfies } q_\pi\big(s,\, \pi'(s)\big) \geq v_\pi(s) \\ \text{THEN} & v_{\pi'}(s) \geq v_\pi(s) \end{array}$$

The *IF clause* can be satisfied by making π' greedy with respect to q_π

$$\begin{aligned} q_\pi\big(s,\, \pi'(s)\big) &= q_{\pi_k}\Big(s,\, \operatorname*{argmax}_a q_\pi(s,a)\Big) \\ &= \max_a q_\pi(s,a) \geq q_\pi(s,\pi(s)) = v_\pi(s) \end{aligned}$$

Hence, with

$$\pi'(s) \triangleq \operatorname*{argmax}_a q_\pi(s,a)$$

we have $v_{\pi'}(s) \geq v_\pi(s)$. Consequently, $\pi' \geq \pi$

How to determine $q_\pi(s,a)$ without knowing $p(s', r \,|\, s, a)$?

Monte Carlo Control: How to Estimate q_π When $p(s', r|s, a)$ is not available

We can apply Monte Carlo technique to estimate $p(s', r \mid s, a)$

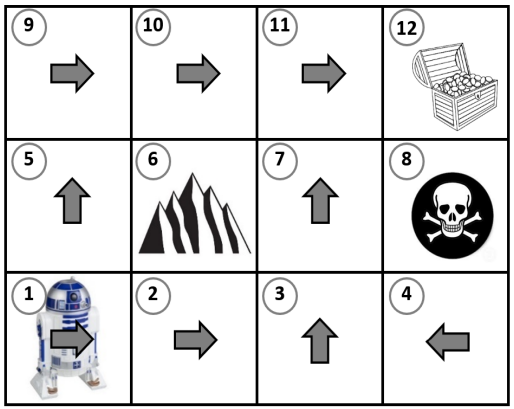
- Involves estimating *all* the transition probabilities
- Computationally intractable for large $\mathcal{S}$ and $\mathcal{A}$

Monte Carlo prediction can be readily applied to estimate $q_\pi(s, a)$, with "visits to s" replaced by "visits to (s, a)"

$Q_\pi(s, a) \to q_\pi(s, a)$ requires

(i) infinite number of episodes, and

(ii) all (s, a) having non-zero probability of being visited



If policy is deterministic, agent can visit **only one action** at each state

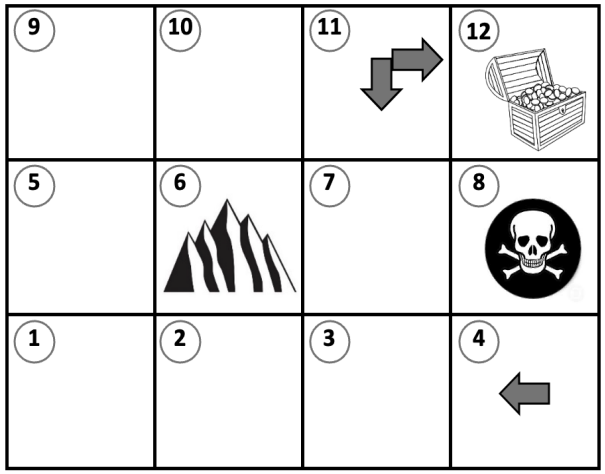
$Q_\pi(s, a) \to q_\pi(s, a)$ requires

(i) infinite number of episodes, and

(ii) all (s, a) having non-zero probability of being visited

Monte Carlo Control with Exploring Starts (Monte Carlo ES)

Assume that every state-action pair has a non-zero probability of being selected at the start of an episode



Episode	Start at	Follow	To obtain
1	$(11, \text{right})$	π_0	$\pi_1 \geq \pi_0$
2	$(4, \text{left})$	π_1	$\pi_2 \geq \pi_1$
3	$(11, \text{down})$	π_2	$\pi_3 \geq \pi_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Initialize

- $\pi(s) \in \mathcal{A}(s)$, arbitrarily, for all $s \in \mathcal{S}$ ($|\mathcal{A}(s)|$ is number of actions at s)
- $Q(s,a) \in \mathbb{R}$, arbitrarily (usually 0), for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
- Returns (s,a) as an empty list, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop forever (for each episode)

Select $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ randomly so that every (s,a) pair has non-zero probability of being selected

Generate episode with T steps starting from (S_0, A_0) following π

$S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$

$G \leftarrow \gamma G + R_{t+1}$

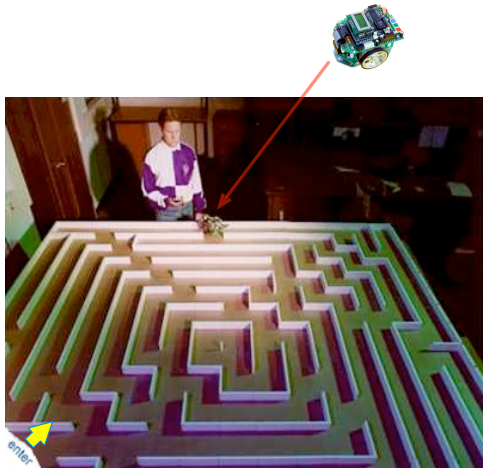
Unless pair (S_t, A_t) appears in $S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}$

Append G to Return(S_t, A_t)

$Q(S_t, A_t) \leftarrow$ average(Return(S_t, A_t))

$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$

- Monte Carlo ES assumes agent can start episode from any (s, a)
- Unrealistic assumption for many types of applications



In microrobot maze competition, a robot is to navigate through a maze starting from a *given location*

To get around this assumption, assign for every (s, a) pair a non-zero probability of being visited by agent during an episode:

ϵ-soft policy : $\pi(a \mid s) \geq \frac{\epsilon}{|\mathcal{A}(s)|} > 0\,, \quad 0 < \epsilon \leq 1$

$|\mathcal{A}(s)|$ is the number of actions at s

With ϵ-soft policy π, how do we find $\pi' \geq \pi$?

Action with $\max q_\pi(s,a)$ is called the greedy action at s

$$A^* \triangleq \operatorname*{argmax}_a q_\pi(s,a) \in \mathcal{A}(s)$$

Assign higher probability to A^* to make an ϵ-soft policy greedy

ϵ-greedy policy

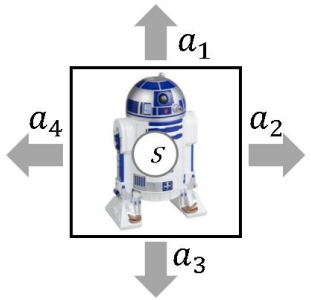
Assign $\epsilon/|\mathcal{A}(s)|$ to each non-greedy action and remaining probability to A^*, where $0 < \epsilon \leq 1$ and $|\mathcal{A}(s)|$ is total number of actions at s

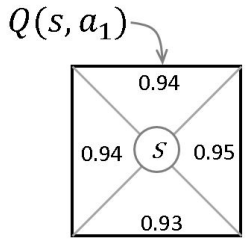
The probability of A^* being selected at s is

$$1 - \frac{\epsilon}{|\mathcal{A}(s)|} \underbrace{\Big(|\mathcal{A}(s)| - 1\Big)}_{\#\text{ of non-greedy actions}} = 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$$

ϵ-greedy policy

$$\pi(a \mid s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{for } a = A^* \\ \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{for } a \neq A^* \end{cases}$$





$A^* = a_2$

By definition, ϵ-greedy policy is ϵ-soft policy

Let $\epsilon = 0.1$

$$\pi(a \mid s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{|\mathcal{A}(s)|} = 1 - 0.1 + \dfrac{0.1}{4} = 0.925 & \text{for } a = a_2 \\ \dfrac{\epsilon}{|\mathcal{A}(s)|} = \dfrac{0.1}{4} = 0.025 & \text{for } a \neq a_2 \end{cases}$$

Check: Since the number of non-greedy actions here is 3, we have: *total probability* $= 0.925 + 0.025 \times 3 = 100\%$

Make π greedy with respect to $q_\pi(s,a)$ to obtain π'

$$\pi'(a \,|\, s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{for } a = A^* \triangleq \operatorname{argmax}_a q_\pi(s,a) \\ \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{for } a \neq A^* \end{cases}$$

Once again, invoke *Policy Improvement Theorem*

$$\text{If } \; q_{\pi'}(s,a) \geq v_\pi(s), \quad \text{then } \; \pi' \geq \pi$$

We need to show that with ϵ-greedy policy π'

$$q_{\pi'}(s,a) \geq v_\pi(s)$$

Let π' be an ϵ-greedy policy with respect to $q_\pi(s,a)$. Now

$$\begin{aligned} q_{\pi'}(s,a) &= \sum_a \pi'(a \mid s)\, q_\pi(s,a) \\ &= \underbrace{\frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a)}_{\text{for all actions, incl. } A^*} + \underbrace{(1-\epsilon) \max_a q_\pi(s,a)}_{\text{extra for } A^* \text{ only}} \end{aligned}$$

Proposition

$$\max_a q_\pi(s,a) \geq \sum_a \left[\left(\frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|}}{1-\epsilon} \right) q_\pi(s,a) \right]$$

Monte Carlo Control *without* Exploring Starts: Proof of Proposition

Let $w_a = \dfrac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|}}{1-\epsilon}$

Then

$$\sum_a w_a = \sum_a \left(\frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|}}{1-\epsilon} \right)$$

$$\begin{aligned}
\sum_a \left[\left(\frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|}}{1-\epsilon} \right) q_\pi(s,a) \right] &= \frac{1}{1-\epsilon} \sum_a \left(\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|} \right) \\
= \sum_a w_a \, q_\pi(s,a) &= \frac{1}{1-\epsilon} \Bigg(\underbrace{\pi(a_1|s) + \cdots + \pi(a_{|\mathcal{A}|}|s)}_{|\mathcal{A}| \text{ terms sum to } 1} \\
&\quad - \underbrace{\left(\frac{\epsilon}{|\mathcal{A}(s)|} + \cdots + \frac{\epsilon}{|\mathcal{A}(s)|} \right)}_{|\mathcal{A}(s)| \text{ terms sum to } \epsilon} \Bigg) \\
&= \frac{1}{1-\epsilon} \left(1 - \epsilon \right) \\
&= 1
\end{aligned}$$

which is a weighted sum of $q_\pi(s,a)$, with w_a being the weights

Since $\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|} \geq 0$ and $1-\epsilon \geq 0$, we have $w_a \geq 0$

We note that $w_a \geq 0$ and $\sum_a w_a = 1$ imply that $w_a \leq 1$

Since $\sum_a w_a \, q_\pi(s,a)$ is a weighted sum of $q_\pi(s,a)$ with the weights w_a summing to 1, it cannot be greater than the largest $q_\pi(s,a)$, i.e., $\max_a q_\pi(s,a)$. Therefore

$$\max_a q_\pi(s,a) \geq \sum_a w_a \, q_\pi(s,a) = \sum_a \left[\left(\frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|}}{1-\epsilon} \right) q_\pi(s,a) \right]$$

Monte Carlo Control *without* Exploring Starts: How to Find $\pi' \geq \pi$

$$
\begin{aligned}
q_{\pi'}(s,a) &= \sum_a \pi'(a \mid s)\, q_\pi(s,a) \\
&= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\epsilon) \cdot \underbrace{\max_a q_\pi(s,a)}_{\text{see Proposition}} \\
&\geq \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\epsilon) \sum_a \left[\left(\frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|}}{1-\epsilon} \right) q_\pi(s,a) \right] \\
&= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + \sum_a \pi(a \mid s)\, q_\pi(s,a) - \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) \\
&= \sum_a \pi(a \mid s)\, q_\pi(s,a) \\
&= v_\pi(s)
\end{aligned}
$$

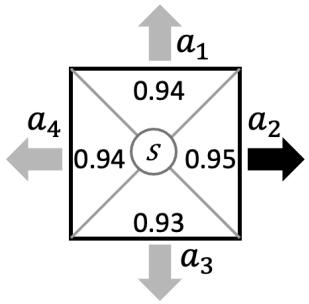
Hence, $q_{\pi'}(s,a) \geq v_\pi(s)$ in *Policy Improvement Theorem* is satisfied

Consequently, we have $v_{\pi'}(s) \geq v_\pi(s)$ and so $\pi' \geq \pi$

Through policy iteration, eventually we obtain $\pi' = \pi_*$

The optimal policy π_* thus obtained is probabilistic, i.e., it still explores with probability of

$$\frac{\epsilon}{|\mathcal{A}(s)|}\Big(|\mathcal{A}(s)| - 1\Big) = \epsilon\left(1 - \frac{1}{|\mathcal{A}(s)|}\right)$$

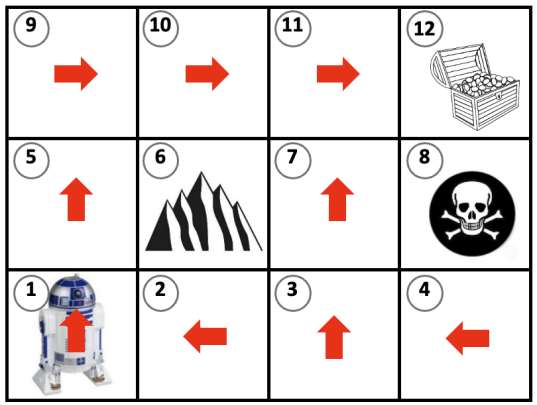


$\epsilon = 0.1$

$$\pi'_*(a_1|s) = 0.025$$
$$\pi'_*(a_2|s) = 0.925$$
$$\pi'_*(a_3|s) = 0.025$$
$$\pi'_*(a_4|s) = 0.025$$

Not desirable in practice

For example, at state 7 we would want the robot to follow greedy action as shown and not to explore



For practical application, we can obtain optimal deterministic policy:

$$\bar{\pi}_*(s) = \max_a q_{\pi'_*}(s, a) \qquad \text{e.g., } \bar{\pi}_*(7) = a_2$$

Paremeter ◦ A small $0 < \epsilon \leq 1$

Initialize ◦ An arbitrarily ϵ-soft policy π

◦ $Q(s, a) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

◦ Returns(s, a) as an empty list, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Repeat forever (for each episode)

Generate an episode with T steps following π

$S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$

$G \leftarrow \gamma G + R_{t+1}$

Unless pair (S_t, A_t) appears in $S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}$

Append G to Return(S_t, A_t)

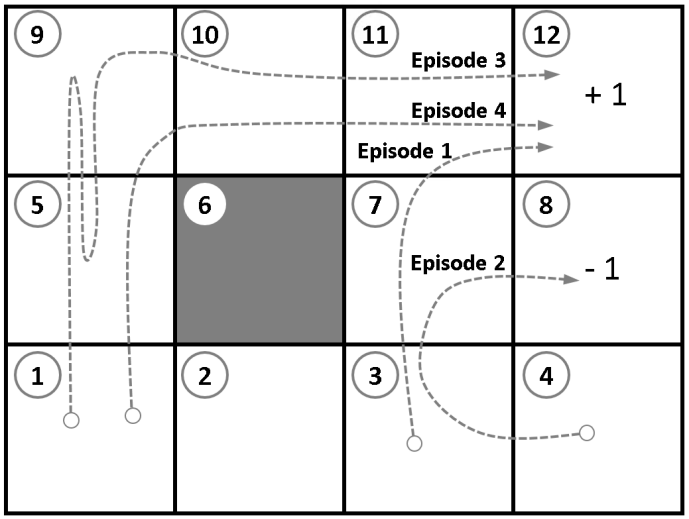
$Q(S_t, A_t) \leftarrow$ average(Return(S_t, A_t))

$A^* \leftarrow \underset{a}{\text{argmax}} Q(S_t, a)$

For all $a \in \mathcal{A}(S_t)$:

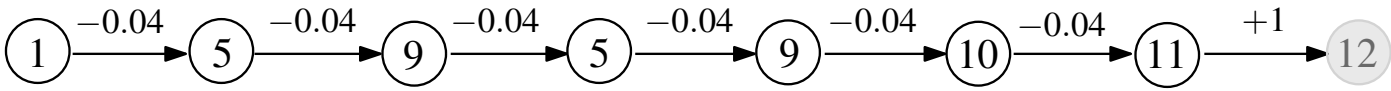
$$\pi(a \,|\, S_t) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(S_t)|} & \text{if } a = A^* \\ \frac{\epsilon}{|\mathcal{A}(S_t)|} & \text{if } a \neq A^* \end{cases}$$

Estimate of v_π (or q_π) updated only after *completion* of episode



When episode terminates, we "look back" and calculate return G_t for each state $S_t = s$ visited by the agent during episode, i.e.,

$$G_t = \sum_{k=0}^{T-t-1} \left(\gamma^k R_{t+k+1} \,\middle|\, S_t = s \right)$$

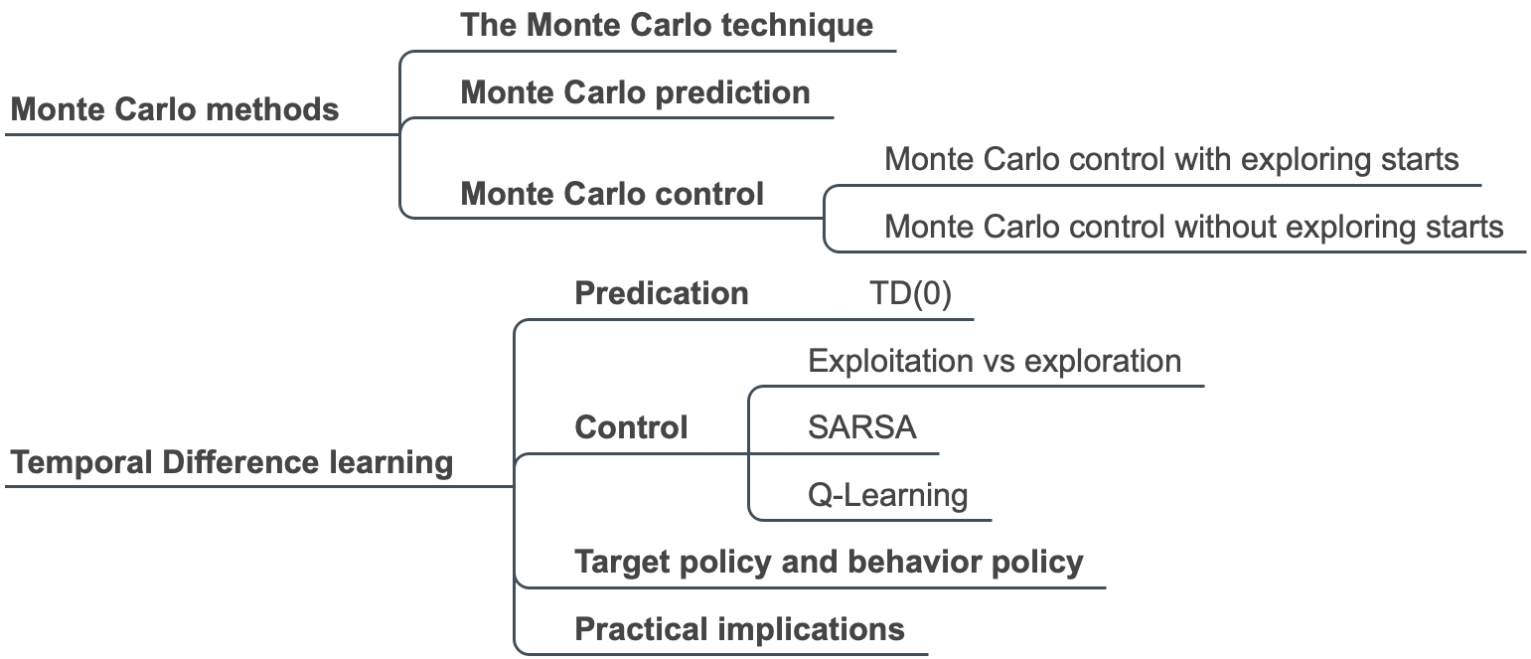


$$G_t\big|_{S_t=1} = 0.9^0 \cdot (-0.04) + 0.9^1 \cdot (-0.04) + 0.9^2 \cdot (-0.04) + 0.9^3 \cdot (-0.04) + 0.9^4 \cdot (-0.04) + 0.9^5 \cdot (-0.04) + 0.9^6 \cdot (+1)$$

If agent is expected to perform task while still learning how best to do it, online (i.e., during episode) updating is needed

$\Rightarrow$ Temporal Difference Methods

Monte Carlo Methods: Summary



Bootstrapping

TD methods update v or q values after one or a few time steps within an episode, before these values converge to a close approximate of their true values

Use iterative update rule of the form

$$\text{updated value} \leftarrow \text{current value} + \text{learning rate} \times \underbrace{(\text{estimated target value} - \text{current value})}_{\text{temporal difference error}}$$

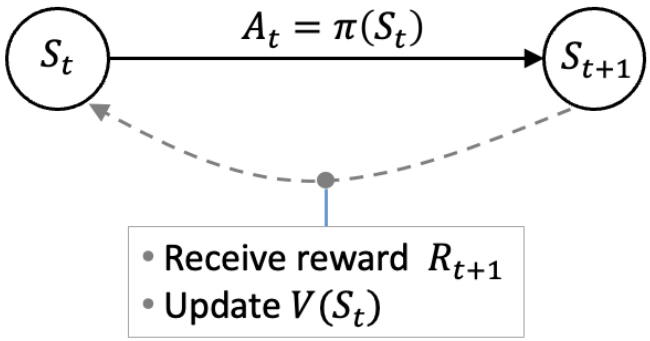
- TD(0) methods: Value is updated after each time step t
- Estimated target is "moving" (may be different for each time step)
- As t increases, estimated target approaches true v_π or $q_*(s, a)$

- TD(0) prediction: For evaluating a given policy π
- TD(0) control: For finding q_* (then π_*)
  - *SARSA*
  - Q-learning

$$\text{updated value} \leftarrow \text{current value} + \text{learning rate} \times \underbrace{(\text{estimated target value} - \text{current value})}_{\text{temporal difference error}}$$

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{estimated target value}} - V(S_t) \Big]$$

At each time step t, adjust $V(S_t)$ towards estimated target



Estimated target value is usually different at each time step

- As $t \to \infty$, $R_{t+1} + \gamma V(S_{t+1}) \to v_\pi$
- Temporal difference error $\to 0$
- $V(S_t) \to v_\pi$

Input π (the policy to be evaluated)

Parameter step size $\alpha \in (0, 1]$

$V(s) = 0$ set arbitrary initial values for non-terminal states

state values are zero for all terminal states by definition

Loop for each episode

  Initialize S

  Loop for each step of episode

    $A \leftarrow \pi(S)$

    Take action A, receive R and observe S'

    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma\, V(S') - V(S)\big]$

    $S \leftarrow S'$

until $S \in \hat{\mathcal{S}}$, with $\hat{\mathcal{S}}$ being the terminal state set

Output $V(s) \approx v_\pi(s)$

For finding q_* (then π_*): *SARSA* and Q-learning

At time step t, agent at state s (i.e., S_t)

1. Choose an action a (i.e., A_t) according to

   Behavior Policy

   - Determines how agent chooses an action
   - Affects convergence: Issue of exploitation vs exploration

2. Take action a, receive reward r and observe new state s'

3. Update $Q(s, a)$ value according to update rule:

$$\text{updated value} \leftarrow \text{current value} + \text{learning rate} \times \underbrace{(\text{estimated target value} - \text{current value})}_{\text{temporal difference error}}$$

SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[\underbrace{R_{t+1} + \gamma\, Q(S_{t+1}, A_{t+1})}_{\text{estimated target value}} - Q(S_t, A_t) \Big]$$

Q-Learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[\underbrace{R_{t+1} + \gamma \max_{a'} Q\left(S_{t+1}, a'\right)}_{\text{estimated target value}} - Q(S_t, A_t) \Big]$$

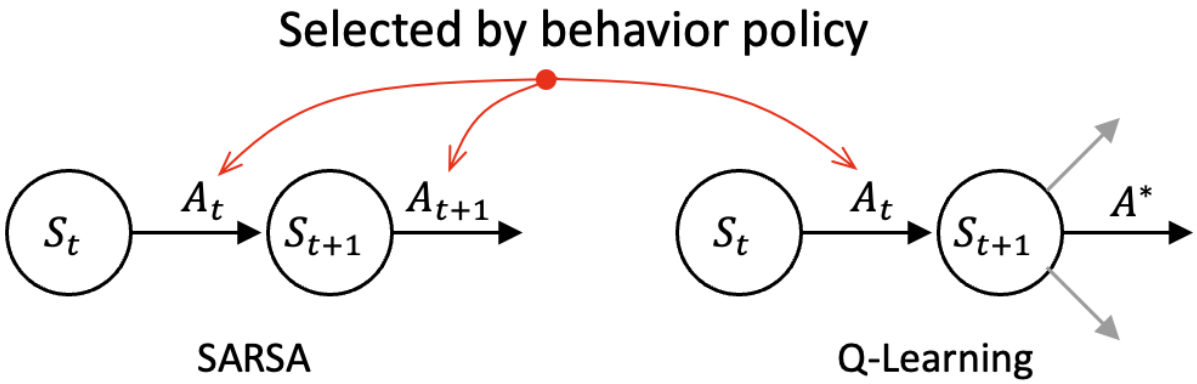
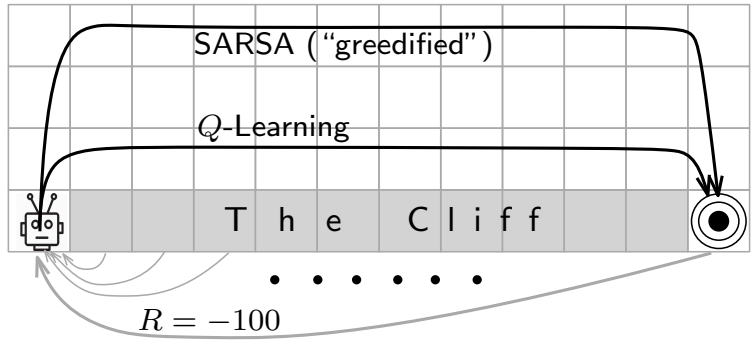


Illustration of Difference between *SARSA* and *Q*-learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma \, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma \max_{a'} Q\left(S_{t+1}, a'\right) - Q(S_t, A_t) \right]$$

$$A_t = \pi(a \,|\, s) \begin{cases} = & 1 - \epsilon + \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{if } a = A^* \\ & \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{otherwise} \end{cases}$$



$R = -1$ for all other actions

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \Big[R_{t+1} + \gamma \, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \Big[R_{t+1} + \gamma \max_{a'} Q\left(S_{t+1}, a'\right) - Q(S_t, A_t) \Big]$$

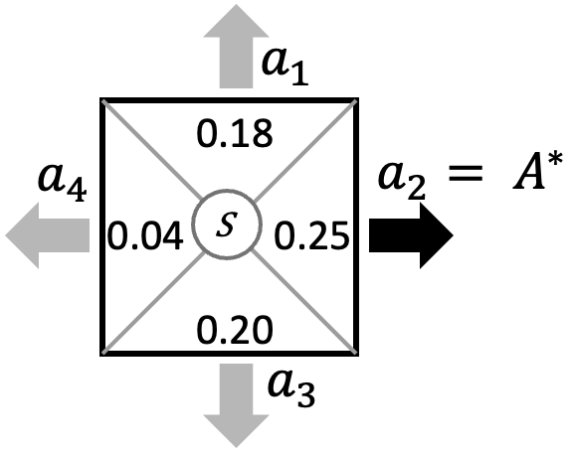
Estimated values Q converge to Q^* as $t \to \infty$ if

(1) $\sum_{k=0}^{\infty} \alpha_t^2 < \infty$ and $\sum_{k=0}^{\infty} \alpha_t \to \infty$

(2) All (s, a) pairs are (asymptotically) visited infinitely often

- (1) can be met by setting $\alpha_t = \frac{1}{t}$ with finite α_0 (e.g., 1)
- (2) can be satisfied if behavior policy ensures all (s, a) pairs have non-zero probability of being visited by agent (This is called persistent exploration)

Proof can be found in: Watkins, C. J. C. H. and Dayan, P., Q-learning, Machine Learning, 8:279-292, 1992



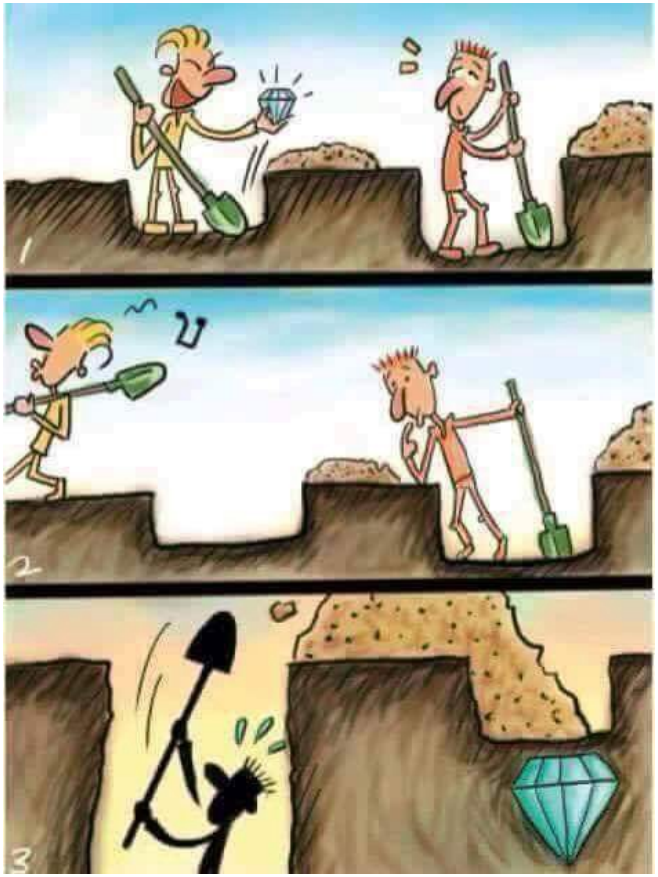
Exploitation: Use greedy policy to select currently known best action

$$\begin{aligned} a_{k+1} &= \max_{a'} Q_k\left(s_{k+1}, a'\right) \\ &\triangleq A^* = a_2 \end{aligned}$$

Exploration: Try action other than currently known best action

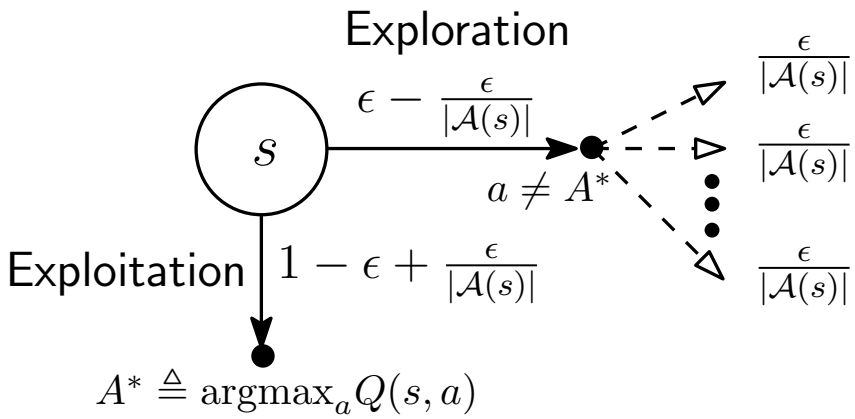
$$\begin{aligned} a_{k+1} &\neq \max_{a'} Q_k\left(s_{k+1}, a'\right) \\ &= a_1,\ a_3,\ \text{or}\ a_4 \end{aligned}$$

- Exploration is essential for learning
- If we do not try new things, then we will never develop new capabilities and skills
- However, if we always explore and do not utilize what we have learned, then we will most likely not be able to achieve anything significant



Behavior policy must balance exploration and exploitation

https://medium.com/deep-math-machine-learning-ai/ch-12-1-model-free-reinforcement-learning-algorithms-monte-carlo-sarsa-q-learning-65267cb8d1b4



$$\pi(a \,|\, s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{if } a = A^* \triangleq \text{argmax}_a \, Q(s, a) \\ \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{if } a \neq A^* \end{cases}$$

- ϵ is kept small so that agent will focus on task at hand most of the time, and will explore only occasionally
- Reduce exploration as learning continues by setting $\epsilon_k = \frac{1}{k}$

Parameters Step size $\alpha \in (0, 1]$, and small $0 < \epsilon < 1$

$Q(s, a) = 0$ Set arbitrary initial Q values for non-terminal states

Q values are zero for all terminal states by definition

Loop for each episode

Initialize S

Choose A from S using policy derived from Q (e.g., ϵ-greedy)

Loop for each step of episode

Take action A; receive R and observe S'

Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma\, Q(S',\, A') - Q(S, A)\big]$

$S \leftarrow S'$; $A \leftarrow A'$

until $S \in \hat{\mathcal{S}}$, with $\hat{\mathcal{S}}$ being the terminal state set

Parameters Step size $\alpha \in (0, 1]$, and small $0 < \epsilon < 1$

$Q(s, a) = 0$ Set arbitrary initial Q values for non-terminal states

Q values are zero for all terminal states by definition

Loop for each episode

Initialize S

Loop for each step of episode

Choose A from S using policy derived from Q (e.g., ϵ-greedy)

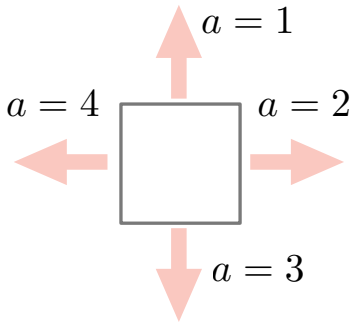
Take action A; receive R and observe S'

$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_{a'} Q(S', a') - Q(S, A)\big]$

$S \leftarrow S'$

until $S \in \hat{\mathcal{S}}$, with $\hat{\mathcal{S}}$ being the terminal state set

0	1	2
3	4	5
6	7	8



$$\rho(s, a, s') = \begin{cases} 1 & \text{if } s \neq 8 \text{ and } s' = 8 \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma = 0.9\,, \quad a_k = 1/k\,, \quad a_0 = 1$$

- Learning consists of series of episodes
- Starting at initial state robot makes transitions according to ϵ-greedy exploration and stops when reaching $s = 8$
- Will not actually simulate probabilistic selection of action in this example
- Will simply assume sequence of actions for an episode (as if they were observed to have occurred) so as to carry out calculation of values for Q-function

- For record keeping, define function $N(s, a)$ which counts accumulative number of times (over a set of episodes) a has been taken at s
- Two types of diagrams, showing
  - Values of $N(s, a)$ at end of an episode
  - Values of Q-function at end of an episode
- Initial values of $N(s, a)$ and Q-function:

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

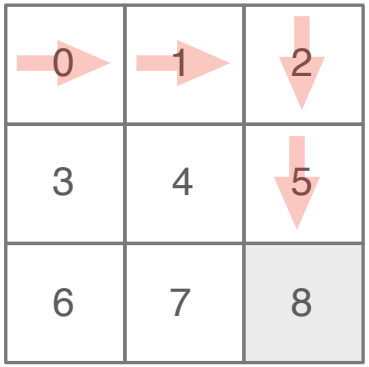
$N(s, a)$

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

Values of Q-function

Episode 1:

- Assume action sequence as shown in diagram
- $\alpha_k = 1/k,\ \alpha_0 = 1$
- $\gamma = 0.9$



Sample calculations:

$$
\begin{aligned}
Q_1(0,2) &= Q_0(0,2) + \alpha_0\Big(\rho(0,2,1) + \gamma \max_{a'} Q_0\left(1,a'\right) - Q_0(0,2)\Big) \\
&= 0 + 1\cdot(0 + 0.9\cdot 0 - 0) \\
&= 0 \\
&\cdots \\
Q_4(5,3) &= Q_3(5,3) + \alpha_3\Big(\rho(5,3,8) + \gamma \max_{a'} Q_3\left(8,a'\right) - Q_3(5,3)\Big) \\
&= 0 + \frac{1}{3}\Big(1 + 0.9\cdot 0 - 0\Big) \\
&= 0.333
\end{aligned}
$$

Detailed calculations are shown in supplementary notes

Values of $N(s,a)$ and Q-function at end of Episode 1

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

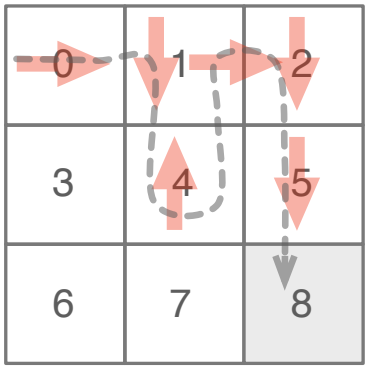
(a) $N(s,a)$

0 0 0	0 1 0	0 2 0
0 3 0	0 4 0	0 5 0 **0.333**
0 6 0	0 7 0	0 8 0

(b) Q-function

Episode 2:

- Assume action sequence as shown in diagram
- $\alpha_0 = 1$
- $\gamma = 0.9$



Sample calculations:

$$
\begin{aligned}
Q_1(0,2) &= Q_0(0,2) + \alpha_0 \left(\rho(0,2,1) + \gamma \max_{a'} Q_0\left(1, a'\right) - Q_0(0,2) \right) \\
&= 0 + 1 \cdot (0 + 0.9 \cdot 0 - 0) \\
&= 0 \\
&\dots \\
&\dots
\end{aligned}
$$

Detailed calculations are shown in supplementary notes

Values of $N(s,a)$ and Q-function at end of Episode 2

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

(a) $N(s,a)$

0 0 0	0 1 0	0 2 **0.075**
0 3 0	0 4 0	0 5 **0.466**
0 6 0	0 7 0	0 8 0

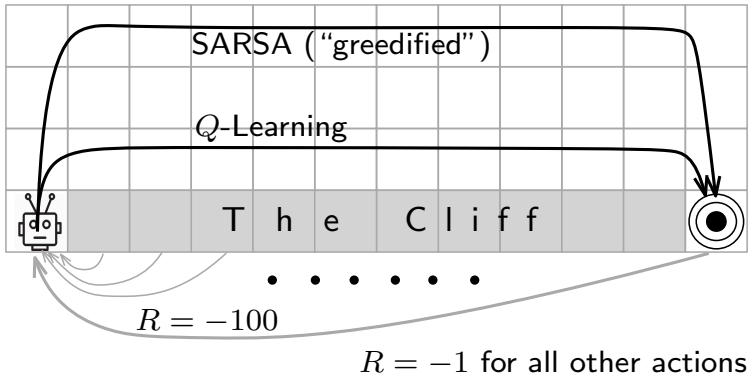
(b) Q-function

Iterative process repeats until values of Q-function converge to optimal values, upon which an optimal policy can be obtained

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma \, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma \max_{a'} Q\left(S_{t+1}, a'\right) - Q(S_t, A_t) \right]$$

$\pi(a \,|\, s) = 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$ if $a = A^*$ and $\pi(a \,|\, s) = \frac{\epsilon}{|\mathcal{A}(s)|}$ otherwise

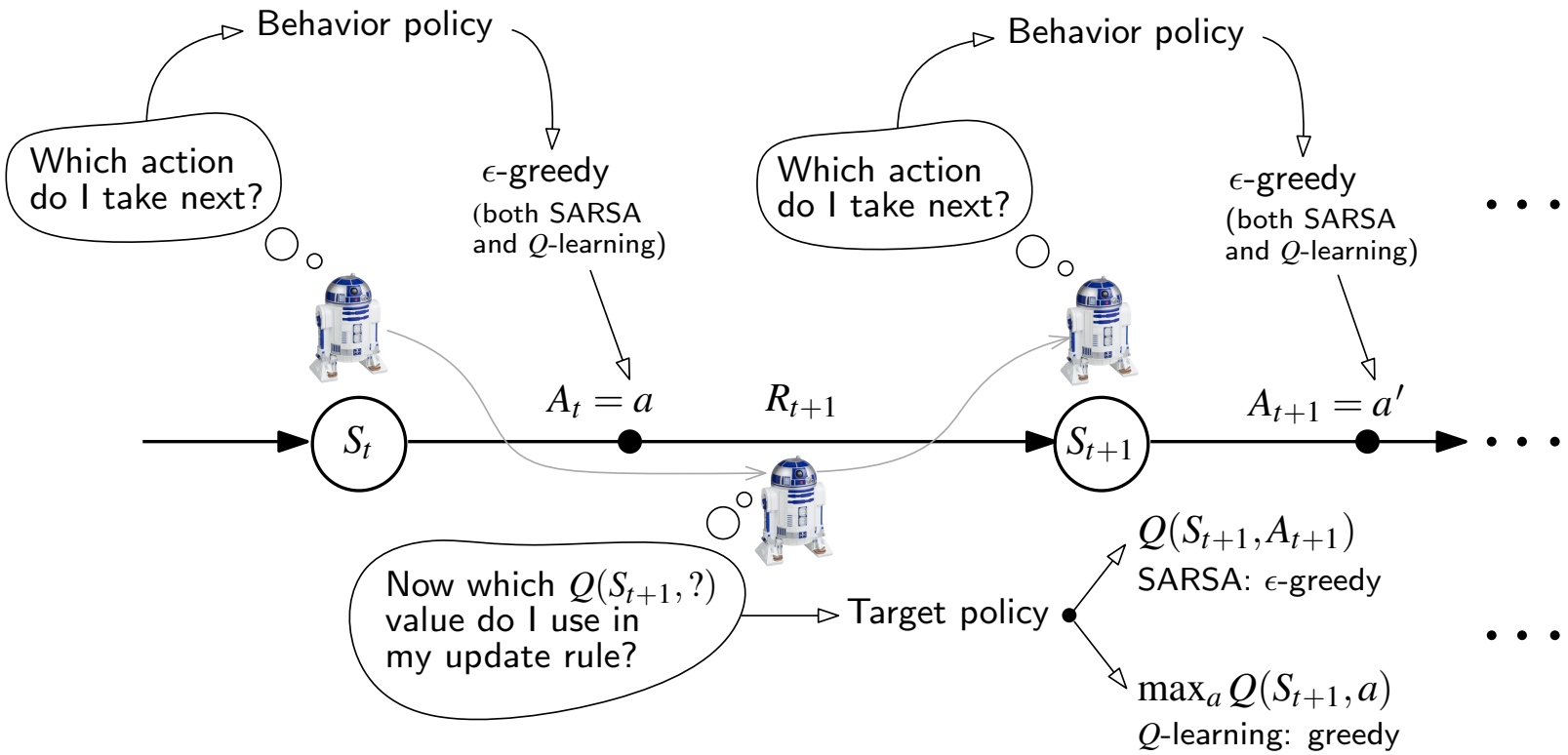


- π_* obtained from Q-learning incurs least cost but highly risky
- SARSA yields optimal ϵ-greedy policy ($\neq \pi_*$ in general)
- Optimal ϵ-greedy policy still explores with probability ϵ, which carries risk of robot falling off cliff. The value $Q(S_{t+1}, A_{t+1})$ would be significantly negative if S_{t+1} is next to cliff and if A_{t+1} is the action that results in robot falling off cliff. SARSA recognizes this risk and keeps robot further away from cliff
- "Greedifying" optimal ϵ-greedy policy yields near-optimal policy

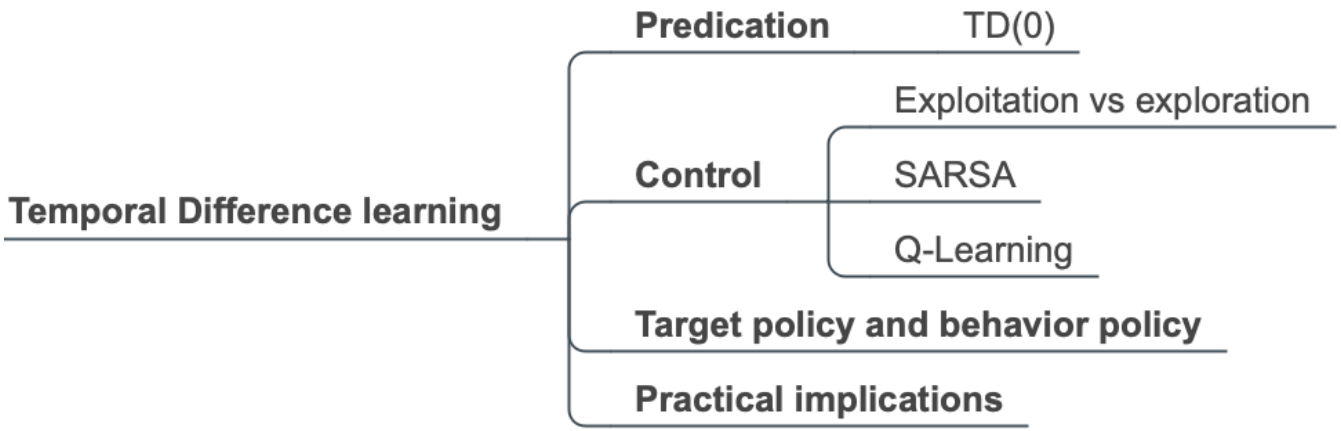
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma\, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma \max_{a'} Q\left(S_{t+1}, a'\right) - Q(S_t, A_t) \right]$$

$\pi(a \mid s) = 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$ if $a = A^*$ and $\pi(a \mid s) = \frac{\epsilon}{|\mathcal{A}(s)|}$ otherwise



Online: Behavior = Target. Offline: Behavior ≠ Target



Implementation Problem

- During learning, need to keep table of $V(s)$ or $Q(s,a)$ values
- To update values, need to search table for current $V(s)$ or $Q(s,a)$
- Table can be huge, e.g., $\sim 2 \cdot 10^{170}$ states $\times$ 250 actions for GO
- Searching through huge table is intractable

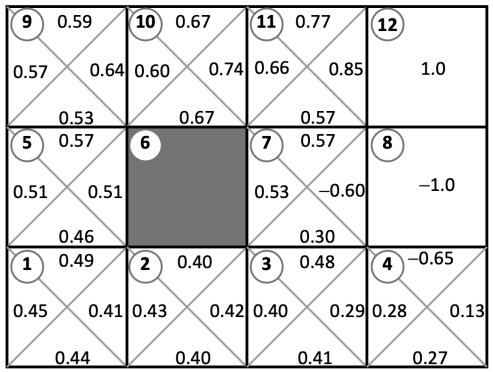
$\Rightarrow$ Deep Reinforcement Learning

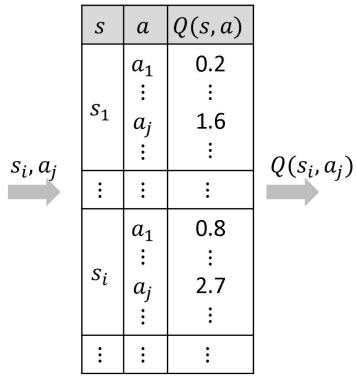
Problem

- During learning, need to keep table of $V(s)$ or $Q(s,a)$ values
- To update values, need to search table for current $V(s)$ or $Q(s,a)$
- Table can be huge, e.g., $\sim 2 \cdot 10^{170}$ states $\times$ 250 actions for GO
- Searching through huge table is intractable

Solution

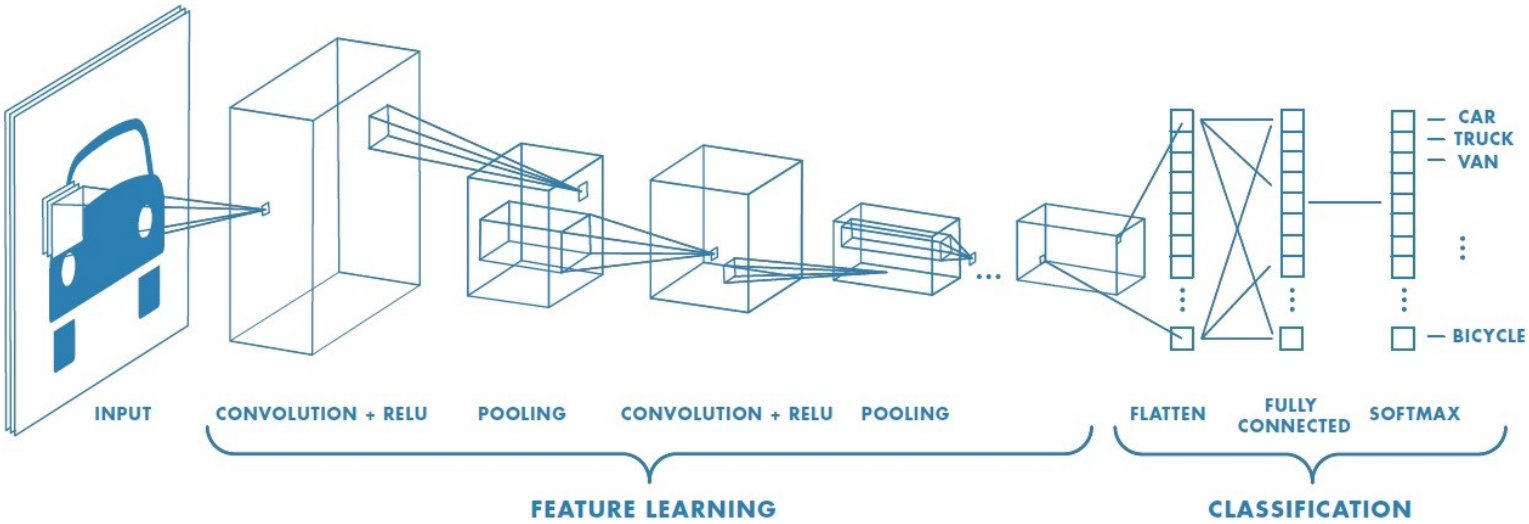
- Use neural network to approximate 'true' but huge table
- This makes Q-learning 'deep' (Deep Q-Network, DQN)

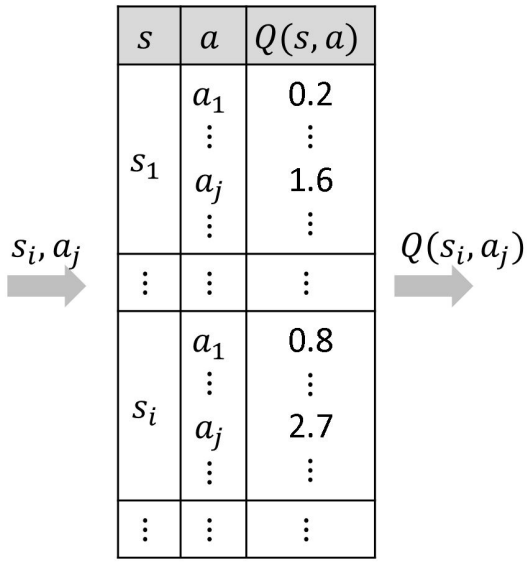


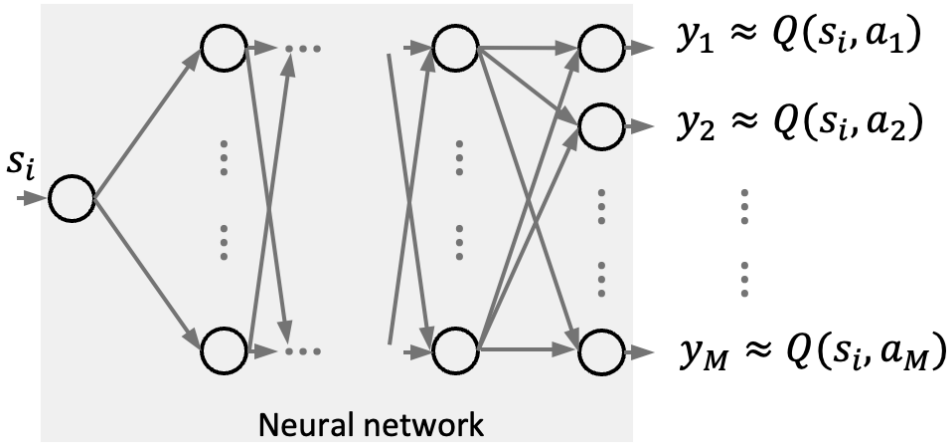


Allows computational models composed of multiple processing layers to learn representations of data with multiple levels of abstraction

- Multiple layers of processing units in layers
- Very low-level (e.g., pixel) data to descriptive labels







- Use neural network to approximate 'true' but huge table

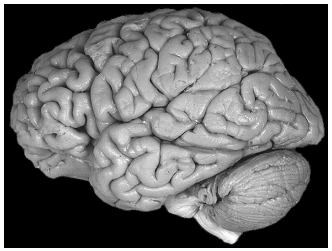
Neural-network in **AlpahGO** is much smaller than Q-table

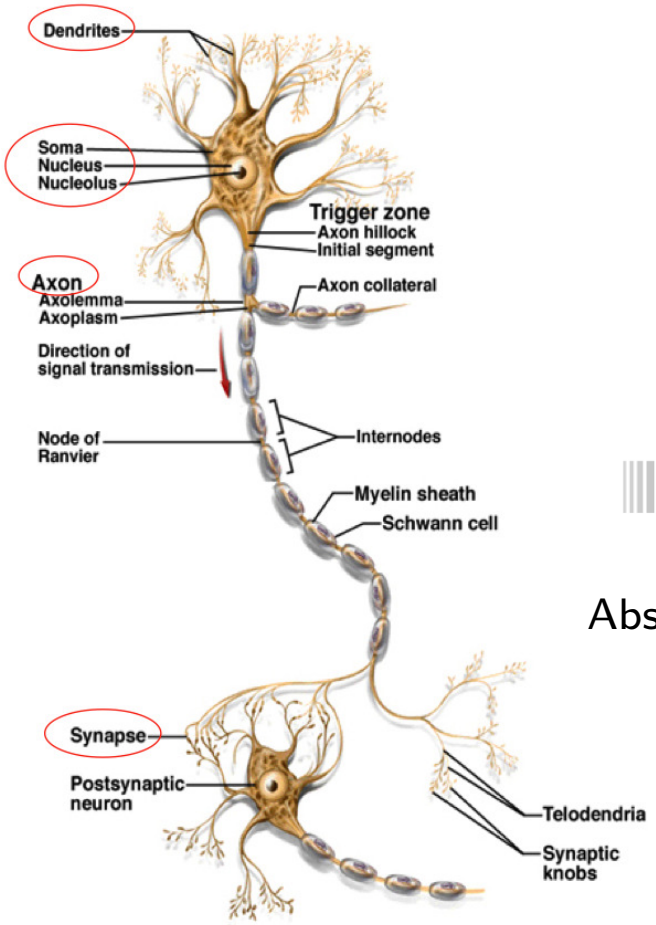
Board positions $\mathcal{S}$	Moves $\mathcal{A}$	Q-table entries
2×10^{170}	250	500×10^{170}

Atoms in known observable universe $\sim 10^{80}$

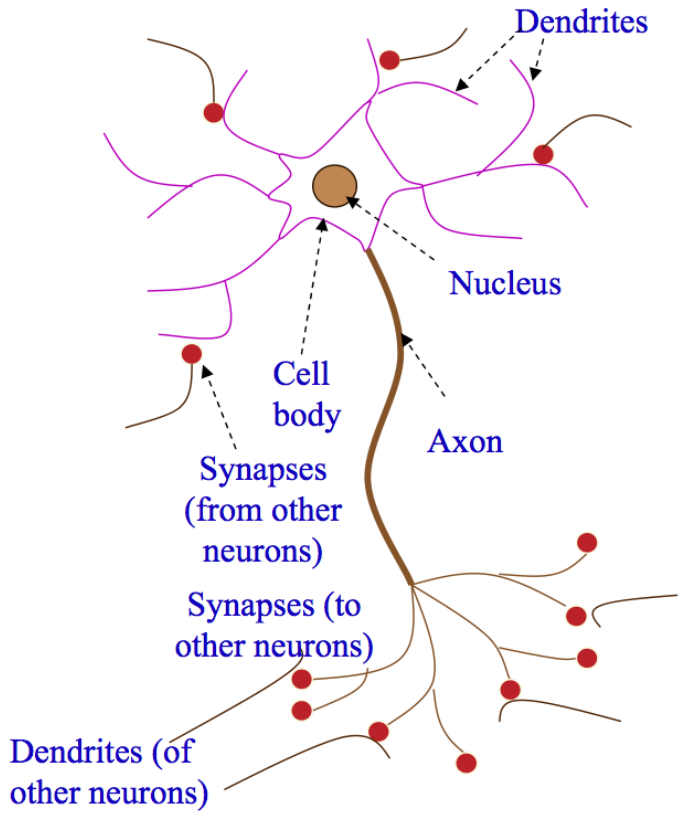
Input layer	$19 \times 19 \times 49$
Hidden layer	$19 \times 19 \times 192 \times \mathbf{12}$ $19 \times 19 \times \mathbf{1}$ $256 \times \mathbf{1}$
Output layer	1

Conceptual Model: Abstraction

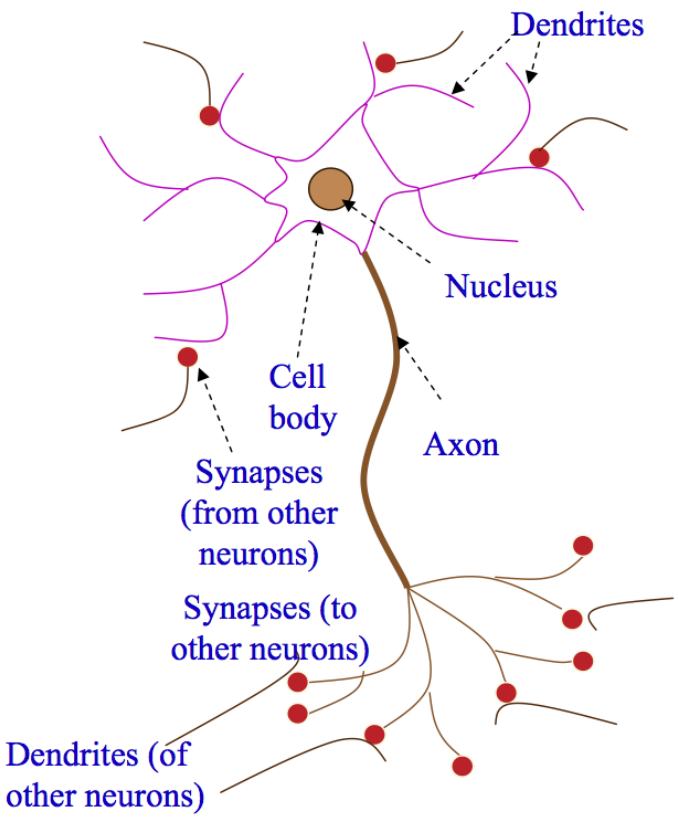




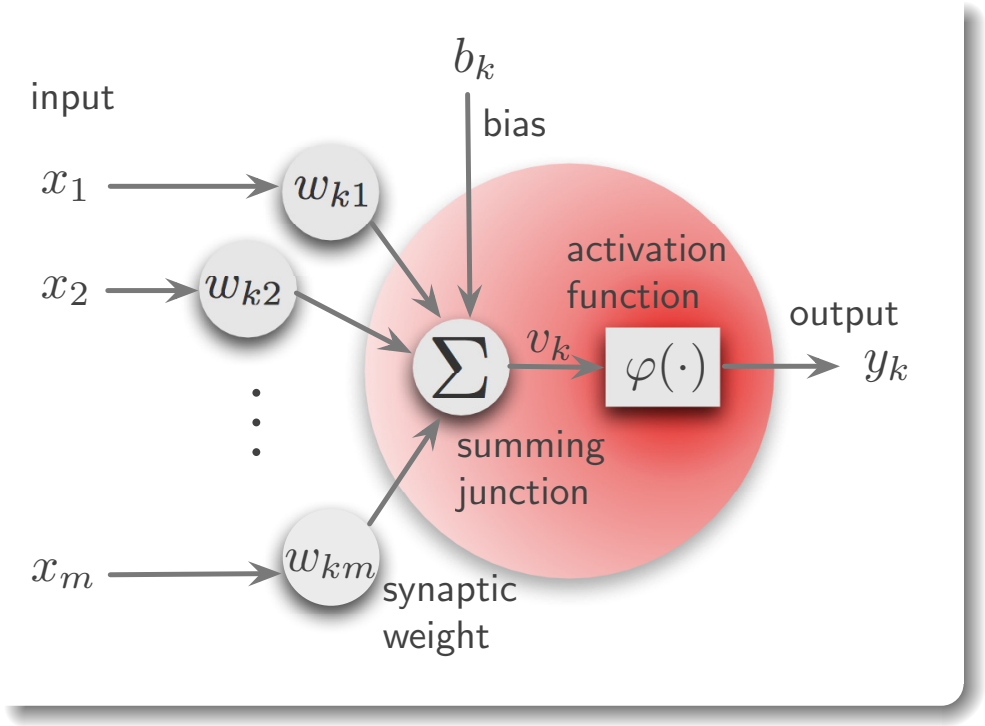
Abstraction



Conceptual model

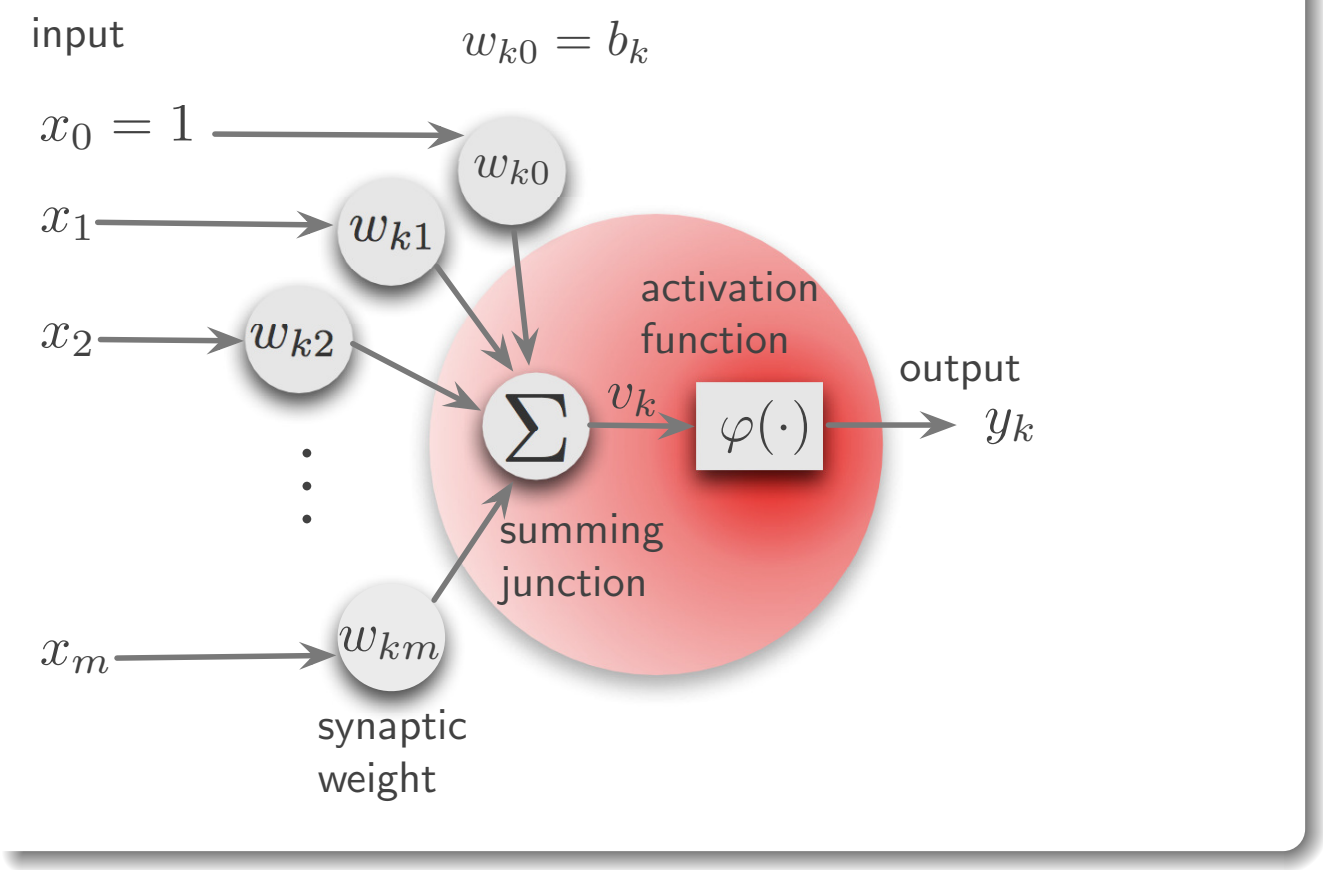


Computational model



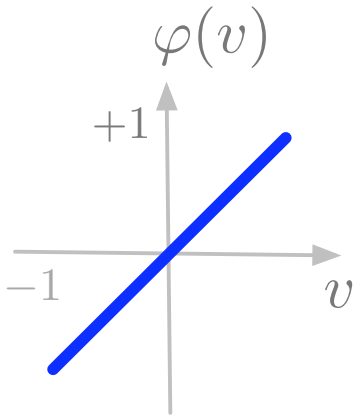
- x: Input (from dendrites)
- w: strength of synaptic connection between neurons (a.k.a. weight)

Model of a neuron with bias as weight

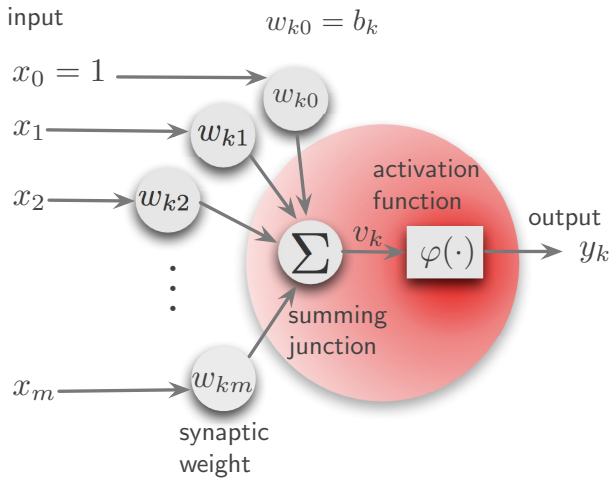


$$v_k = \sum_{j=0}^{m} w_{kj}\, x_j \qquad y_k = \varphi(v_k)$$

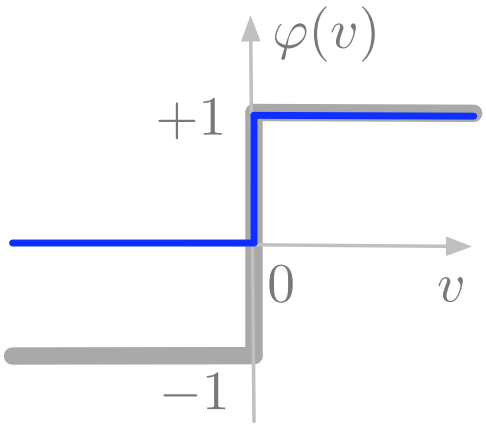
- Linear function



$$\varphi(v) = v$$



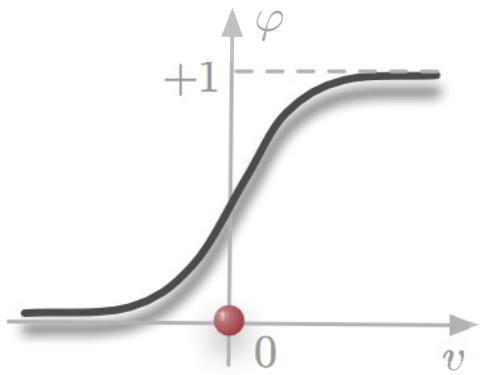
- Threshold function (hardlimiter)



$$\varphi(v) = \text{sgn}[v] = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

or

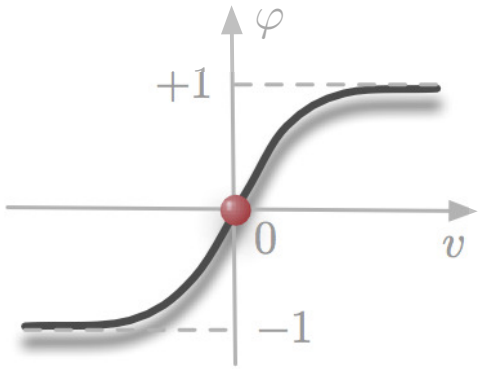
$$\varphi(v) = \text{sgn}[v] = \begin{cases} 1 & \text{if } v > 0 \\ v & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases}$$



Range of activation: $[0, 1]$

$$\varphi(v) = \frac{1}{1 + e^{-av}}$$

$$\frac{d\varphi(v)}{dv} = a\left(1 - \varphi(v)\right)\varphi(v)$$



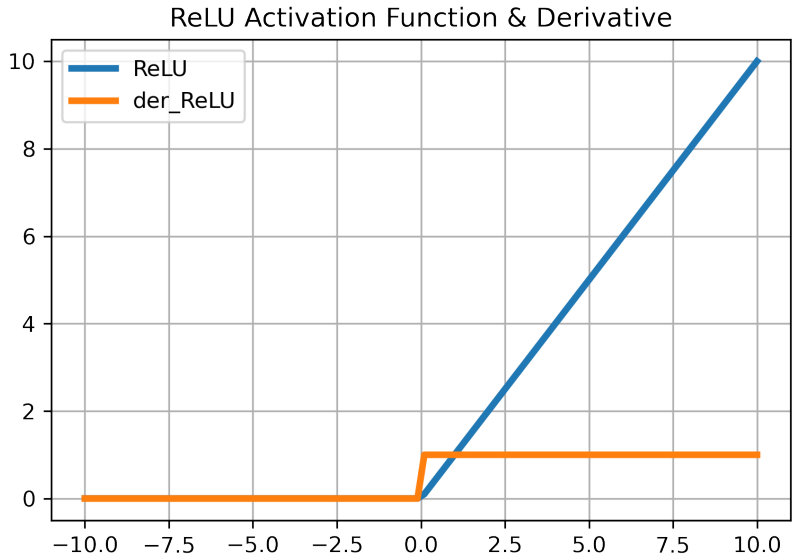
Range of activation: $[-1, 1]$

$$\varphi(v) = \tanh(av)$$

$$\frac{d\varphi(v)}{dv} = a\left(1 - \varphi^2(v)\right)$$

$a > 0$

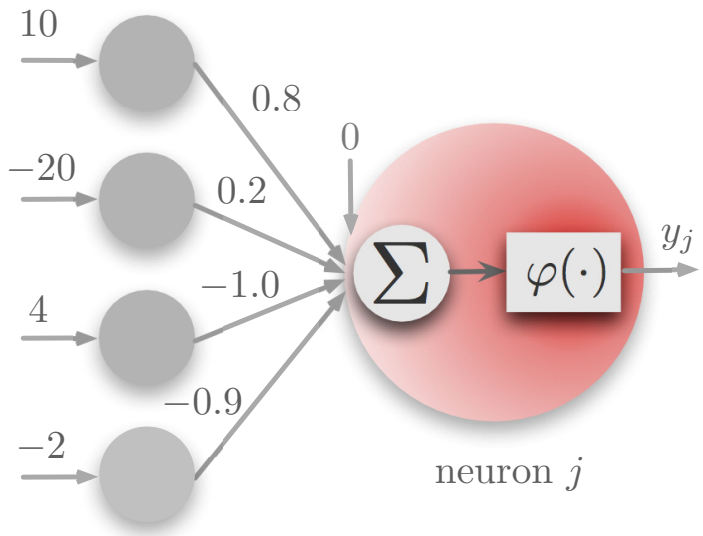
Types of Activation Function: Rectified Linear Unit (ReLU)



$$\varphi(v) = \max(0, v)$$

$$\frac{d\varphi(v)}{dv} = \begin{cases} 0 & \text{if } v < 0 \\ 1 & \text{if } v > 0 \end{cases}$$

- Sparse activation: In randomly initialized network, only about 50% of hidden units have non-zero output
- Better gradient propagation: Fewer vanishing gradient problems compared to sigmoidal activation functions
- Efficient computation: Only comparison, addition and multiplication
- Scale-invariant: $\max(0, ax) = a \max(0, x)$ for $a \geq 0$

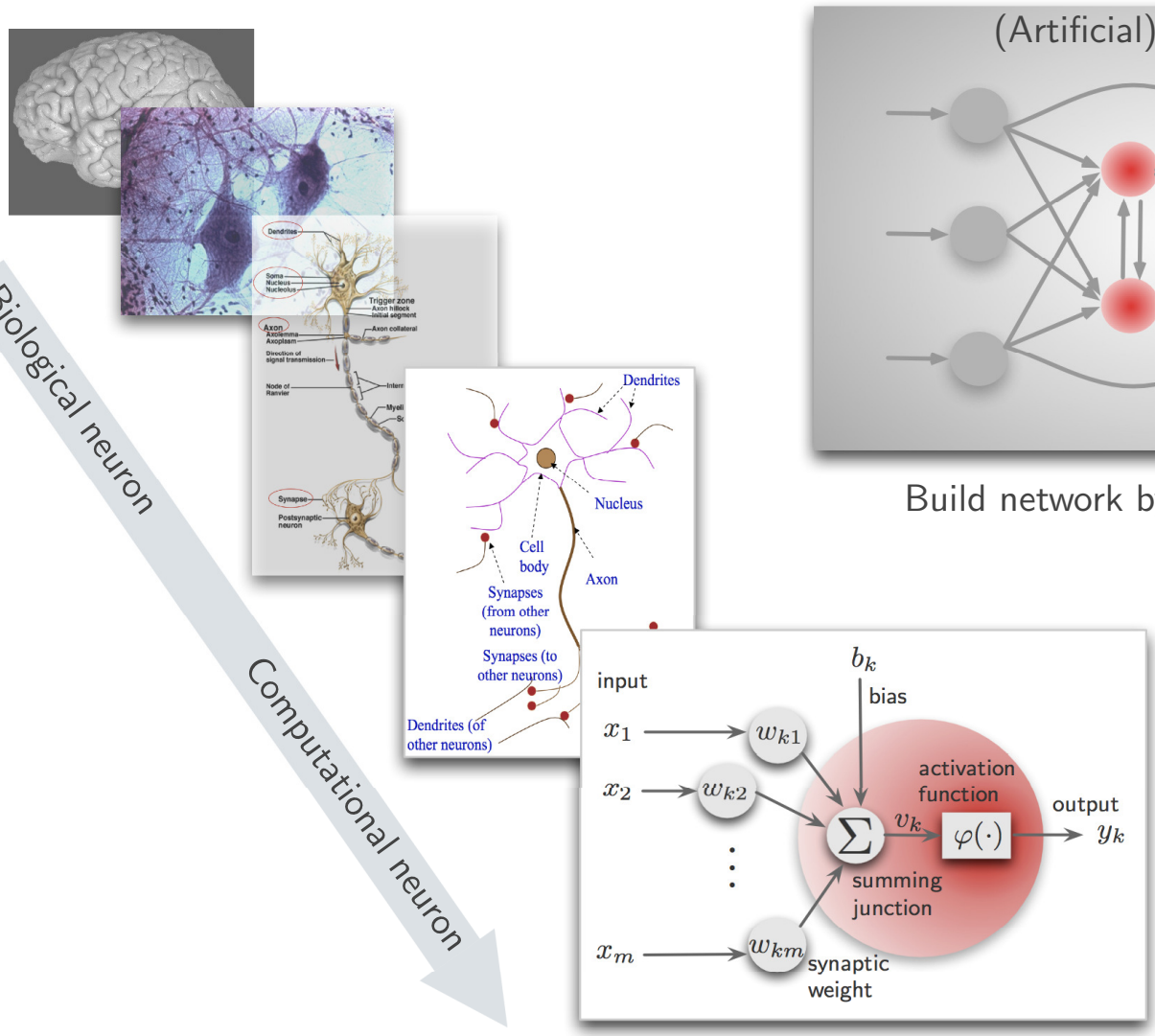


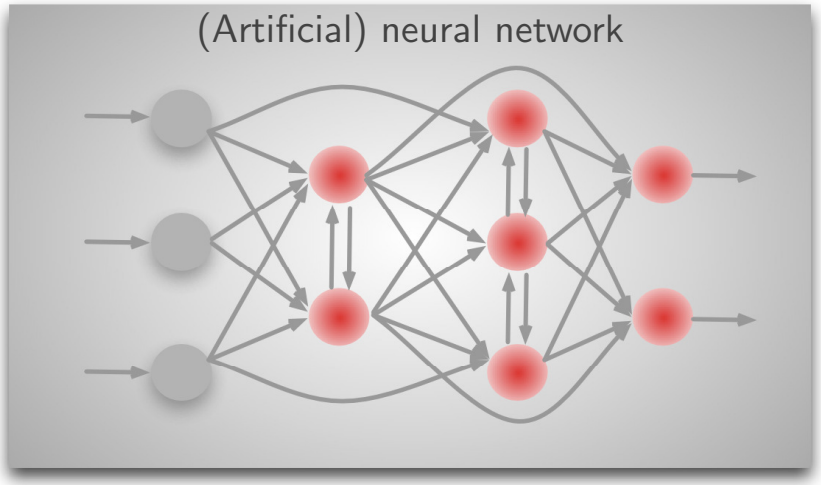
$$x_0 = 1 \qquad w_{j0} = b_j = 0$$
$$x_1 = 10 \qquad w_{j1} = 0.8$$
$$x_2 = -20 \qquad w_{j2} = 0.2$$
$$x_3 = 4 \qquad w_{j3} = -1.0$$
$$x_4 = -2 \qquad w_{j4} = -0.9$$

$$\begin{aligned} v_j &= \sum_{i=0}^{4} w_{ji} x_i \\ &= 0(1) + 0.8(10) + 0.2(-20) - 1.0(4) - 0.9(-2) \\ &= 1.8 \end{aligned}$$

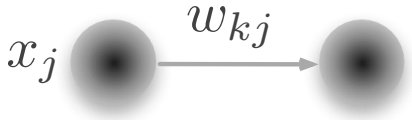
Activation function	Output
Linear	$y_j = v_j = 1.8$
Hardlimiter	$y_j = \varphi(v_j) = 1$

From Biological Neuron to Computational Model

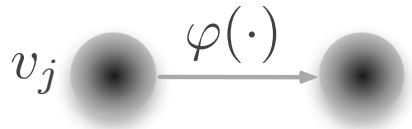




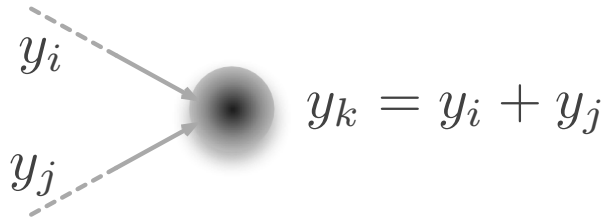
Build network by connecting neurons



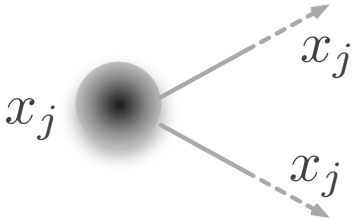
Synaptic link



Activation link

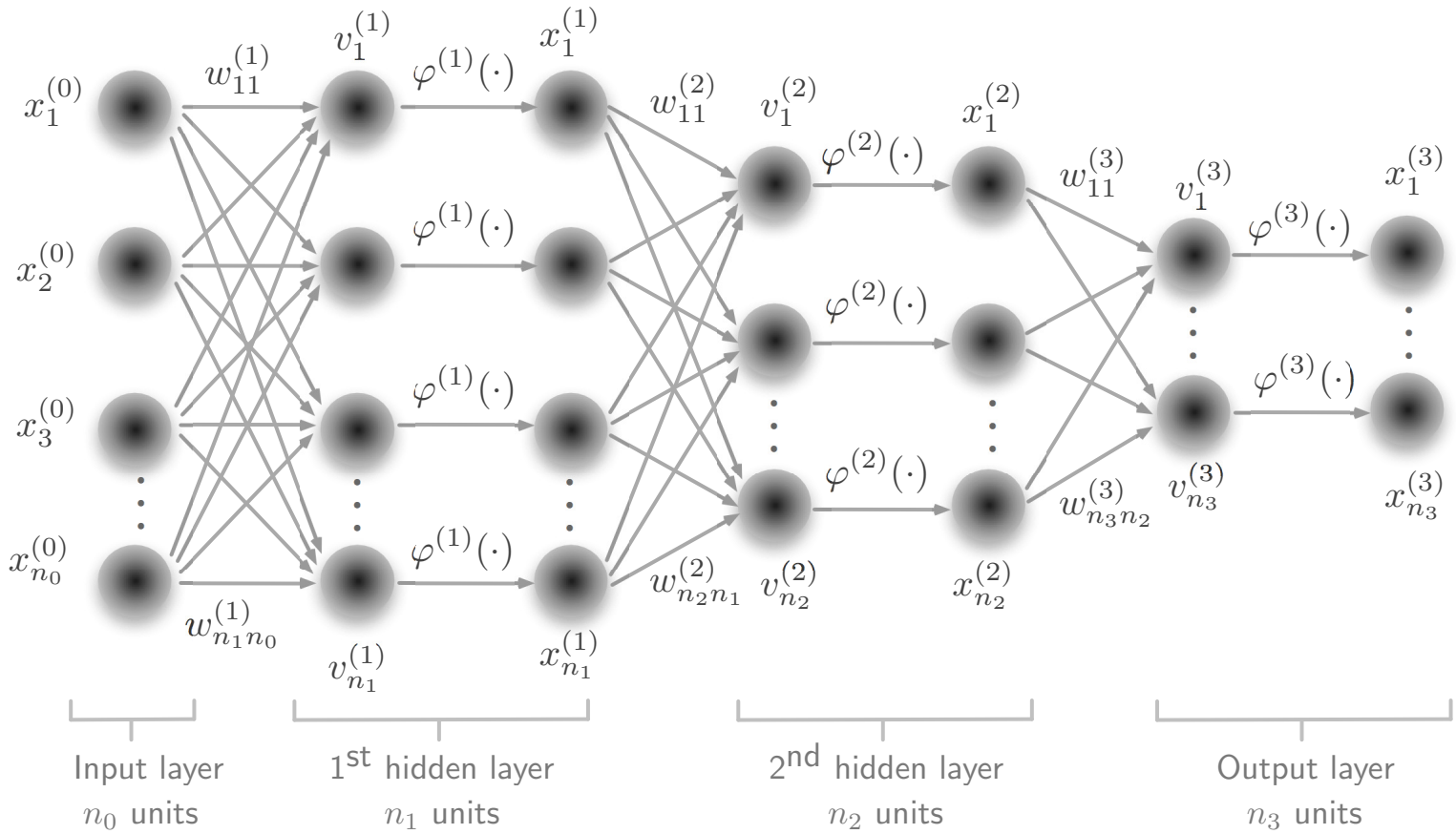


Synaptic convergence



Synaptic divergence

Structure: A Three-Layer Network

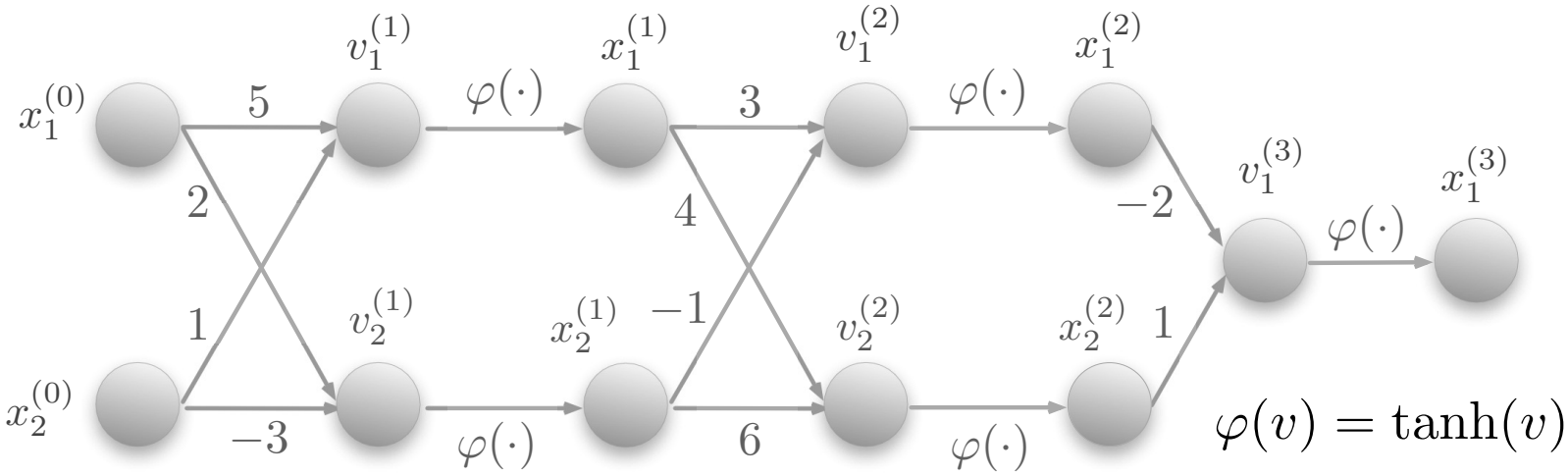


If unit j in layer l has bias b, then add $x_0^{(l-1)} = 1$ and $w_{j0}^{(l)} = b$

$$v_j^{(l)} = \sum_{i=1}^{n_{l-1}} w_{ji}^{(l)} x_i^{(l-1)} \qquad x_j^{(l)} = \varphi^{(l)}\left(v_j^{(l)}\right)$$

Superscript (l) in $\varphi^{(l)}(\cdot)$ usually omitted since same φ used for all units

Back



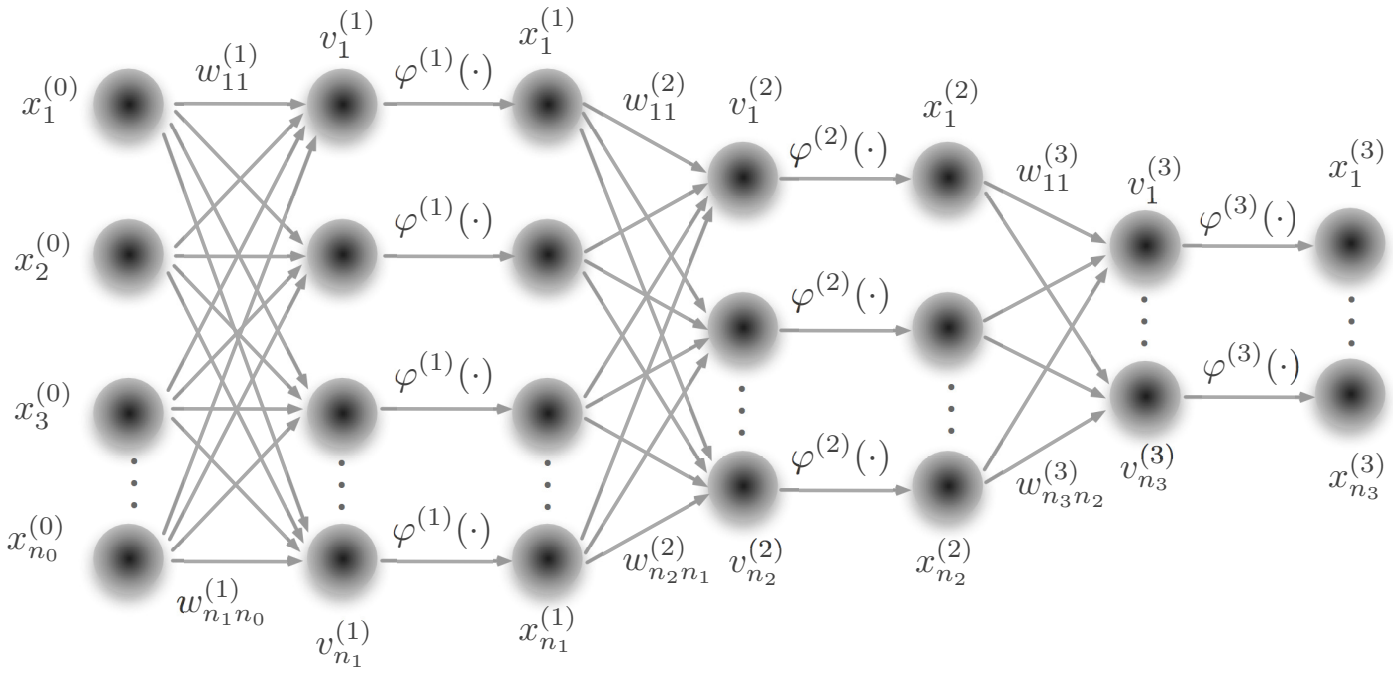
$$v_1^{(1)} = 5x_1^{(0)} + x_2^{(0)} \qquad x_1^{(1)} = \varphi\left(v_1^{(1)}\right)$$

$$v_2^{(1)} = 2x_1^{(0)} - 3x_2^{(0)} \qquad x_2^{(1)} = \varphi\left(v_2^{(1)}\right)$$

$$v_1^{(2)} = 3x_1^{(1)} - x_2^{(1)} \qquad x_1^{(2)} = \varphi\left(v_1^{(2)}\right)$$

$$v_2^{(2)} = 4x_1^{(1)} + 6x_2^{(1)} \qquad x_2^{(2)} = \varphi\left(v_2^{(2)}\right)$$

$$v_1^{(3)} = -2x_1^{(2)} + x_2^{(2)} \qquad x_1^{(3)} = \varphi\left(v_1^{(3)}\right)$$



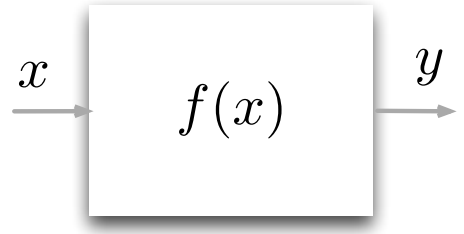
In general, network output can be expressed as function of input $\mathbf{x}^{(0)}$ and weight $\mathbf{W}$

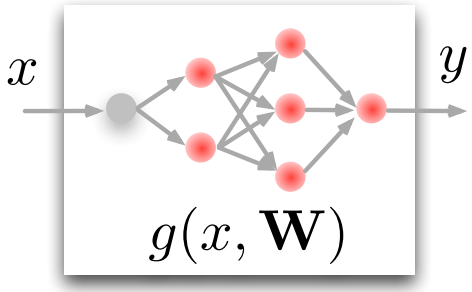
$$\mathbf{x}^{(3)} = g\left(\mathbf{x}^{(0)}, \mathbf{W}\right)$$

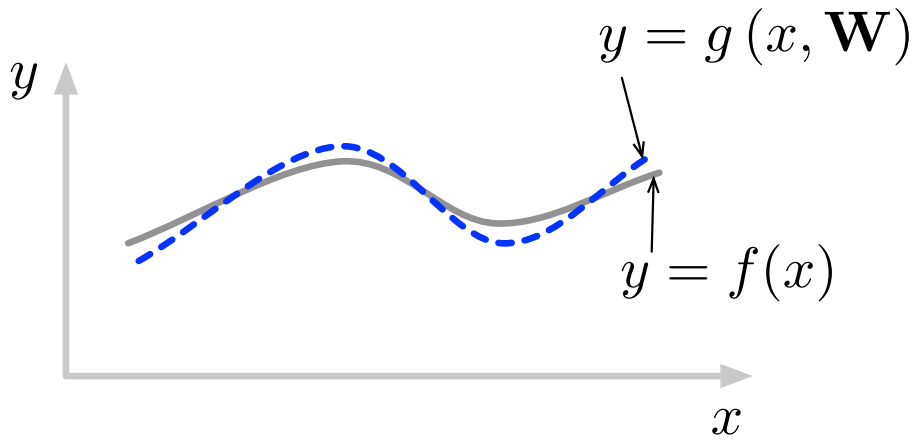
Function approximation

Given a function $\mathbf{y} = f(\mathbf{x})$, find "optimal" $\mathbf{W}^*$ such that

$$\begin{aligned} \mathbf{x}^{(3)} &= g\left(\mathbf{x}^{(0)}, \mathbf{W}^*\right) \\ &\approx f\left(\mathbf{x}^{(0)}\right) = \mathbf{y} \end{aligned}$$



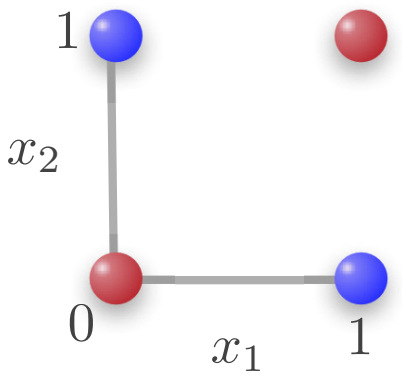




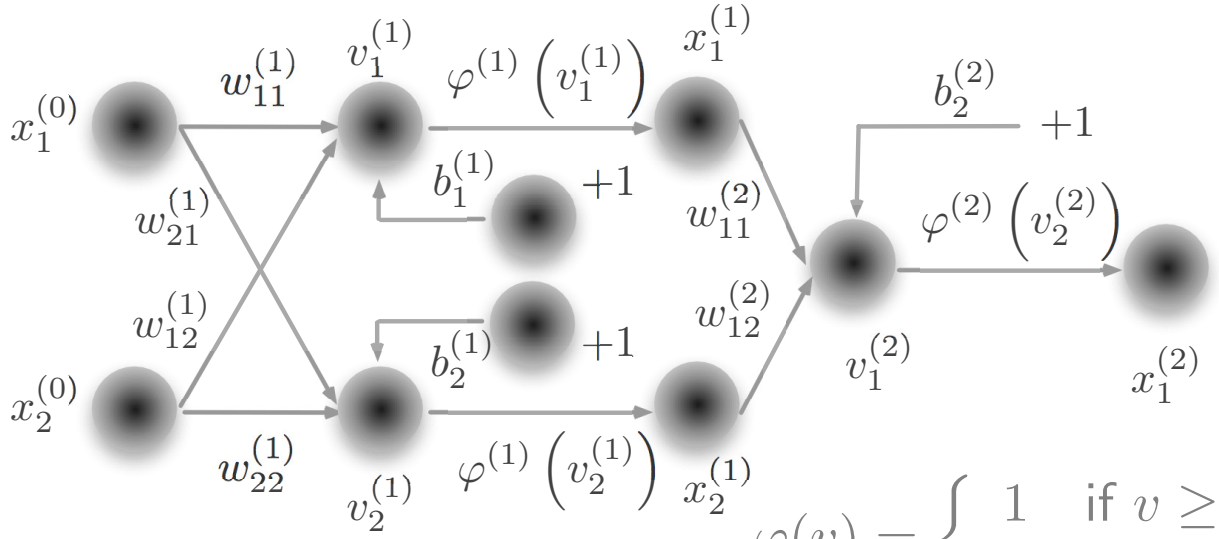
Cybenko's theorem

A MLP with only one layer of hidden units is capable of approximating any function with finitely many discontinuities to arbitrary precision, provided there are enough number of hidden units whose activation functions are non-linear

XOR problem can be solved by a two-layer feedforward network with hardlimiter activation function

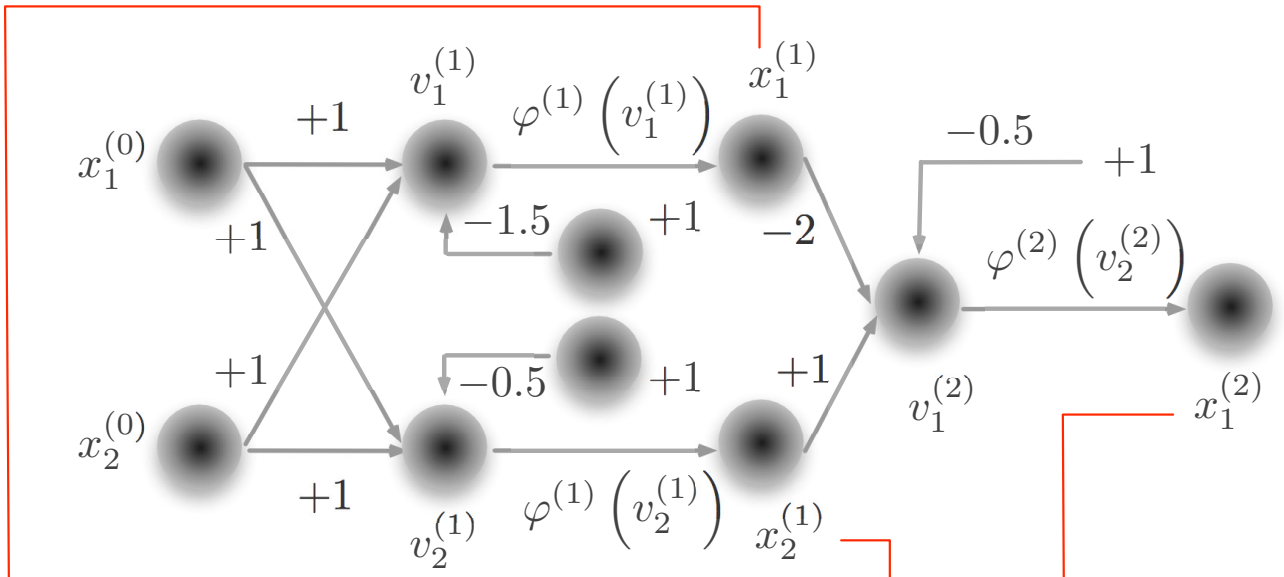


k	$\mathbf{x}(k)$	$d(k)$
1	$[0,0]^T$	0
2	$[0,1]^T$	1
3	$[1,0]^T$	1
4	$[1,1]^T$	0

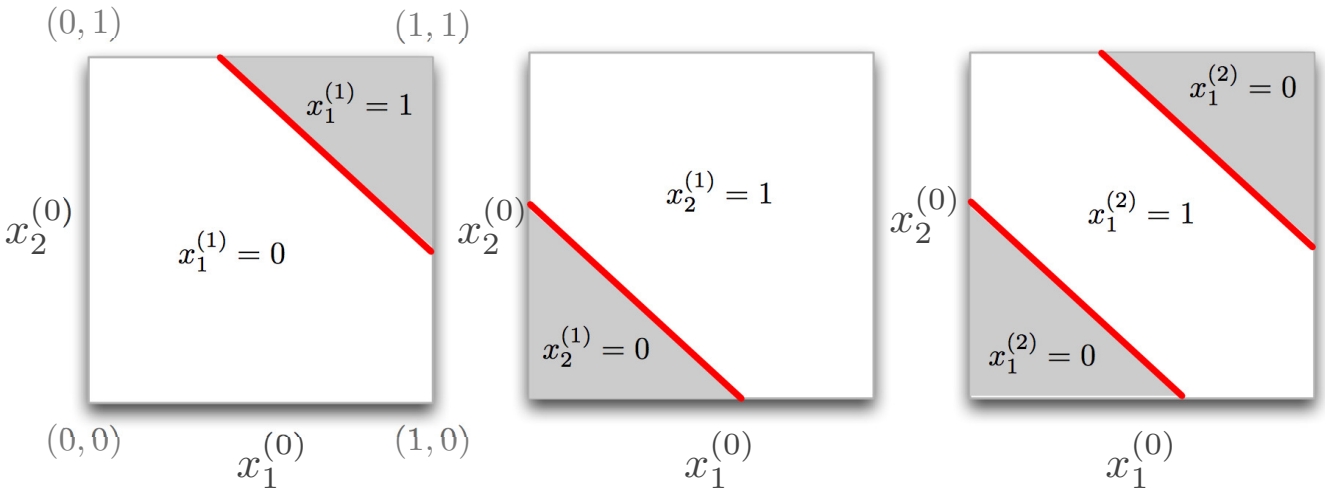


$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

Example: The XOR Problem (continued)

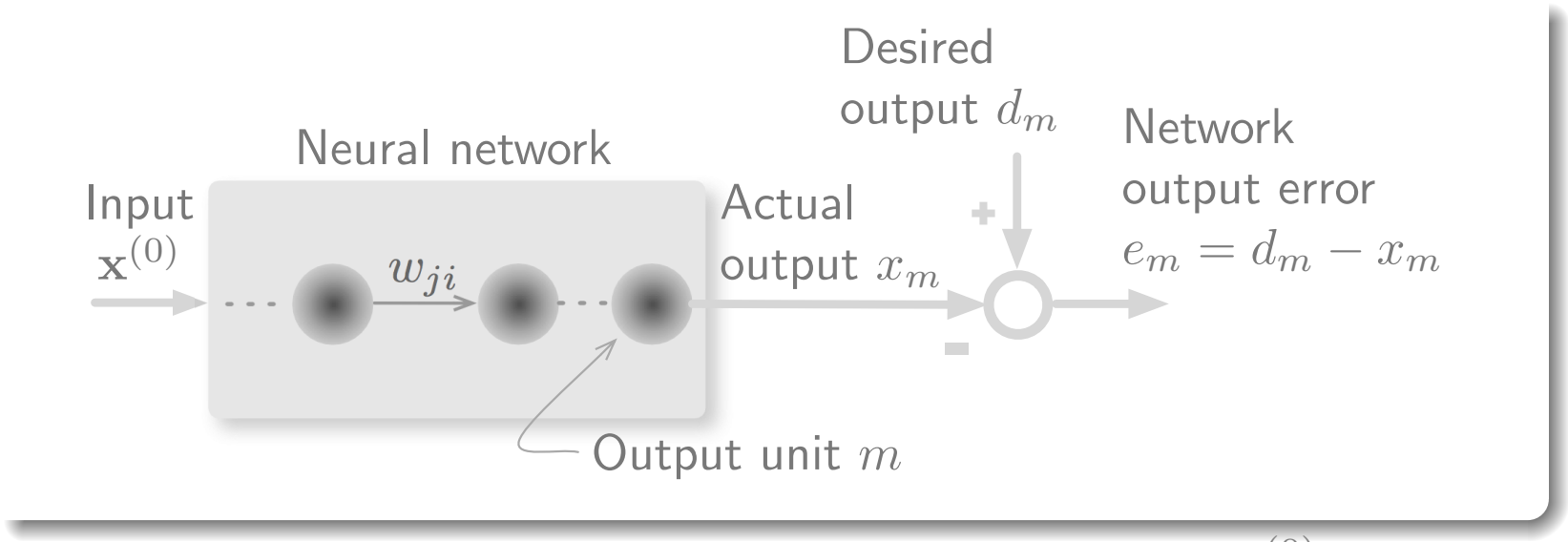


$x_1^{(0)} + x_2^{(0)} - 1.5 = 0$ $\quad x_1^{(0)} + x_2^{(0)} - 0.5 = 0$ $\quad -2x_1^{(1)} + x_2^{(1)} - 0.5 = 0$

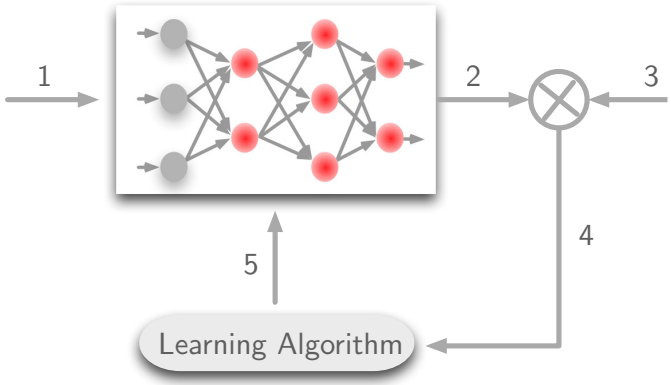


Learning:

- Systematic determination of optimal weights
- Adjust w_{ji} to minimize network output error e



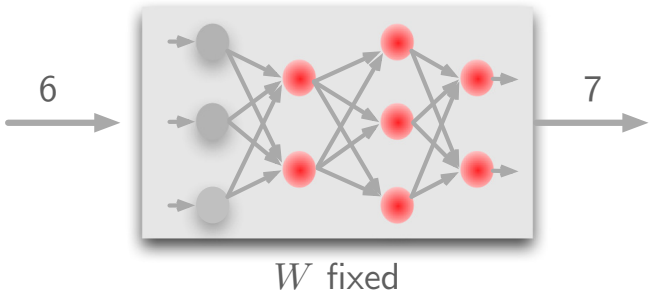
d_m corresponding to each input pattern $\mathbf{x}^{(0)}$ is known



Training a network:

1. Feed input pattern into network
2. Compute network output
3. Compare network output with desired output
4. Determine network output error
5. Run algorithm to adjust weights

Repeat until error is acceptable



Using a network:

- After training, fix weight values

6. Feed input pattern into network
7. Compute network output

- Network with $(L+1)$ layers, indexed as $0, 1, 2, \ldots, L$.
  Layer 0 is the input layer; layer L is the output layer
- Each layer has $(n_l + 1)$ units, where l is the layer index, with 0^{th} unit representing bias
- A training pattern (i.e., example) consists of
  1. a vector $\mathbf{x} = [x_1, x_2, \ldots, x_{n_0}]^T \in R^{n_0 \times 1}$, and
  2. a desired output $\mathbf{d} \in R^{n_L \times 1}$ corresponding to $\mathbf{x}$
- Network learning involves a set of K patterns

$$\{(\mathbf{x}_k, \mathbf{d}_k)\}, k = 1, 2, \ldots, K$$

- During learning, each pattern is presented to network in turn by setting $\mathbf{x}^{(0)}(k) = \mathbf{x}_k$, where $\mathbf{x}^{(0)}(k) = \left[x_1^{(0)}(k), \ldots, x_{n_0}^{(0)}(k)\right]^T$ is the input to network ▸ View

$$\begin{aligned} 1 \text{ iteration} &= 1 \text{ presentation} \\ 1 \text{ epoch} &= K \text{ iterations} \end{aligned}$$

- Define error of unit j in output layer L at iteration k as:

$$e_j(k) = d_j - x_j^{(L)}(k)$$

- Define energy function for network at iteration k as:

$$E(k) = \frac{1}{2}\sum_{j=1}^{n_L} e_j^2(k)$$

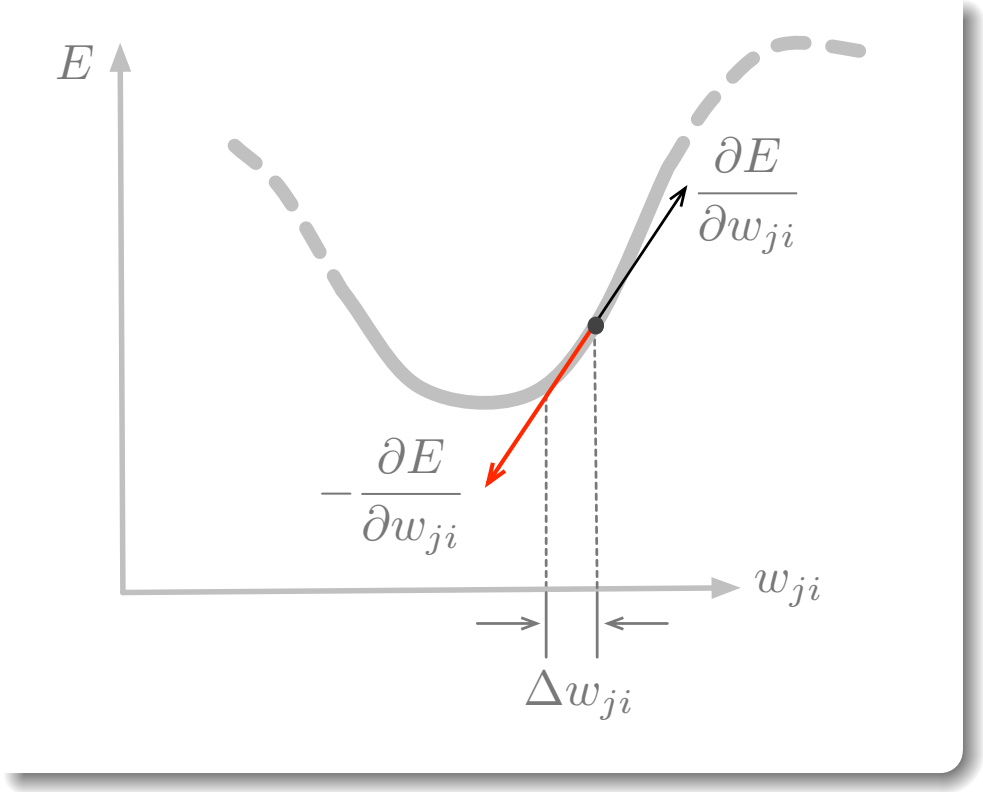
Delta Rule: Adjust weight to reduce error

$$w_{ji}^{(l)}(k+1) = w_{ji}^{(l)}(k) + \Delta w_{ji}^{(l)}(k)$$

$$\Delta w_{ji}^{(l)}(k) = -\eta\left(\frac{\partial E(k)}{\partial w_{ji}^{(l)}(k)}\right)$$

$\eta \in (0, 1]$ is called the learning rate

$$\Delta w_{ji}^{(l)}(k) = -\eta \left(\frac{\partial E(k)}{\partial w_{ji}^{(l)}(k)} \right)$$



η determines size of weight adjustment

In general, E is a function of $\mathbf{W}$, i.e.,

$$E = f\left(\mathbf{x}^{(0)}, \mathbf{d}, \mathbf{W}\right)$$

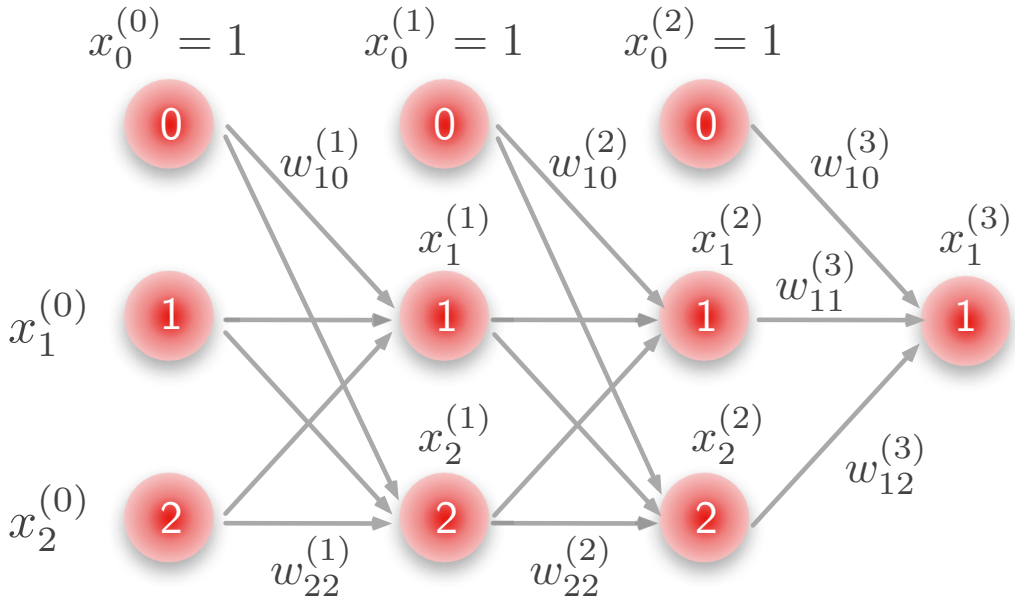
Ideally,

$$E = f\left(\mathbf{x}^{(0)}, \mathbf{d}, \mathbf{W}^*\right) = 0$$

where $\mathbf{W}^*$ is the optimal weight

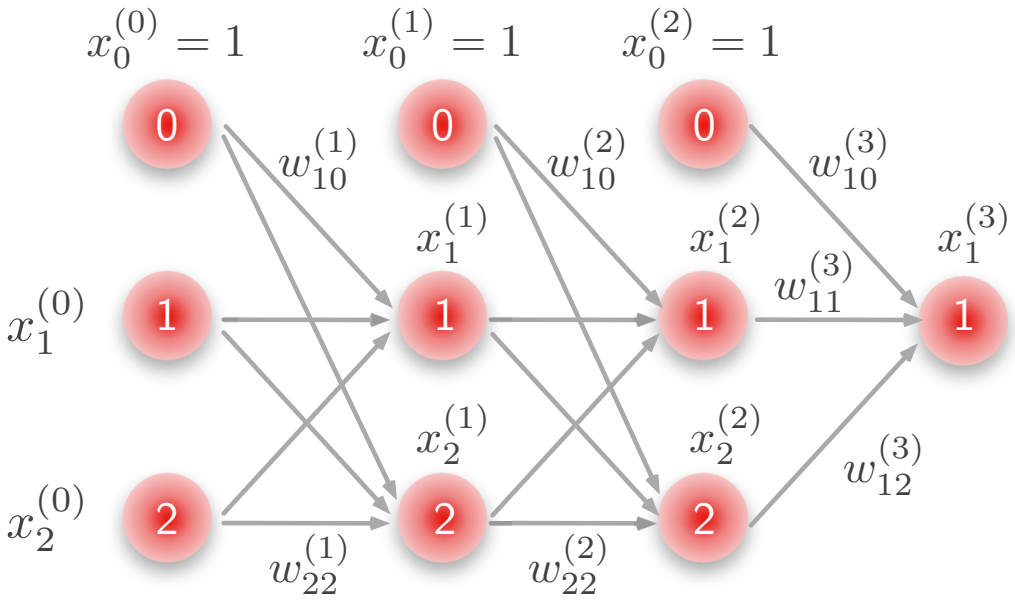
Back

Gradient: Example



$$
\begin{aligned}
v_j^{(l)} &= \sum_{i=1}^{n_{l-1}} w_{ji}^{(l)} \, x_i^{(l-1)} \\
x_j^{(l)} &= \varphi^{(l)} \left(v_j^{(l)} \right)
\end{aligned}
$$

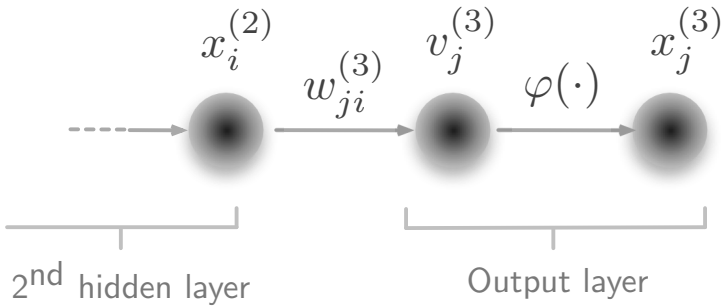
$$
\frac{\partial v_1^{(3)}}{\partial w_{12}^{(3)}} = \frac{\partial}{\partial w_{12}^{(3)}} \left(\sum_{m=0}^{2} w_{1m}^{(3)} \, x_m^{(2)} \right) = \frac{\partial}{\partial w_{12}^{(3)}} \left(w_{10}^{(3)} \, x_0^{(2)} + w_{11}^{(3)} \, x_1^{(2)} + w_{12}^{(3)} \, x_2^{(2)} \right) = x_2^{(2)}
$$



$$
\begin{aligned}
v_j^{(l)} &= \sum_{i=1}^{n_{l-1}} w_{ji}^{(l)}\, x_i^{(l-1)} \\
x_j^{(l)} &= \varphi^{(l)}\left(v_j^{(l)}\right)
\end{aligned}
$$

$$
\frac{\partial v_1^{(3)}}{\partial x_2^{(2)}} = \frac{\partial}{\partial x_2^{(2)}}\left(\sum_{q=0}^{2} w_{1q}^{(3)}\, x_q^{(2)}\right) = \frac{\partial}{\partial x_2^{(2)}}\left(w_{10}^{(3)}\, x_0^{(2)} + w_{11}^{(3)}\, x_1^{(2)} + w_{12}^{(3)}\, x_2^{(2)}\right) = w_{12}^{(3)}
$$

For unit j in output layer, the change in weight from unit i in preceding (i.e., hidden) layer to unit j is:



$$\Delta w_{ji}^{(3)}(k) = -\eta \left(\frac{\partial E(k)}{\partial w_{ji}^{(3)}(k)} \right) = -\eta \left(\frac{\partial E(k)}{\partial v_j^{(3)}(k)} \right) \left(\frac{\partial v_j^{(3)}(k)}{\partial w_{ji}^{(3)}(k)} \right)$$

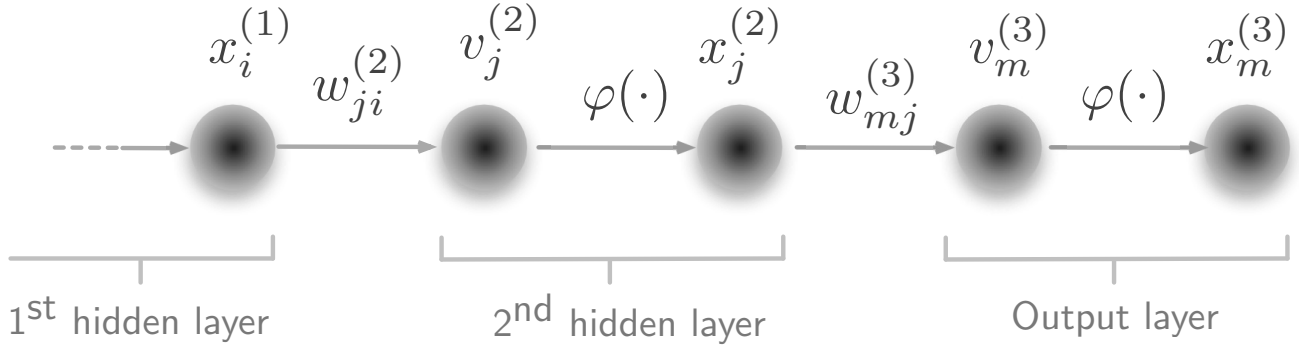
$$\frac{\partial v_j^{(3)}}{\partial w_{ji}^{(3)}} = \frac{\partial}{\partial w_{ji}^{(3)}} \left(\sum_{m=0}^{n_2} w_{jm}^{(3)} x_m^{(2)} \right) = x_i^{(2)} \quad (\text{for } m = i)$$

$$\frac{\partial E}{\partial v_j^{(3)}} = \frac{\partial}{\partial v_j^{(3)}} \left(\frac{1}{2} \sum_{m=1}^{n_3} \left(d_m - x_m^{(3)} \right)^2 \right)$$

$$= -\left(d_j - x_j^{(3)} \right) \frac{\partial x_j^{(3)}}{\partial v_j^{(3)}} \quad (\text{for } m = j)$$

$$= -\left(d_j - x_j^{(3)} \right) \varphi' \left(v_j^{(3)} \right) \equiv -\delta_j^{(3)} \quad \left(\text{where } \varphi'(\cdot) \equiv \frac{\partial \varphi(\cdot)}{\partial(\cdot)} \right)$$

Backpropagation: Hidden Units (for network with two hidden layers)



For unit j in 2^{nd} hidden layer, the change in weight from unit i in 1^{st} hidden layer to unit j is:

$$\Delta w_{ji}^{(2)}(k) = -\eta \left(\frac{\partial E(k)}{\partial w_{ji}^{(2)}(k)} \right) = -\eta \left(\frac{\partial E(k)}{\partial v_j^{(2)}(k)} \right) \left(\frac{\partial v_j^{(2)}(k)}{\partial w_{ji}^{(2)}(k)} \right)$$

$$\frac{\partial v_j^{(2)}}{\partial w_{ji}^{(2)}} = \frac{\partial}{\partial w_{ji}^{(2)}} \left(\sum_{m=0}^{n_1} w_{jm}^{(2)} x_m^{(1)} \right) = \frac{\partial \left(w_{ji}^{(2)} x_i^{(1)} \right)}{\partial w_{ji}^{(2)}} = x_i^{(1)} \quad (\text{for } m = i)$$

$$\Delta w_{ji}^{(2)}(k) = -\eta \left(\frac{\partial E(k)}{\partial w_{ji}^{(2)}(k)} \right) = -\eta \left(\frac{\partial E(k)}{\partial v_j^{(2)}(k)} \right) \left(\frac{\partial v_j^{(2)}(k)}{\partial w_{ji}^{(2)}(k)} \right)$$

$$\begin{aligned}
\frac{\partial E}{\partial v_j^{(2)}} &= \frac{1}{2} \left(\sum_{m=1}^{n_3} \frac{\partial \left(d_m - x_m^{(3)} \right)^2}{\partial v_m^{(3)}} \cdot \frac{\partial v_m^{(3)}}{\partial x_j^{(2)}} \right) \frac{\partial x_j^{(2)}}{\partial v_j^{(2)}} \\
&= \frac{1}{2} \left(\sum_{m=1}^{n_3} \frac{\partial \left(d_m - \varphi \left(v_m^{(3)} \right) \right)^2}{\partial v_m^{(3)}} \cdot \frac{\partial \left(\sum_{q=0}^{n_2} w_{mq}^{(3)} x_q^{(2)} \right)}{\partial x_j^{(2)}} \right) \frac{\partial x_j^{(2)}}{\partial v_j^{(2)}} \\
&= - \left(\sum_{m=1}^{n_3} \underbrace{\left(d_m - x_m^{(3)} \right) \varphi' \left(v_m^{(3)} \right)}_{\delta_m^{(3)}} \underbrace{\frac{\partial \left(w_{mj}^{(3)} x_j^{(2)} \right)}{\partial x_j^{(2)}}}_{(\text{for } q = j)} \right) \frac{\partial \varphi \left(v_j^{(2)} \right)}{\partial v_j^{(2)}} \\
&= - \left(\sum_{m=1}^{n_3} \delta_m^{(3)} w_{mj}^{(3)} \right) \varphi' \left(v_j^{(2)} \right) \equiv -\delta_j^{(2)}
\end{aligned}$$

For output unit j

$$w_{ji}^{(3)}(k+1) = w_{ji}^{(3)}(k) + \Delta w_{ji}^{(3)}(k)$$

$$\begin{aligned} \Delta w_{ji}^{(3)}(k) &= -\eta \left(\frac{\partial E(k)}{\partial w_{ji}^{(3)}(k)} \right) \\ &= \eta\, \delta_j^{(3)}(k)\, x_i^{(2)}(k) \end{aligned}$$

For hidden unit j

$$w_{ji}^{(2)}(k+1) = w_{ji}^{(2)}(k) + \Delta w_{ji}^{(2)}(k)$$

$$\begin{aligned} \Delta w_{ji}^{(2)}(k) &= -\eta \left(\frac{\partial E(k)}{\partial w_{ji}^{(2)}(k)} \right) \\ &= \eta\, \delta_j^{(2)}(k)\, x_i^{(1)}(k) \end{aligned}$$

In general: $$w_{ji}^{(l)}(k+1) = w_{ji}^{(l)}(k) + \eta\, \delta_j^{(l)}(k)\, x_i^{(l-1)}(k)$$

For output layer: $$\delta_j^{(l)}(k) = \left(d_j(k) - x_j^{(l)}(k) \right) \varphi' \left(v_j^{(l)}(k) \right)$$

For hidden layer: $$\delta_j^{(l)}(k) = \left(\sum_{m=0}^{n_{l+1}} \delta_m^{(l+1)}(k) w_{mj}^{(l+1)}(k) \right) \varphi' \left(v_j^{(l)}(k) \right)$$

More details are provided in supplementary notes

Back

- $\varphi(\cdot)$ must be differentiable because of $\varphi'(\cdot)$ in Delta Rule
- For sigmoidal function

$$\varphi(v) = \frac{1}{1+e^{-av}}, \quad a > 0$$

$\varphi'(\cdot)$ can be expressed in terms of $\varphi(\cdot)$, i.e.,

$$\begin{aligned}
\varphi'(v) &\equiv \frac{\partial \varphi(v)}{\partial v} = \frac{a e^{-av}}{(1+e^{-av})^2} \\
&= a\left(\frac{e^{-av}+1-1}{1+e^{-av}}\right) \\
&= a\left(\frac{1+e^{-av}}{1+e^{-av}}\right)\left(\frac{-1}{1+e^{-av}}\right) \\
&= a\,\varphi(v)\Big(1-\varphi(v)\Big)
\end{aligned}$$

- For $\varphi(v) = \tanh(av)$, $\varphi'(v) = a\left(1-\varphi^2(v)\right)$

Let $x = \varphi(v)$. Then $\varphi'(v) = ax(1-x)$

Calculation of δ

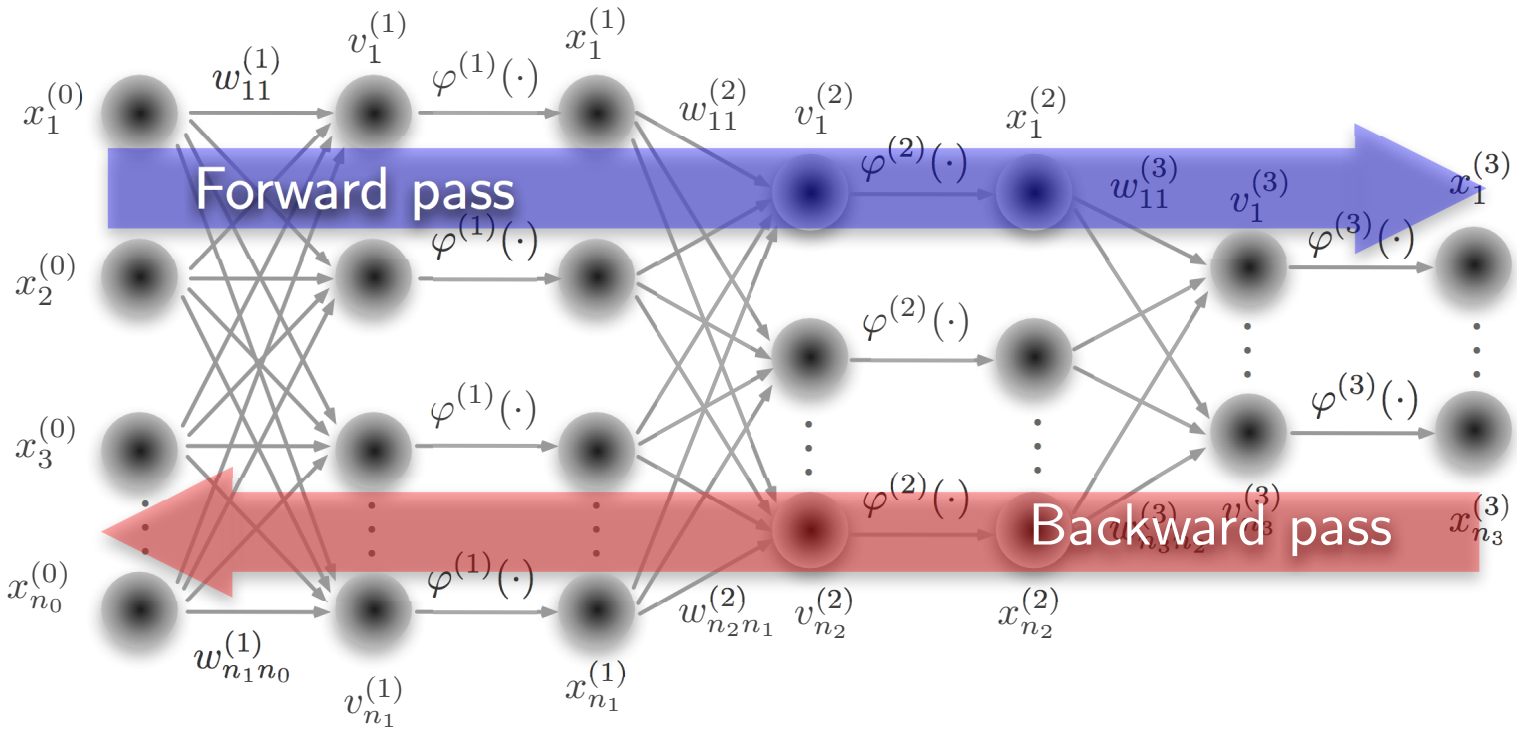
For output layer:

$$\begin{aligned}\delta_j^{(l)} &= \left(d_j - x_j^{(l)}\right)\varphi'\left(v_j^{(l)}\right) \\ &= a x_j^{(l)}\left(1 - x_j^{(l)}\right)\left(d_j - x_j^{(l)}\right)\end{aligned}$$

For hidden layer:

$$\begin{aligned}\delta_j^{(l)} &= \left(\sum_{m=1}^{n_{l+1}} \delta_m^{(l+1)} w_{mj}^{(l+1)}\right)\varphi'\left(v_j^{(l)}\right) \\ &= a\, x_j^{(l)}\left(1 - x_j^{(l)}\right)\sum_{m=0}^{n_{l+1}} \delta_m^{(l+1)} w_{mj}^{(l+1)}\end{aligned}$$

Iteration index k has been omitted in all variables

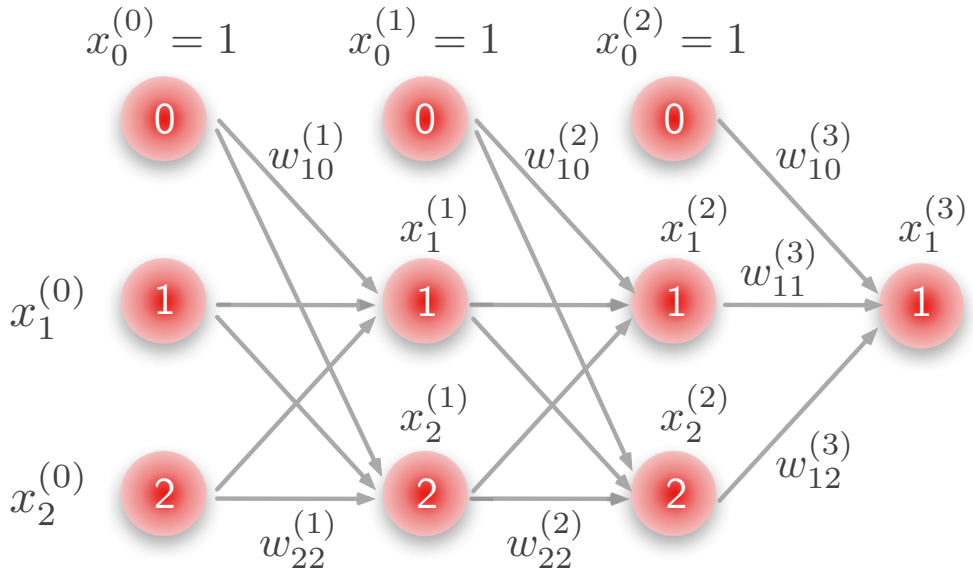


Forward: $v_j^{(l)} = \sum_{i=1}^{n_{l-1}} w_{ji}^{(l)} x_i^{(l-1)}, \quad x_j^{(l)} = \varphi\left(v_j^{(l)}\right)$

Backward: $w_{ji}^{(l)}(k+1) = w_{ji}^{(l)}(k) + \eta\, \delta_j^{(l)}(k)\, x_i^{(l-1)}(k)$

For output layer: $\delta_j^{(l)} = \left(d_j - x_j^{(l)}\right) \varphi'\left(v_j^{(l)}\right)$

For hidden layer: $\delta_j^{(l)} = \left(\sum_{m=1}^{n_{l+1}} \delta_m^{(l+1)} w_{mj}^{(l+1)}\right) \varphi'\left(v_j^{(l)}\right)$



$$\varphi(v) = \frac{1}{1+e^{-av}} = \frac{1}{1+e^{-v}}$$

(i.e., $a = 1$)

Back

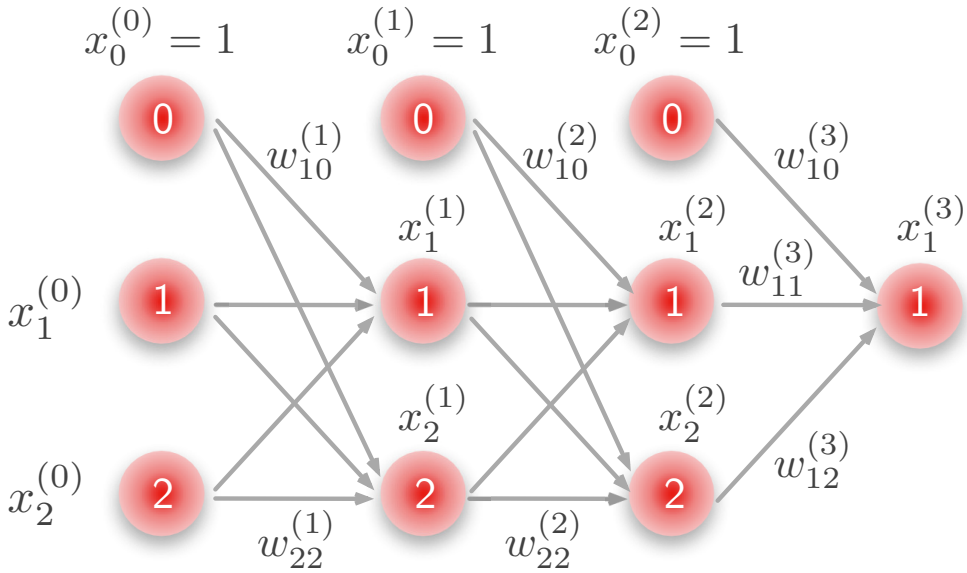
$$v_j^{(1)} = \sum_{i=0}^{2} w_{ji}^{(1)} x_i^{(0)}, \quad x_j^{(1)} = \varphi\left(v_j^{(1)}\right), \quad j = 1, 2$$

$$v_j^{(2)} = \sum_{i=0}^{2} w_{ji}^{(2)} x_i^{(1)}, \quad x_j^{(2)} = \varphi\left(v_j^{(2)}\right), \quad j = 1, 2$$

$$v_j^{(3)} = \sum_{i=0}^{2} w_{ji}^{(3)} x_i^{(2)}, \quad x_j^{(3)} = \varphi\left(v_j^{(3)}\right), \quad j = 1$$

$$
\begin{aligned}
\delta_1^{(3)} &= \left(d_1 - x_1^{(3)}\right) \varphi'\left(v_1^{(3)}\right) \\
\delta_1^{(2)} &= \left(\sum_{m=1}^{1} w_{m1}^{(3)} \delta_m^{(3)}\right) \varphi'\left(v_1^{(2)}\right) = w_{11}^{(3)} \delta_1^{(3)} x_1^{(2)} \left(1 - x_1^{(2)}\right) \\
\delta_2^{(2)} &= \left(\sum_{m=1}^{1} w_{m2}^{(3)} \delta_m^{(3)}\right) \varphi'\left(v_2^{(2)}\right) = w_{12}^{(3)} \delta_1^{(3)} x_2^{(2)} \left(1 - x_2^{(2)}\right) \\
\delta_1^{(1)} &= \left(\sum_{m=0}^{2} w_{m1}^{(2)} \delta_m^{(2)}\right) \varphi'\left(v_1^{(1)}\right) \\
&= \left(w_{11}^{(2)} \delta_1^{(2)} + w_{21}^{(2)} \delta_2^{(2)}\right) x_1^{(1)} \left(1 - x_1^{(1)}\right) \\
\delta_2^{(1)} &= \left(\sum_{m=0}^{2} w_{m2}^{(2)} \delta_m^{(2)}\right) \varphi'\left(v_2^{(1)}\right) \\
&= \left(w_{12}^{(2)} \delta_1^{(2)} + w_{22}^{(2)} \delta_2^{(2)}\right) x_2^{(1)} \left(1 - x_2^{(1)}\right)
\end{aligned}
$$

Example: Backward Pass (continued)



View

$$\Delta w_{10}^{(3)} = \eta\, \delta_1^{(3)} \quad \Delta w_{11}^{(3)} = \eta\, \delta_1^{(3)} x_1^{(2)} \quad \Delta w_{12}^{(3)} = \eta\, \delta_1^{(3)} x_2^{(2)}$$

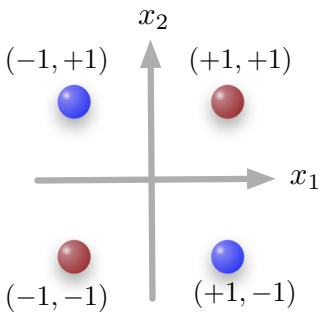
$$\Delta w_{10}^{(2)} = \eta\, \delta_1^{(2)} \quad \Delta w_{11}^{(2)} = \eta\, \delta_1^{(2)} x_1^{(1)} \quad \Delta w_{12}^{(2)} = \eta\, \delta_1^{(2)} x_2^{(1)}$$

$$\Delta w_{20}^{(2)} = \eta\, \delta_2^{(2)} \quad \Delta w_{21}^{(2)} = \eta\, \delta_2^{(2)} x_1^{(1)} \quad \Delta w_{22}^{(2)} = \eta\, \delta_2^{(2)} x_2^{(1)}$$

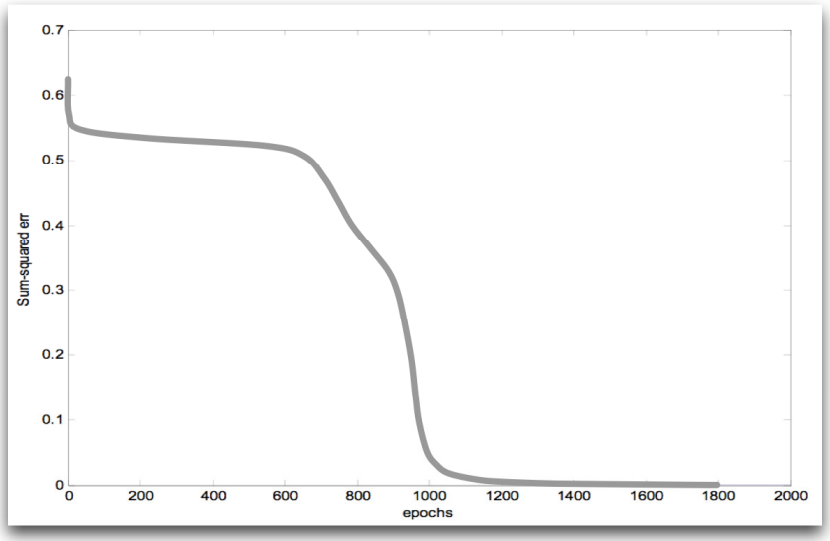
$$\Delta w_{10}^{(1)} = \eta\, \delta_1^{(1)} \quad \Delta w_{11}^{(1)} = \eta\, \delta_1^{(1)} x_1^{(0)} \quad \Delta w_{12}^{(1)} = \eta \delta_1^{(1)} x_2^{(0)}$$

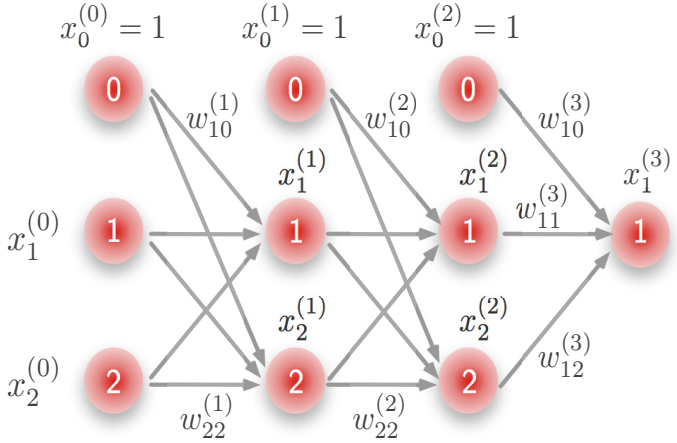
$$\Delta w_{20}^{(1)} = \eta\, \delta_2^{(1)} \quad \Delta w_{21}^{(1)} = \eta\, \delta_2^{(1)} x_1^{(0)} \quad \Delta w_{22}^{(1)} = \eta\, \delta_2^{(1)} x_2^{(0)}$$

Example: XOR



i	$\mathbf{x}_i$	d_i
1	$[-1,-1]^T$	-1
2	$[-1,+1]^T$	$+1$
3	$[+1,-1]^T$	$+1$
4	$[+1,+1]^T$	-1





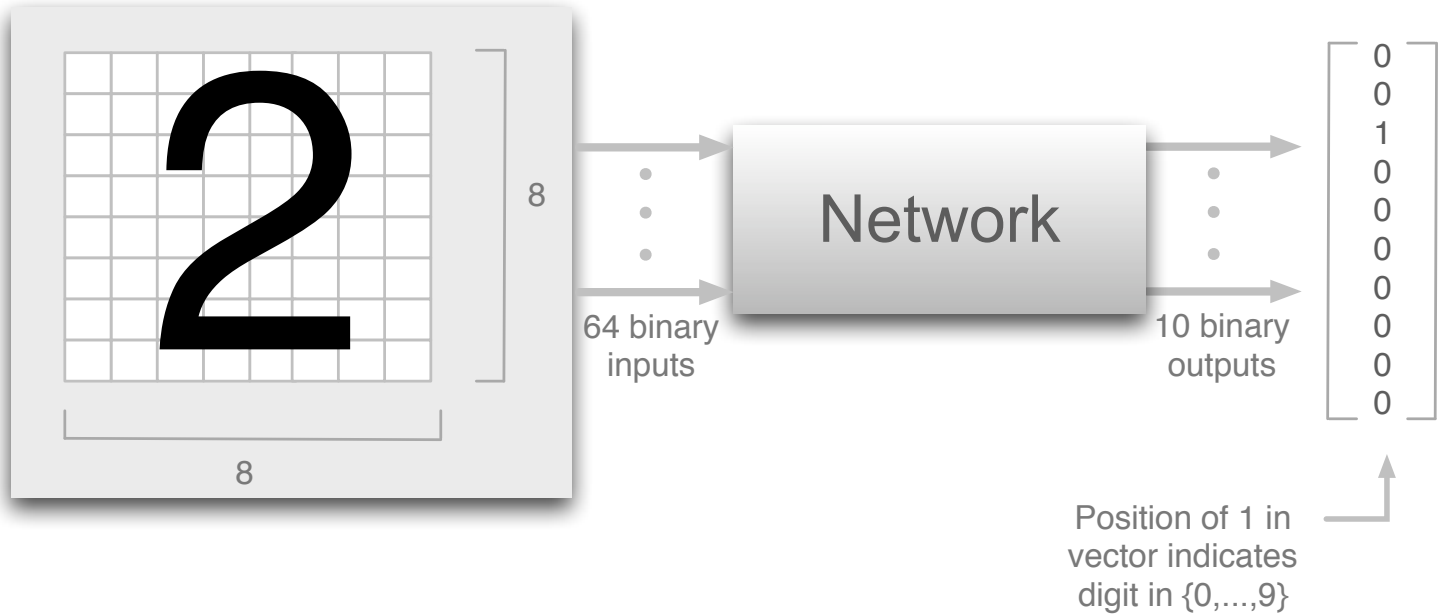
$$\mathbf{w}^{(1)} = \begin{bmatrix} -1.87 & 5.17 & 5.10 \\ -4.66 & 3.-06 & 3.04 \end{bmatrix}$$

$$\mathbf{w}^{(2)} = \begin{bmatrix} -1.06 & 3.69 & -5.29 \\ 2.53 & -4.47 & 3.54 \end{bmatrix}$$

$$\mathbf{w}^{(3)} = \begin{bmatrix} -0.36 & 6.49 & -6.51 \end{bmatrix}$$

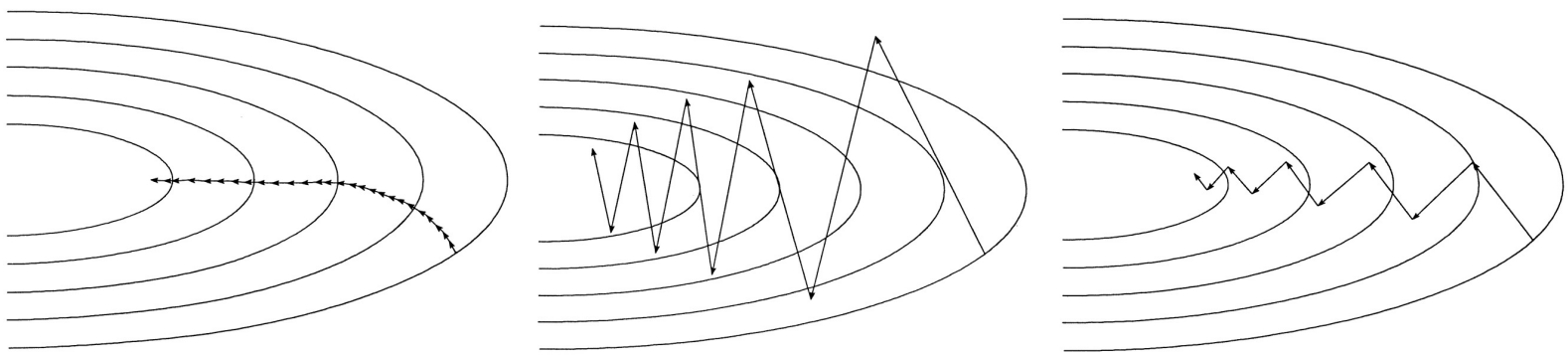
- $\eta = 0.8$ with random initial weights

Example: Optical Character Recognition



$$\Delta w_{ji}^{(l)}(k) = -\eta \left(\frac{\partial E(k)}{\partial w_{ji}^{(l)}(k)} \right)$$

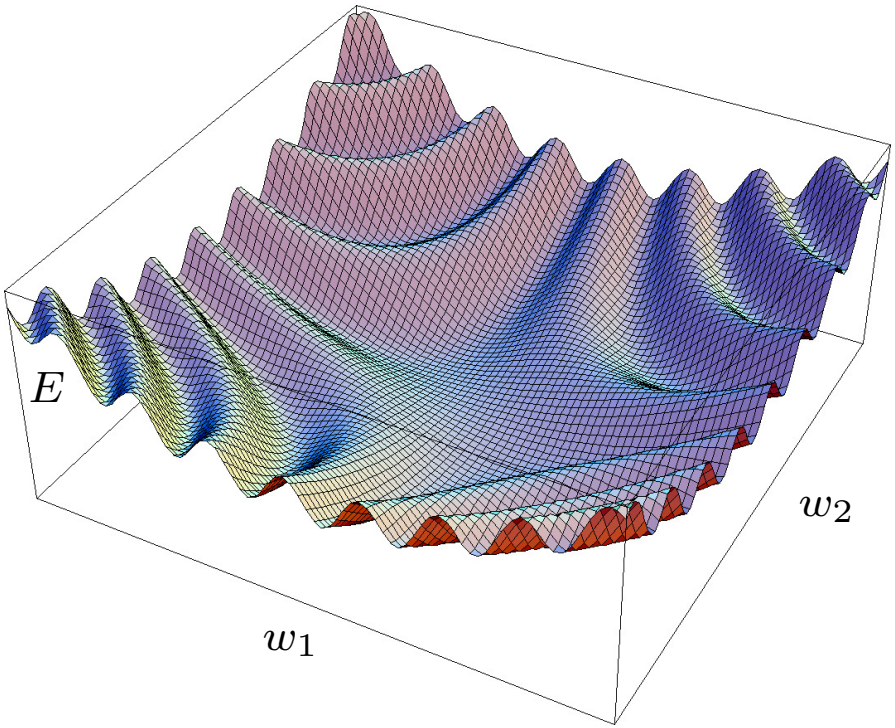
Learning rate η affects speed of gradient descent towards minimum of E ▸ View



Too small | Too large | Preferred

- Small learning rate results in long time to convergence
- Large learning rate results in oscillatory behavior and may lead to instability of learning process

Illustration of error surface



By definition

$$E = f\left(\mathbf{x}^{(0)}, \mathbf{d}, \mathbf{W}\right)$$

Ideally, we want to find optimal weight $\mathbf{W}^*$ such that E reaches global minimum, i.e.,

$$E = f\left(\mathbf{x}^{(0)}, \mathbf{d}, \mathbf{W}^*\right) = 0$$

- However, error-backpropagation algorithm *does not* guarantee convergence to global minimum
- Learning process may get trapped in local minima

- Number of hidden layers

  More hidden layers tend to give more accurate approximation but may also exacerbate problem of local minima

- Number of units in hidden layer

  Too few will lead to high training and generalization error due to underfitting

  Too many may lead to low training error but still lead to high generalization error due to overfitting

- Convergence is not guaranteed
  Although convergence can usually be obtained in practice by selecting suitable learning rate, there is no general proof that learning process will converge
- Convergence to global minimum is not guaranteed
  Gradient descent (if convergent) only guarantees reduction of network output error down to a minimum, but does not guarantee that such minimum is global
- Learning often takes a long time to converge
  Complex problems often need hundreds or thousands of epochs
- Network is essentially a "black box"
  It does not provide an intuitive (e.g., causal) explanation for computed result because what can be learned are operational parameters, not general abstract knowledge of a domain

In Q-learning, we update state-action pair (s_i, a_j) using

$$Q(s_i, a_j) \leftarrow Q(s_i, a_j) + \alpha \Big[\underbrace{r + \gamma \max_{a'} Q(s', a')}_{\text{estimated target value}} - Q(s_i, a_j) \Big]$$

Need the current $Q(s_i, a_j)$ and $Q(s', a')$ values in Q-table

Q-table as function $f(\cdot)$ with input s_i and output $\mathbf{q}$

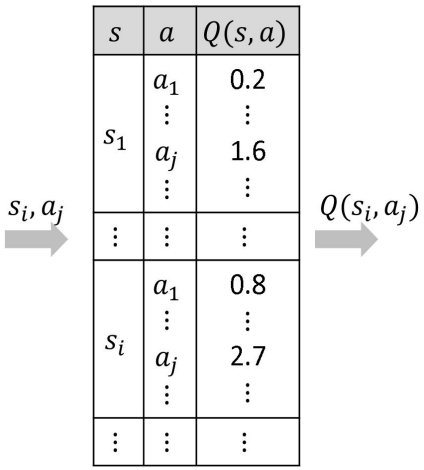
s_i, a_j ⟶

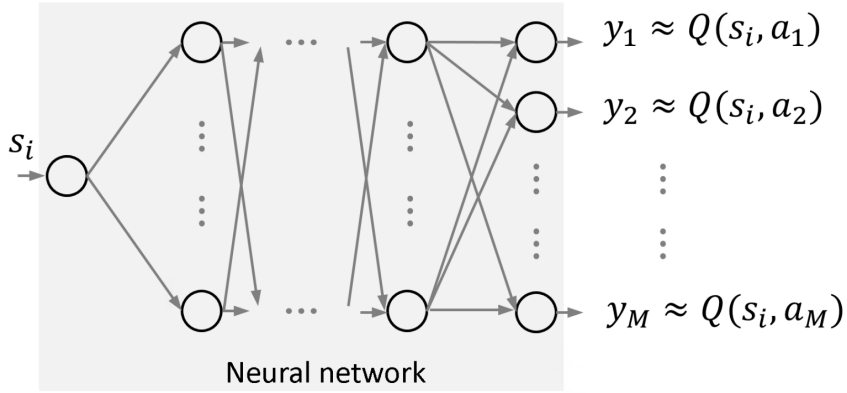
s	a	$Q(s,a)$
s_1	a_1 ⋮ a_j ⋮	0.2 ⋮ 1.6 ⋮
⋮	⋮	⋮
s_i	a_1 ⋮ a_j ⋮	0.8 ⋮ 2.7 ⋮
⋮	⋮	⋮

⟶ $Q(s_i, a_j)$

$$f(s_i) = \mathbf{q} = \begin{bmatrix} Q(s_i, a_1) \\ Q(s_i, a_2) \\ \vdots \\ Q(s_i, a_j) \\ \vdots \\ Q(s_i, a_M) \end{bmatrix}, \quad M = |\mathcal{A}(s_i)|$$

Approximating Q-Table with NN





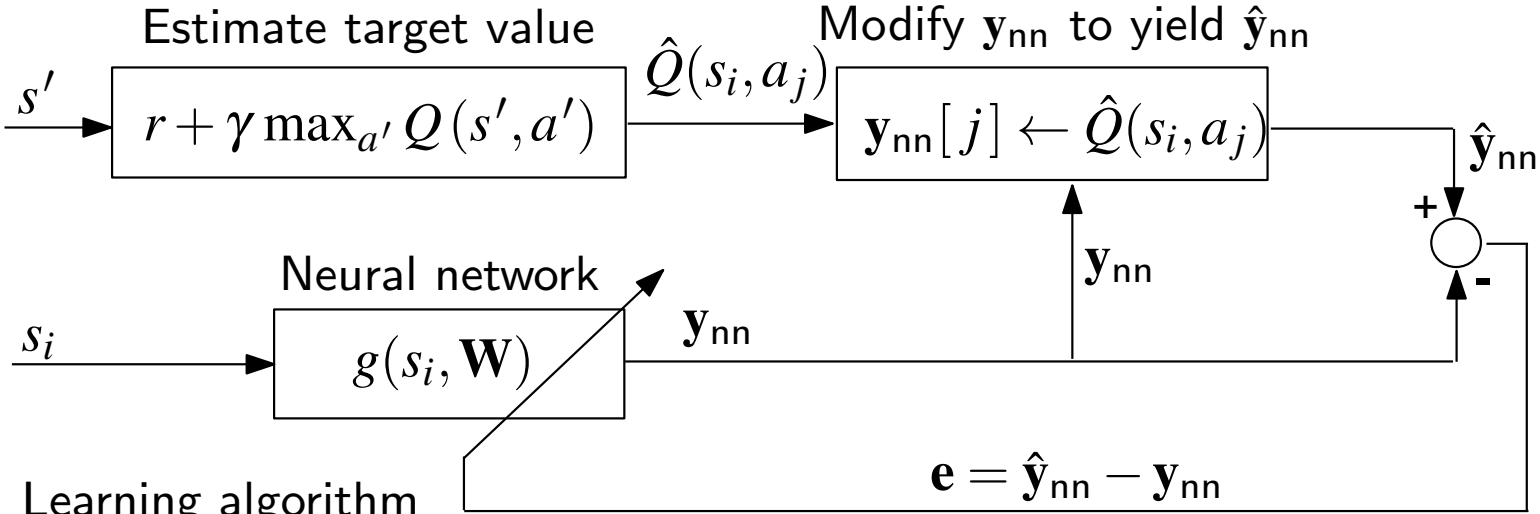
$$\mathbf{y}_{\text{nn}} = g(s_i, \mathbf{W}^*)$$

$$\mathbf{q} - \mathbf{y}_{\text{nn}} = f(s_i) - g(s_i, \mathbf{W}^*) = \begin{bmatrix} Q(s_i, a_1) \\ Q(s_i, a_2) \\ \vdots \\ Q(s_i, a_j) \\ \vdots \\ Q(s_i, a_M) \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} \to \mathbf{0}$$

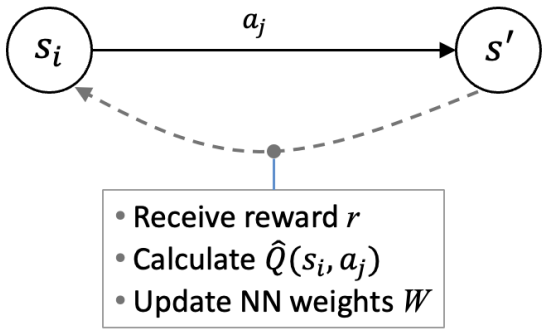
$$Q(s_i, a_j) \leftarrow Q(s_i, a_j) + \alpha \Big[\underbrace{r + \gamma \max_{a'} Q(s', a')}_{\text{estimated target value } \hat{Q}(s_i, a_j)} - Q(s_i, a_j) \Big]$$

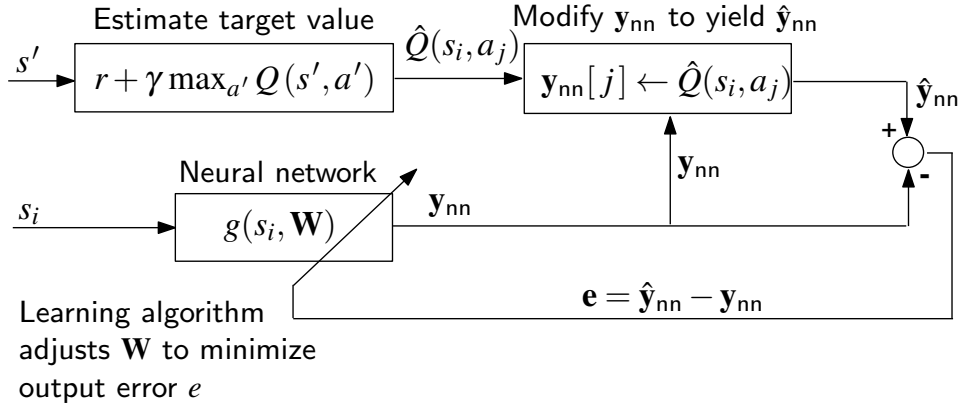
where s' is the next state

Estimated-target-value as desired output for training NN



Learning algorithm adjusts $\mathbf{W}$ to minimize output error e





- At s_i take action a_j. Observe new state s' and receive reward r
- Generate NN output $\mathbf{y}_{\text{nn}} = g(s_i, \mathbf{W})$
- Calculate $\hat{Q}(s_i, a_j) = r + \gamma \max_{a'} Q(s', a')$
- Replace j^{th} element of $\mathbf{y}_{\text{nn}}$ with $\hat{Q}(s_i, a_j)$ to obtain $\hat{\mathbf{y}}_{\text{nn}}$
- Run error-backpropagation algorithm using $\hat{\mathbf{y}}_{\text{nn}}$ as desired output
- Repeats until the weights $\mathbf{W}$ converge

$$\mathbf{y}_{\text{nn}} \triangleq \begin{bmatrix} Q(s, a_1) \\ Q(s, a_2) \\ \vdots \\ Q(s, a_M) \end{bmatrix} = g(s, \mathbf{W}^*) \approx \begin{bmatrix} q_*(s, a_1) \\ q_*(s, a_2) \\ \vdots \\ q_*(s, a_M) \end{bmatrix}, \quad \text{for all } s \in \mathcal{S}$$