DATA130008 Introduction to Artificial Intelligence

复旦大学大数据学院
School of Data Science, Fudan University

魏忠钰

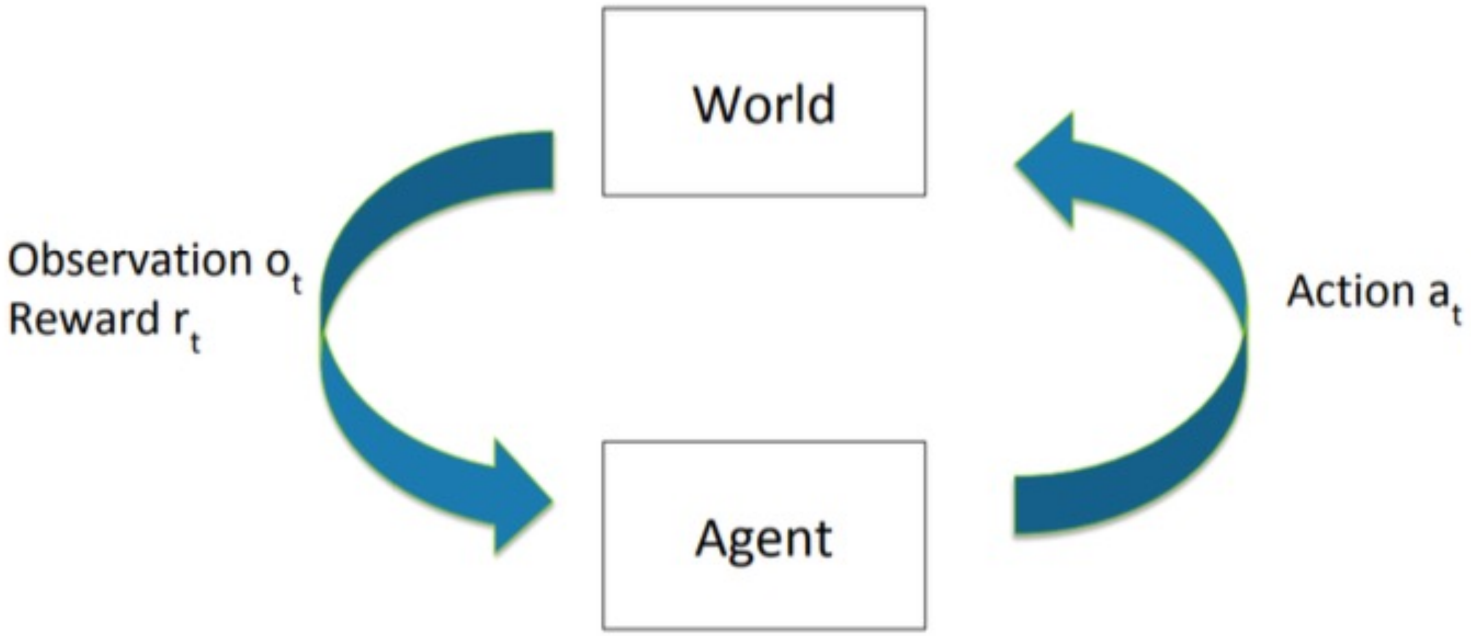
Value Function Approximation

Data Intelligence and Social Computing Lab (DISC)

November 23rd, 2021

Sequential Decision Making

- An agent makes a sequence of actions: $\{a_t\}$
- Observes a sequence of observations: $\{o_t\}$ i.e., $\{s_t\}$
- Receives a sequence of rewards: $\{r_t\}$
- Trajectory / episodes: $s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3$ …



Markov Decision Processes (MDPs)

- Markov Process + Reward + Action
- S is a (finite) set of Markov states s ∈ S
- A is a (finite) set of actions a ∈ A
- P is dynamics / transition-model for each action,

$$P(s_{t+1} = s' | s_t = s, a_t = a)$$

- R is a reward function

$$R(s_t = s, a_t = a) = E[r_t | s_t = s, a_t = a]$$

- γ is discount factor $\gamma \in [0,1]$
- MDP is a tuple: (S, A, P, R, γ)

Important Elements for MDPs

- Trials: Interact with the environment to collect trials

$$\tau = s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T$$

- Return: from time step t

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots$$

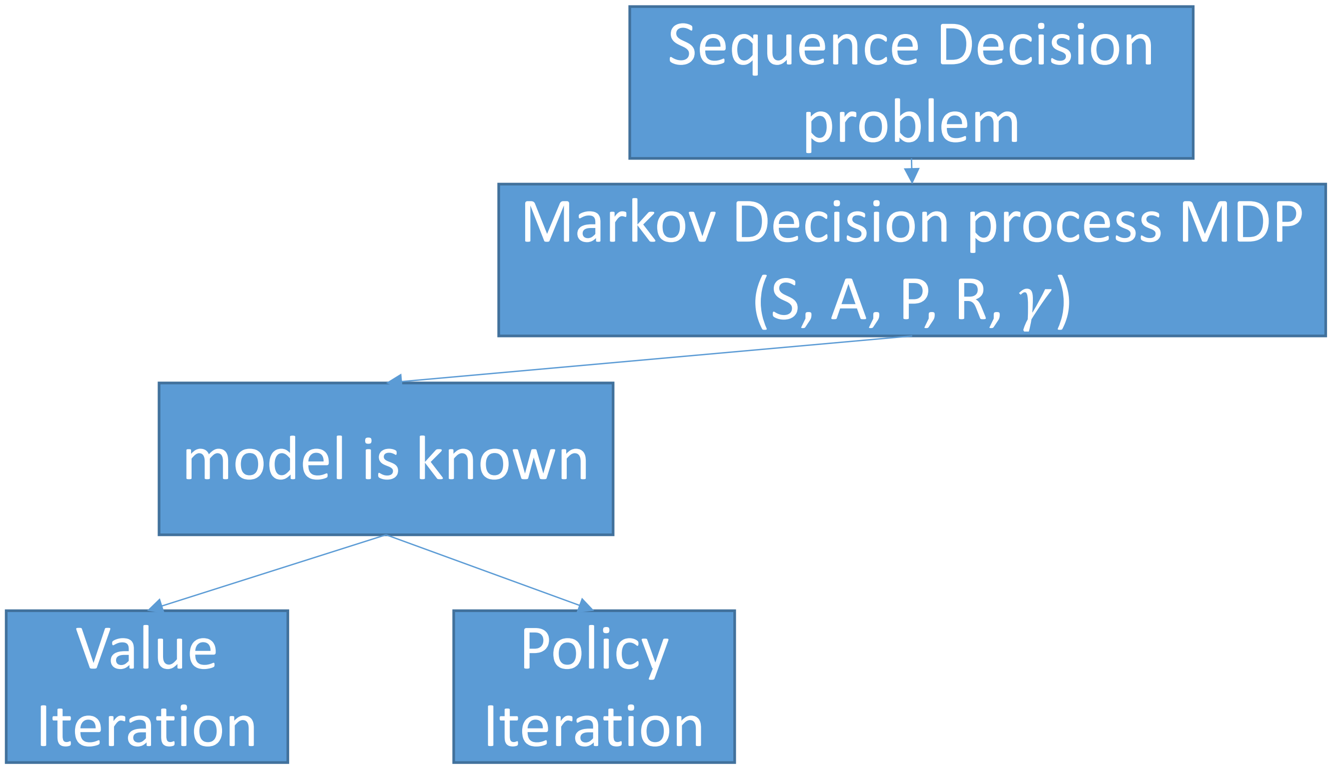
- Value Function: expected return from starting in state s

$$V(s) = E[G_t | s_t = s] = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots | s_t = s]$$

- Policy : a distribution over actions / a one-to-one mapping

$$\pi(s) = \pi(a|s) = P(a_t = a | s_t = s)$$

Summary of Tabular RL learning



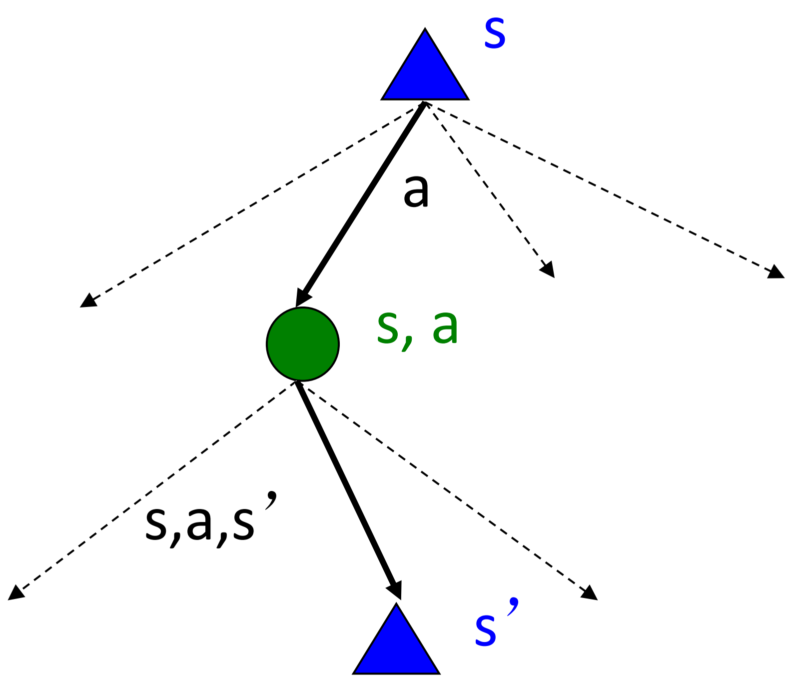
Bellman equations for solving MDPs

- Bellman equation:

$$V^*(s) = \max_{a \in A(s)} Q^*(s,a)$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

$$V^*(s) = \max_{a \in A(s)} \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')\}$$



Value Iteration (Bellman Update Equation)

- Start with $V_0(s) = 0$

- Given vector of $V_k(s)$ values:

$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s')\}$$

- Repeat until convergence

Policy Iteration for solving MDPs

- Step 1: Policy evaluation: calculate values for **fixed policy** until convergence.

$$V^{\pi}(s) \leftarrow R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi}(s')$$

- Step 2: Policy improvement: **update policy** using one-step look-ahead with **current value**

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \{R(s,a) + \sum_{s'} P(s'|s,a) V^*(s')\}$$

- Repeat steps until policy converges

When models are unknown

- Assume a Markov decision process (MDP):
  - A set of states s ∈ S
  - A set of actions (per state) A
- Looking for a policy π(s)

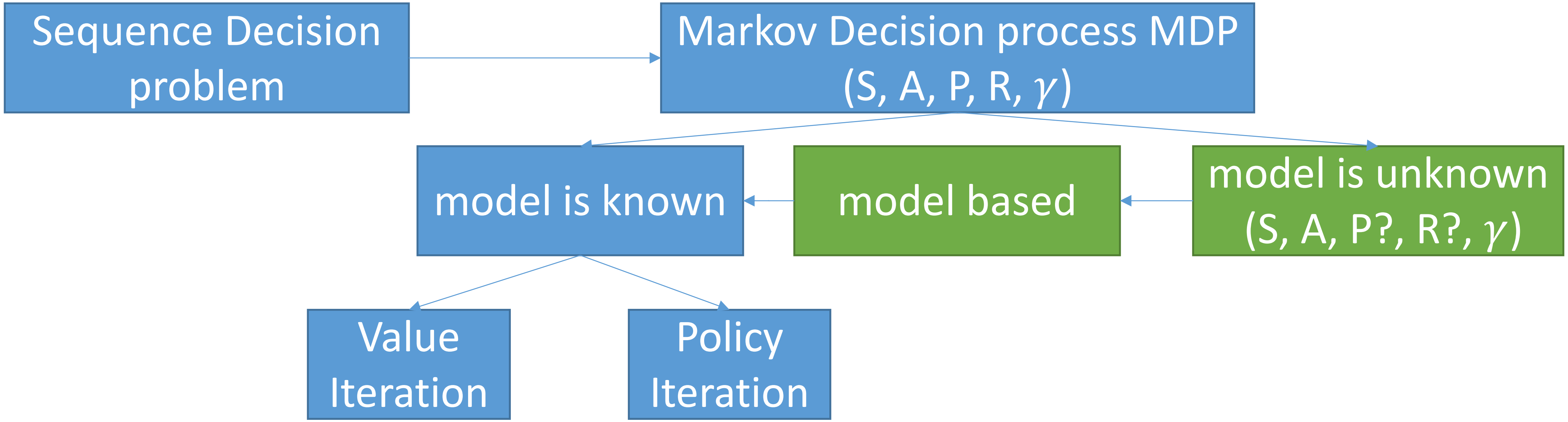
- Try actions and states out to learn, i.e. collect trials

$$\tau = s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T$$

- The return of a trial. $\gamma \in [0,1]$ is the discount factor

$$G(\tau) = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$$

Summary of Tabular RL learning



Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on trials
  - Solve for values as if the learned model were correct

- Step 1: Learn empirical MDP model
  - Count outcomes s’ for each (s, a)
  - Normalize to give an estimate of $P(s_{t+1} = s' | s_t = s, a_t = a)$
  - Discover each $R(s_t = s, a_t = a)$ when we experience (s, a, s’)

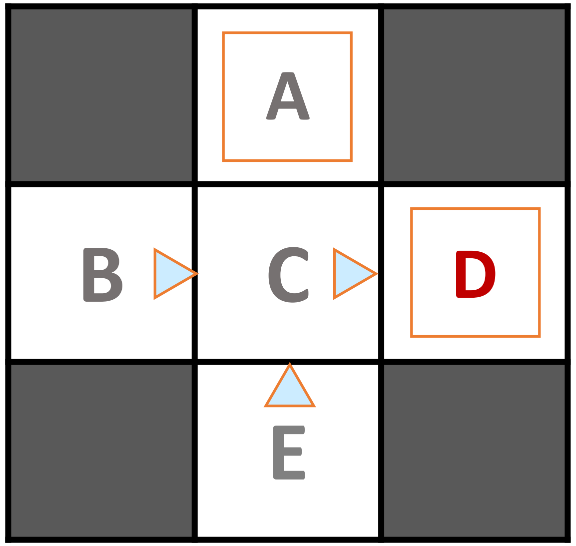
- Step 2: Solve the learned MDP
  - For example, use value iteration

Example: Model-Based Learning

Input Policy π: {(B, east), (C, east), (E, north)}
End states: A and D
Start states: B and E
Assume: $\gamma = 1$

Observed Episodes (Training)

T1: { (B, east, C, -1), (C, east, D, -1), (D, exit, x, +10)}
T2: { (B, east, C, -1), (C, east, D, -1), (D, exit, x, +10)}
T3: { (E, north, C, -1), (C, east, D, -1), (D, exit, x, +10)}
T4: { (E, north, C, -1), (C, east, A, -1), (A, exit, x, -10)}



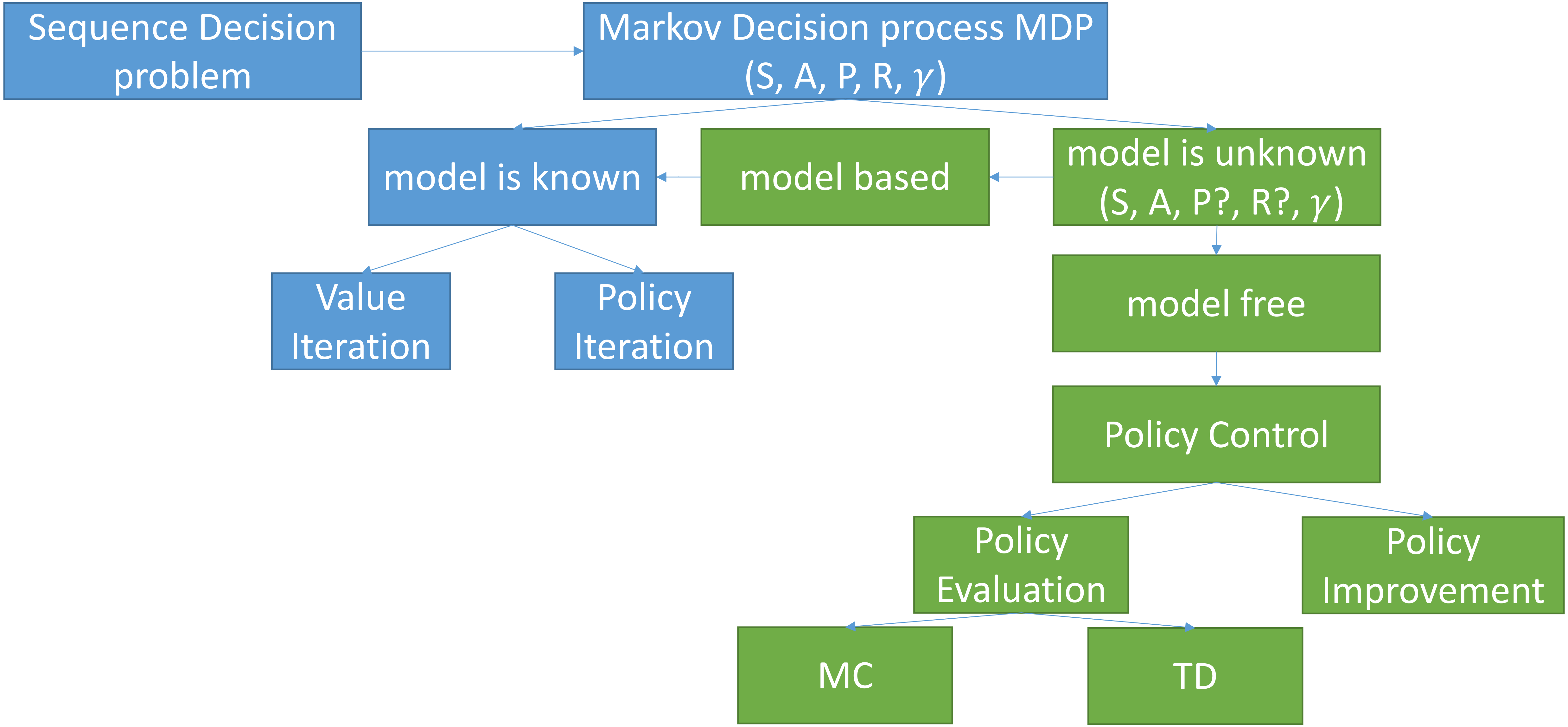
Transition Model

(s, a)	A	B	C	D	E	x	Total #
(A, exit)						1.0	1
(B, east)			1.0				2
(C, east)	.25			.75			4
(D, exit)						1.0	3
(E, north)			1.0				2

Reward Model

(s, a)	reward	Total #	utility
(A, exit)	-10	1	-10
(B, east)	-1, -1	2	-1
(C, east)	-1, -1, -1, -1	4	-1
(D, exit)	10, 10, 10	3	10
(E, north)	-1, -1	2	-1

Summary of Tabular RL learning



Policy Evaluation

- Input: a fixed policy $\pi(s)$
- You don't know the transitions $P(s'|\, s, a)$
- You don't know the rewards $R(s, a, s')$
- Goal: learn state values $V^{\pi}(S)$

- In this case:
  - No choice about what actions to take
  - Just execute the policy and learn from experience

Monte Carlo (MC), i.e., Direct Evaluation

- Given episodes, $s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T$

- Goal: Compute values for each state under π

$$V^{\pi}(s) = E_{\pi}[G_t | S_t = s]$$

- Idea: Average together observed sample values

$$V^{\pi}(s) = \frac{1}{N} \left(\sum_{i=1}^{n} v_i^{\pi}(s)\right)$$

$$v_i^{\pi}(s) = G_t | S_t = s$$

Incremental Monto Carlo (MC)

- After each episode $i = s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \ldots$
- Define $G_{i,t} = r_{i,t} + \gamma r_{i,t+1} + \gamma^2 r_{i,t+2} + \cdots$ as return from time step t onwards in i_{th} episoder
- For state s visited at time step t in episode i
  - $G_i(s)$ is the total return obtained for state s after episode i.
  - Increment counter of total first visits: $N_i(s) = N_{i-1}(s) + 1$
  - Update estimate:

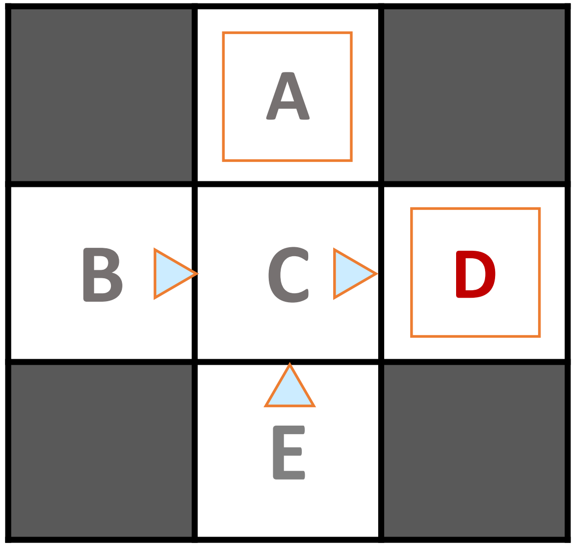
$$V_i^{\pi}(s) = V_{i-1}^{\pi}(s)\left(\frac{N_i(s) - 1}{N_i(s)} + \frac{G_{i,t}}{N(s)}\right) = V_{i-1}^{\pi}(s) + \frac{1}{N_i(s)}\left(G_{i,t} - V_{i-1}^{\pi}(s)\right)$$

Example: Monte Carlo

Input Policy π: {(B, east), (C, east), (E, north)}
End states: A and D
Start states: B and E
Assume: $\gamma = 1$

Observed Episodes

T1: { (B, east, C, -1), (C, east, D, -1), (D, exit, x, +10)}
T2: { (B, east, C, -1), (C, east, D, -1), (D, exit, x, +10)}
T3: { (E, north, C, -1), (C, east, D, -1), (D, exit, x, +10)}
T4: { (E, north, C, -1), (C, east, A, -1), (A, exit, x, -10)}



T1:

$$v^{\pi}(B) = G_{1,1} = (-1) + \gamma(-1) + \gamma * \gamma * (+10) = 8$$
$$v^{\pi}(C) = G_{1,2} = (-1) + \gamma * (+10) = 9$$
$$v^{\pi}(D) = G_{1,3} = +10 = 10$$

T2:

$$v^{\pi}(B) = G_{2,1} = 8; v^{\pi}(C) = G_{2,2} = 9; v^{\pi}(D) = G_{2,3} = 10$$

T3:

$$v^{\pi}(E) = 8; v^{\pi}(C) = 9; v^{\pi}(D) = 10$$

T4:

$$v^{\pi}(E) = -12; v^{\pi}(C) = -11; v^{\pi}(A) = -10$$

	$V^{\pi}(B)$	$V^{\pi}(C)$	$V^{\pi}(D)$	$V^{\pi}(E)$	$V^{\pi}(A)$
T1 updating	8 / 1	9 / 1	10 / 1		
T2 updating	(8+8)/2 = 8	(9+9) / 2 = 9	(10+10)/2 = 10		
T3 updating		(9+9+9)/3 = 9	(10*3)/3=10	8 / 1 = 8	
T4 updating		(9*3-11)/4 = 4		(8-12)/2=-2	-10/1 = -10
Total #	2	4	3	2	1

Temporal Difference Learning

- Aim: estimate $V^{\pi}(s)$ given episodes generated under policy π
  - $s_1, a_1, r_1, s_2, a_2, r_2, \ldots$ where the actions are sampled from π
- Simplest TD learning: update value towards estimated value

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha(\underbrace{[r_t + \gamma V^{\pi}(s_{t+1})]}_{\text{TD target}} - V^{\pi}(s_t))$$

- TD error:

$$\delta_t = r_t + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$$

- Can immediately update value estimate after (s, a, r, s') tuple

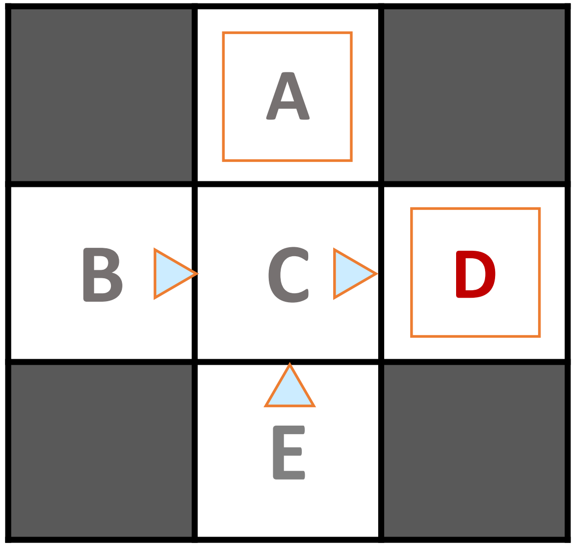
"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning." – Sutton and Barto 2017

Example: Temporal Difference Policy Evaluation

Input Policy π: {(B, east), (C, east), (E, north)}
End states: A and D
Start states: B and E
Assume: γ = 1
Learning rate: alpha = 0.5
States value initialized as 0 (including x)

Observed Episodes (Training)

T1: { (B, east, C, -1), (C, east, D, -1), (D, exit, x, +10)}
T2: { (B, east, C, -1), (C, east, D, -1), (D, exit, x, +10)}
T3: { (E, north, C, -1), (C, east, D, -1), (D, exit, x, +10)}
T4: { (E, north, C, -1), (C, east, A, -1), (A, exit, x, -10)}



Update for T1

	$V^\pi(B)$	$V^\pi(C)$	$V^\pi(D)$	$V^\pi(E)$	$V^\pi(A)$
Initial	0	0	0	0	0
(B, east, C, -1)	-0.5	0	0	0	0
(C, east, D, -1)	-0.5	-0.5	0	0	0
(D, exit, x, +10)	-0.5	-0.5	5	0	0

$$V^\pi(s_t) = V^\pi(s_t) + \alpha(\underbrace{[r_t + \gamma V^\pi(s_{t+1})]}_{\text{TD target}} - V^\pi(s_t))$$

Temporal Difference VS Monte Carlo

- With episode $s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T$
- **Both update the state value by samples**

$$V^{\pi}(s) = V^{\pi}(s) + \alpha(v^{\pi}(s) - V^{\pi}(s))$$

- **Monte Carlo**

$$v^{\pi}(s) = \sum_{k=0}^{T-1} \gamma^k r_{t+k} | S_t = s$$

- **Temporal Difference**

$$v^{\pi}(s) = r_t + \gamma V^{\pi}(S_{t+1}) | S_t = s$$

- In TD, use (s, a, r s’) to update V(S)
  - O(1) operation per update; in an episode of length L, O(L)
- In MC have to wait till episode finishes, then also O(L)
- TD exploits Markov structure

Model-free Control 權值最優

- You don’t know the transitions $P(s'|\, s, a)$
- You don’t know the rewards $R(s, a)$
- You choose the actions now
- Goal: learn the optimal policy / values： ｜S｜X｜A｜values

- In this case:
  - **On-policy** training VS **off-policy** training
  - Fundamental tradeoff: **exploration** vs. **exploitation**

On and Off-Policy Learning

- **Behavior policy $\boldsymbol{\pi_2}$** : gather experience from
- **Estimated policy $\boldsymbol{\pi_1}$** : the one you want to estimate the value of

- On-policy learning ($\pi_1 = \pi_2$)
  - Learn to estimate a policy from experience obtained from following that policy
- Off-policy learning ($\pi_1 \neq \pi_2$) 通常都是在異策略進行訓練
  - Learn to estimate a policy using experience gathered from following a different policy

✓ Sometimes trying actions out is costly

✓ Would like to use historical data from old policies

Sample-based Policy Evaluation

- Sample Episodes following a policy $\boldsymbol{\pi_2}$
  - $s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T$

- Compute values for each state under $\boldsymbol{\pi_1}$
  - $Q^{\boldsymbol{\pi_1}}(s, a)$

- **Behavior policy $\boldsymbol{\pi_2}$** : gather experience from
- **Estimated policy $\boldsymbol{\pi_1}$** : the one you want to estimate the value of

Monte Carlo (MC) Off Policy Evaluation

- Aim: estimate value of policy π_1, $V^{\pi_1}(s)$, given episodes generated under behavior policy π_2
  - $s_1, a_1, r_1, s_2, a_2, r_2, \ldots$ where the actions are sampled from π_2
- $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots$ in MDP M under policy π
- $V^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s]$
- Have data from a different policy, behavior policy π_2

π_2
Sample generation

π_1
Value update

- Note that, not all G_t can be used in this case.

Exploration with ε-greedy policy

- Let |A| be the number of actions
- Then an **ε-greedy policy** w.r.t. a state-value $Q^{\pi}(s,a)$ is

$$\pi^{\epsilon}(s) = \begin{cases} a, & \textit{with probabiliy of } \frac{\varepsilon}{|A|} \\ \underset{a}{\operatorname{argmax}}\, Q^{\pi}(s,a), & \textit{with probabiliy of } 1 - \varepsilon \end{cases}$$

都用argmax的話有些決策不會被檢視到

Monte Carlo for On-policy Policy Iteration

Initialize $Q(s,a)=0$, $N(s,a)=0\ \forall(s,a)$, Set $\epsilon=1$, $k=1$

$\pi_k=\epsilon\text{-greedy}(Q)$ // Create initial ϵ-greedy policy

loop

  Sample k-th episode $(s_{k,1},a_{k,1},r_{k,1},s_{k,2},\ldots,s_{k,T})$ given π_k

  $G_{k,t}=r_{k,t}+\gamma r_{k,t+1}+\gamma^2 r_{k,t+2}+\cdots\gamma^{T_i-1}r_{k,T_i}$

π_{k}
Sample generation

  for $t=1,\ldots,T$ **do**

    if First visit to (s,a) in episode k **then**

      $N(s,a)=N(s,a)+1$

      $Q(s_t,a_t)=Q(s_t,a_t)+\frac{1}{N(s,a)}(G_{k,t}-Q(s_t,a_t))$

    end if

π_{k}
Value update

  end for

  $k=k+1$, $\epsilon=1/k$

  $\pi_k=\epsilon\text{-greedy}(Q)$ // Policy improvement

end loop

$\pi_{\mathrm{k+1}}$
Policy Improvement

Policy Evaluation without model using TD

- MC: Sample Episodes
  - $s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_T$
- TD: Sample Transitions
  - $(s_1, a_1, r_1, s_2, a_2), (s_2, a_2, r_2, s_2, a_2), \ldots, (s_{T-1}, a_{T-1}, r_{T-1}, s_T, a_T)$
- Compute values for each state
  - $Q^{\boldsymbol{\pi}} \quad (s, a)$
- TD based policy evaluation updates values with transition samples.

Q-Learning: Learning towards Optimal Q(s,a)

- Key idea: Maintain state-action Q estimates and use the value of the best future action for bootstrap update

- SARSA: $(s_1, a_1, r_1, s_2, a_2)$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(\boxed{(r_t + \gamma Q(s_{t+1}, \underline{a_{t+1}}))} - Q(s_t, a_t))$$

- Q-learning: $(s_1, a_1, r_1, s_2, argmax(s_2))$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(\boxed{(r_t + \gamma \max_{a'} Q(s_{t+1}, a'))} - Q(s_t, a_t))$$

Refresh your knowledge

- In tabular MDPs, if using a decision policy that visits all states an infinite number of times, and in each state **randomly selects** an action.

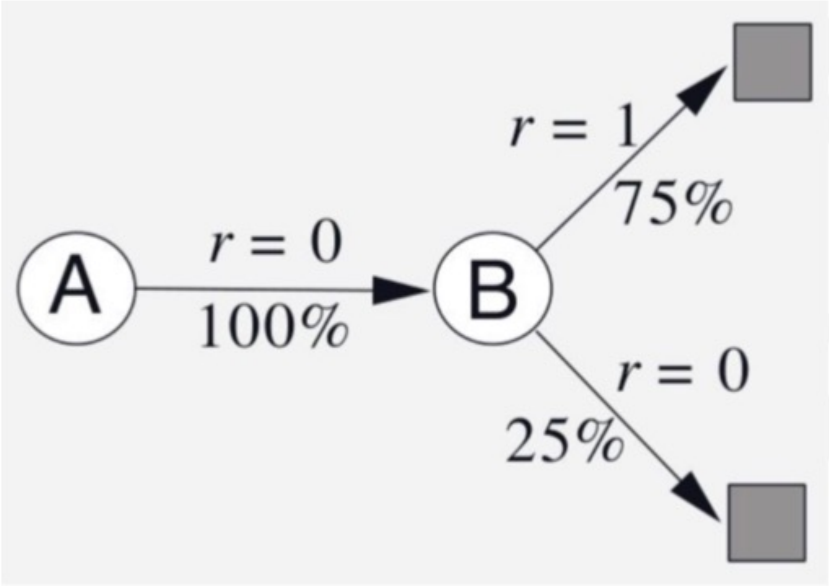
Experience Replay Does this	T or F
Q-learning will converge to the optimal Q-values	T
SARSA will converge to the optimal Q-values	F
Q-learning is learning off-policy	~~F~~ T
SARSA is learning ~~off~~-policy on	~~T~~ F

Batch MC and TD

- Batch (Offline) solution for finite dataset
  - Given set of K episodes
  - Repeatedly sample an episode from K
  - Apply MC or TD to the sampled episode

Batch MC and TD: AB Example

- Two states A,B with $\gamma = 1$
- Given 8 episodes of experience:
  - A, 0, B, 0
  - B, 1 (observed 6 times)
  - B, 0



- Run the 8 episodes once：
  - $V^{MC}(A) =$; $V^{MC}(B) =$
  - $V^{TD}(A) =$; $V^{TD}(B) =$

- Run the 8 episodes infinite number of times：
  - $V^{TD}(A) =$; $V^{TD}(B) =$
  - $V^{MC}(A) =$; $V^{MC}(B) =$

Outline

- States Generalization

Generalizing Across States

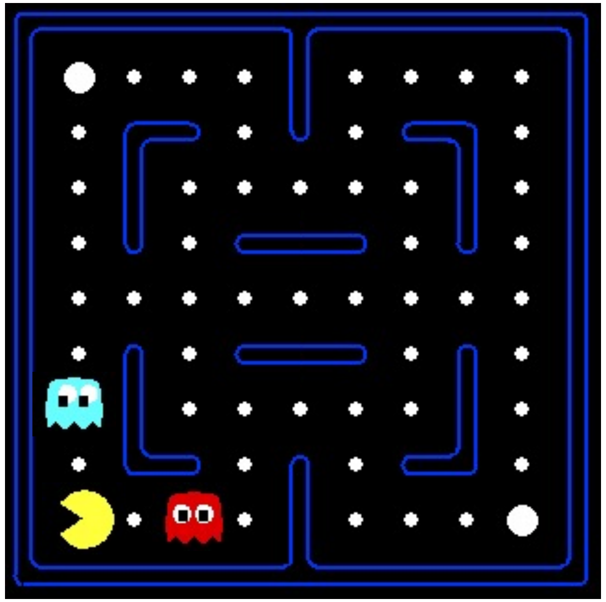
- Tabular RL keeps a table of all state and q values

- In realistic situations, we cannot possibly learn about every single state.
  - Storage: too many states to hold the tables in **memory**
  - Experience: too many **trajectories** to visit every single state
  - Computation: too much **iterations** to take

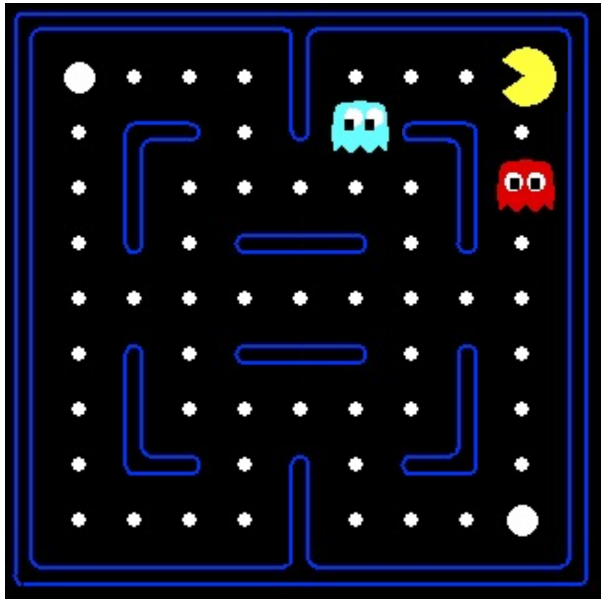
- We want more compact representation that generalizes across states or states and actions

Example: Pacman

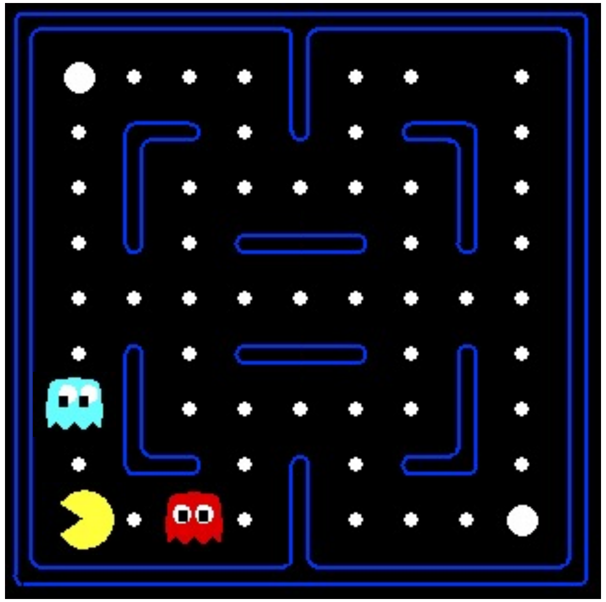
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:

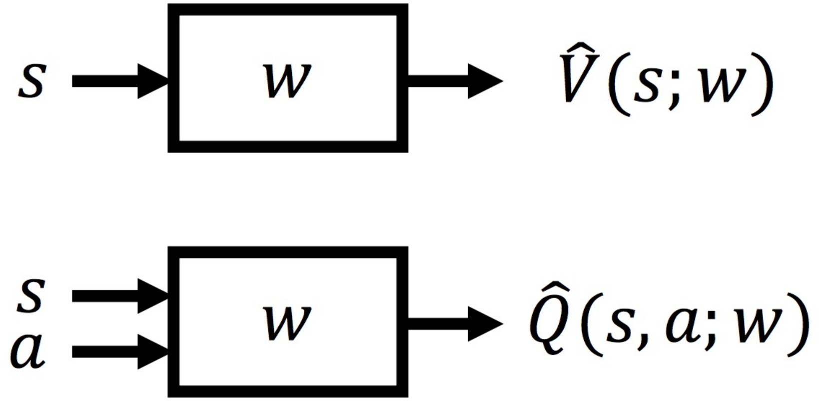


Or even this one!



Approximate Value Functions

- Represent a (state-action/state) value function with a parameterized function instated of a table



- Map a larger space (original state space) into a smaller one (parameter space). And similar states can be mapped closer for generalization.
- Advantages of Generalization
  - Reduce memory needed to store (P, R)/V/Q/π
  - Reduce experience needed to find a good P, R/V/Q/ π
  - Reduce computation needed to compute (P, R)/V/Q/ π

Outline

- States Generalization
- Function Approximation Preliminaries

Function Approximators

- Many possible function approximators including
  - Linear combinations of features
  - Neural networks
  - Decision trees
- linear combinations of features
  - Feature extractor

$$\boldsymbol{f}(s) = (f_1(s), f_2(s), \ldots, f_n(s))$$

  - Weight vector

$$\mathbf{w} = (w_1, w_2, \ldots, w_n)$$

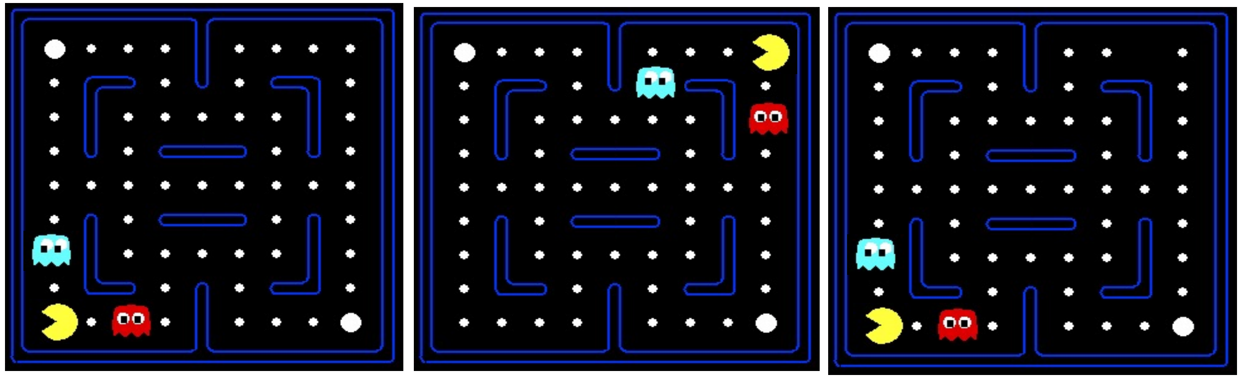
  - Approximated values

$$\hat{V}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

$$\hat{Q}(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \cdots + w_n f_n(s, a)$$

Linear Combination of Features

- Features are functions from states to real numbers that capture important properties of the state
  - f_1: Distance to closest ghost
  - f_2: Distance to closest dot
  - f_3: Number of ghosts
  - f_4: Is Pacman in a tunnel? (0/1)



s_1 s_2 s_3

  - $f(s_1) = f(s_2) = f(s_3) = (2,1,2,1)$
  - w: $[1,1,1,1]$
  - $\hat{V}(s;w) = \mathrm{w}_1 * \mathrm{f}_1(s) + \mathrm{w}_2 * \mathrm{f}_2(s) + \mathrm{w}_3 * \mathrm{f}_3(s) + \mathrm{w}_4 * \mathrm{f}_4(s)$

Recall Gradient Descent

- Consider an **objective function J(w)** is the loss between true value and the approximated value. It is a differentiable function of a parameter vector **w**

$$J(\boldsymbol{w}) = E \quad [(y - \hat{f}(y; \boldsymbol{w}))^2]$$

- Goal is to find parameter **w** that minimizes J.
- The gradient of **J(w)** is :

$$\nabla_w J(\boldsymbol{w}) = [\frac{\partial J(\boldsymbol{w})}{\partial w_1}, \frac{\partial J(\boldsymbol{w})}{\partial w_2}, \frac{\partial J(\boldsymbol{w})}{\partial w_3}, \dots, \frac{\partial J(\boldsymbol{w})}{\partial w_n}]$$

- Compute the gradient and update parameters with a learning rate

$$\mathbf{w} \leftarrow \boldsymbol{w} - \alpha \nabla_w J(\boldsymbol{w})$$

- Until converge

Outline

- States Generalization
- Function Approximation Preliminaries
- Value Function Approximation (VFA) for Policy Evaluation

Model free policy evaluation

- Recall model-free policy evaluation
  - Following a fixed policy π
  - Goal is to estimate V^{π}

$$V_t{}^{\pi}(s) = V_{t-1}{}^{\pi}(s) + \alpha((v^{\pi}(s) - V_{t-1}{}^{\pi}(s))$$

- Maintained a look up table to store V^{π}
- Updated these estimates after each episode (Monte Carlo methods) or after each step (TD methods)

- In value function approximation, change the estimate update step to include fitting the function approximator

VFA for Policy Evaluation

- **True Value:** Assume we could query any state s and an oracle would return the true value for $V^{\pi}(S)$.

- **Approximation:** Find the **approximate representation** for V^{π} given a particular parameterized function $\hat{V}^{\pi}(S;\boldsymbol{w})$.

- **Goal:** Find the parameter vector **w** that minimizes the loss between a true value function $V^{\pi}(s)$ and its approximation $\hat{V}^{\pi}(S;\boldsymbol{w})$.

- How to learn these parameters?

Linear Value Function Approximation

- Represent a value function with a weighted linear combination of features. And f(s) is the feature vector.

$$\hat{V}(s;\boldsymbol{w}) = \sum_{j=1}^{n} f_j(s) * w_j = \boldsymbol{f}(s)^T\boldsymbol{w}$$

- Given $V^{\pi}(s)$ is the true state value of s, Objective function can be:

$$J(\boldsymbol{w}) = E_{\pi}[(V^{\pi}(s) - \hat{V}(s;\boldsymbol{w}))^2]$$

- Weight update for gradient descent is:

$$\Delta\boldsymbol{w} = -\frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\alpha}\nabla_{w}J(\boldsymbol{w}) \qquad \nabla_{w}J(\boldsymbol{w}) = E_{\pi}[-2 * (V^{\pi}(s) - \hat{V}(s;\boldsymbol{w})) * \nabla_{w}\hat{V}(s;\boldsymbol{w})]$$

- Weight update using stochastic gradient descent, using linear function

$$\Delta w = \overset{\text{學習力}}{\alpha} * ((V^{\pi}\overset{\text{查值}}{(s) -} \boldsymbol{f}(s)^T\boldsymbol{w}) * \overset{\text{特徵值}}{\boldsymbol{f}(s)})$$

- Update equals = step-size * prediction error * feature value

Outline

- States Generalization
- Function Approximation Preliminaries
- Value Function Approximation (VFA) for Policy Evaluation
- MC and TD for VFA in Policy Evaluation

Difference of MC and TD in VFA in Policy Evaluation

$$\Delta w = \alpha * \left(V^{\pi}(s) - \hat{V}^{\pi}(s; \boldsymbol{w}) \right) * \nabla_w \hat{V}^{\pi}(s; \boldsymbol{w})$$

- They can share the same approximation function.
- They can use the same way to search for the optimized parameters.

- TD and MC differs in the way of obtaining true value for each sample.

結束後計算累積回報

- MC obtain a sample value after each episode, while TD can update the parameter after each transition

Monte Carlo Value Function Approximation

- Use the accumulated rewards as the ground true value for updating.

$$G_{\mathrm{t}} = \sum_{t=1}^{T} \gamma^{t-1} r_t$$

- Do supervised learning on a set of (state, return) pairs
  - $(s_1, G_1), (s_2, G_2), (s_3, G_3), \ldots, (s_T, G_T)$
  - Substitute G_t for the true $V^{\pi}(s_t)$ when fit function approximator
- Find weights to minimize mean squared error

$$J(\boldsymbol{w}) = E_{\pi}[G_{\mathrm{t}} - \hat{V}(s; \boldsymbol{w}))^2]$$

- Using linear VFA for policy evaluation, gradient computing

$$\begin{aligned}\Delta \boldsymbol{w} &= \alpha * \left(\left(V^{\pi}(\boldsymbol{s}) - \hat{V}^{\pi}(s_t; \boldsymbol{w}) \right) * \nabla_w \hat{V}^{\pi}(s_t; \boldsymbol{w}) \right) \\ &= \alpha * \left(\left(G_t - \hat{V}^{\pi}(s_t; \boldsymbol{w}) \right) * \nabla_w \hat{V}^{\pi}(s_t; \boldsymbol{w}) \right) \\ &= \alpha * ((G_t - \boldsymbol{f}(s_t)^T \boldsymbol{w}) * \boldsymbol{f}(s_t))\end{aligned}$$

MC Linear VFA for Policy Evaluation

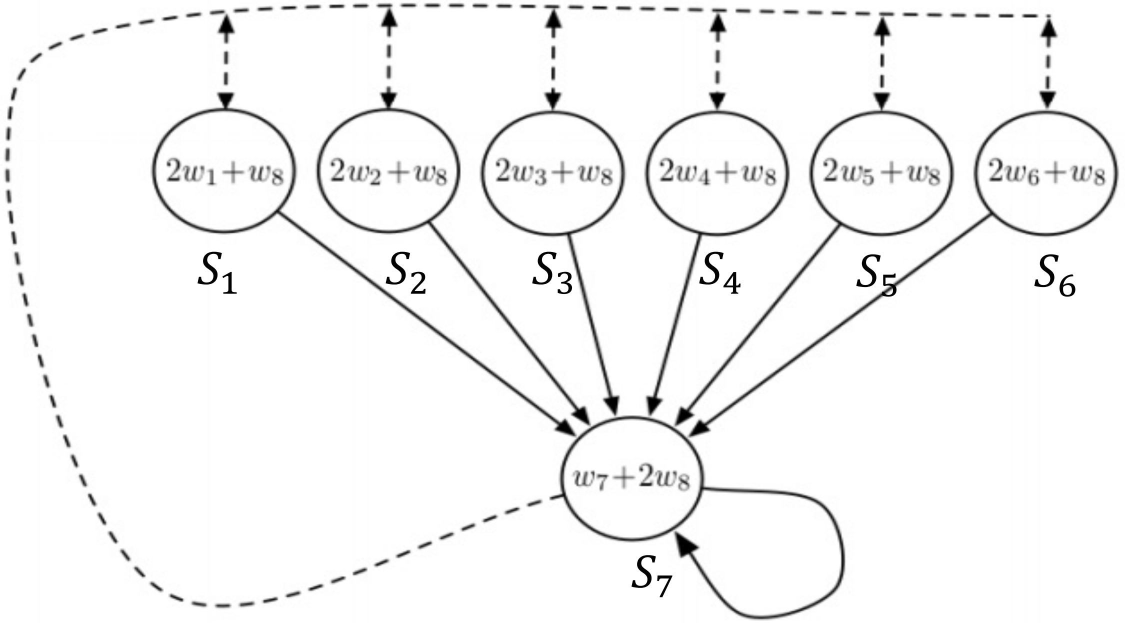
1: Initialize $\mathbf{w} = \mathbf{0}$, $k = 1$
2: **loop**
3: Sample k-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,L_k})$ given π
4: **for** $t = 1, \ldots, L_k$ **do**
5: **if** First visit to (s) in episode k **then**
6: $G_t(s) = \sum_{j=t}^{L_k} r_{k,j}$
7: Update weights:

$$\boldsymbol{w} = \boldsymbol{w} + \alpha * \left(\left(G_t - \boldsymbol{f}(s)^T \boldsymbol{w}\right) * \boldsymbol{f}(s)\right)$$

8: **end if**
9: **end for**
10: $k = k + 1$
11: **end loop**

Example for MC Policy Evaluation

- 7 states, 8 features
- 2 actions
  - a_1 ->: deterministic
  - a_2 -->: random walk to other states (1/6)
  - Terminate in S_7 for some probability
- no reward
- Trajectory:
  - τ: $S_1, a_1, 0, S_7, a_1, 0, S_7, a_1, 0, terminate$
- Suppose $w_0 = [1,1,1,1,1,1,1,1], \alpha = 0.5, \gamma = 0.9$
- Demonstrate the update of $V^\pi(S)$, following τ



S_1: [2,0,0,0,0,0,0,1]
S_2: [0,2,0,0,0,0,0,1]
S_3: [0,0,2,0,0,0,0,1]
S_4: [0,0,0,2,0,0,0,1]
S_5: [0,0,0,0,2,0,0,1]
S_6: [0,0,0,0,0,2,0,1]
S_7: [0,0,0,0,0,0,1,2]

Feature Vectors

$$MC\ updates: \Delta w = \alpha\left(G_t(s_t) - x(s_t)^T w\right) * x(s_t)$$

TD Value Function Approximation

- In value function approximation update is based on (s_t, a_t, r_t, s_{t+1})

$$r + \gamma \hat{V}^{\pi}(s', w)$$

- Do supervised learning on a set of (state, next state value) pairs

$$\left(s_1, r_1 + \gamma \hat{V}^{\pi}(s_2, \boldsymbol{w})\right), \left(s_2, r_2 + \gamma \hat{V}^{\pi}(s_3, \boldsymbol{w})\right), \left(s_3, r_3 + \gamma \hat{V}^{\pi}(s_4, \boldsymbol{w})\right), \ldots, \left(s_4, r_4 + \gamma \hat{V}^{\pi}(s_5, \boldsymbol{w})\right)$$

- Find weights to minimize mean squared error

$$J(\boldsymbol{w}) = E_{\pi}[(r + \gamma \hat{V}^{\pi}(s_{t+1}, \boldsymbol{w}) \; - \hat{V}^{\pi}(s_t; \boldsymbol{w}))^2]$$

- Gradient:

$$\Delta \boldsymbol{w} = \alpha * \left(\left(V^{\pi}(\boldsymbol{s}) - \hat{V}^{\pi}(s_t; \boldsymbol{w})\right) * \nabla_w \hat{V}^{\pi}(s_t; \boldsymbol{w})\right)$$

$$= \alpha * \left(\left(r_{\mathrm{t}} + \gamma \hat{V}^{\pi}(s_{t+1}, \boldsymbol{w}) - \hat{V}^{\pi}(s_t; \boldsymbol{w})\right) * \nabla_w \hat{V}^{\pi}(s_t; \boldsymbol{w})\right)$$

$$= \alpha * (\left(r_{\mathrm{t}} + \gamma \hat{V}^{\pi}(s_{t+1}, \boldsymbol{w}) - \boldsymbol{f}(s_t)^T \boldsymbol{w}\right) * \boldsymbol{f}(s_t))$$

TD Value Function Approximation

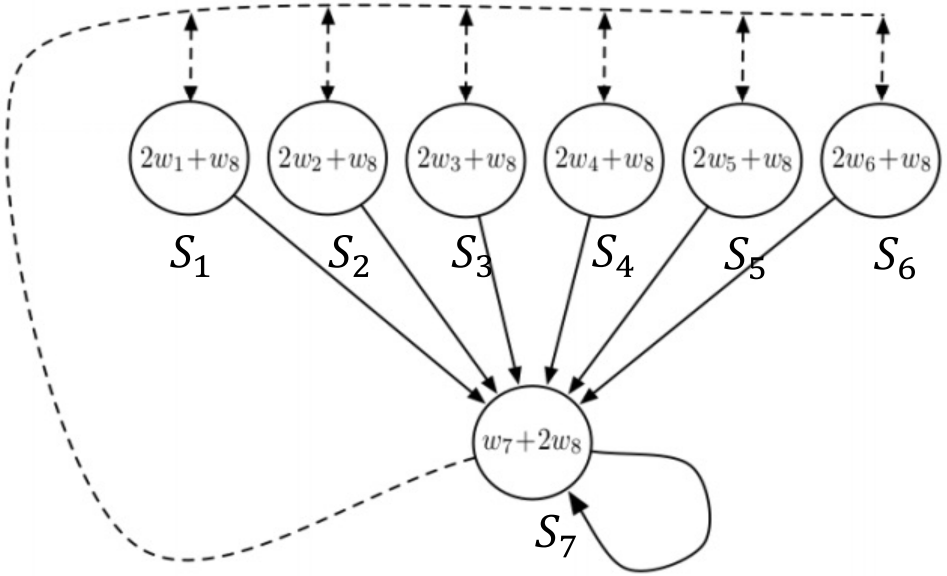
Initialize $\mathbf{w} = \mathbf{0}$, $k = 1$
loop
    Sample tuple (s_k, a_k, r_k, s_{k+1}) given π
    Update weights:

$$\boldsymbol{w} = \boldsymbol{w} + \alpha * \left(\left(r + \gamma \widehat{V}^{\pi}(s_{t+1}, \boldsymbol{w}) - \boldsymbol{f}(s_t)^T \boldsymbol{w}\right) * \boldsymbol{f}(s_t)\right)$$

    $k = k + 1$
end loop

Example for TD Policy Evaluation

- 7 states, 8 features
- 2 actions
  - a_1 ->: deterministic
  - a_2 -->: random walk to other states (1/6)
- no reward
- Transition: $(S_1, a_1, 0, S_7)$
- Suppose $w_0 = [1,1,1,1,1,1,1,1]$, $\alpha = 0.5, \gamma = 0.9$
- Demonstrate the update of $V^\pi(S)$



S_1: [2,0,0,0,0,0,0,1]
S_2: [0,2,0,0,0,0,0,1]
S_3: [0,0,2,0,0,0,0,1]
S_4: [0,0,0,2,0,0,0,1]
S_5: [0,0,0,0,2,0,0,1]
S_6: [0,0,0,0,0,2,0,1]
S_7: [0,0,0,0,0,0,1,2]

Feature Vectors

$$TD\ updates: \Delta w = \alpha\left(r + \gamma * x(s_{t+1})^T w - x(s_t\ )^T w\right) * x(s_t)$$

Outline

- States Generalization
- Function Approximation Preliminaries
- Value Function Approximation (VFA) for Policy Evaluation
- MC and TD for VFA in Policy Evaluation
- Value Function Approximation (VFA) for Policy Control

Policy Control using Value Function Approximation

- Use value function approximation to represent state-action values

$$\widehat{\mathrm{Q}}^{\pi}(s,a;\boldsymbol{w}) \approx Q^{\pi}(s,a)$$

- Policy Iteration
  - Policy evaluation： using value function approximation (MC or TD)
  - Policy Improvement: $\varepsilon - greedy$, same as non-VFA setting

Linear State Action Value Function Approximation

- Use features to represent both the state and action

$$\boldsymbol{f}\,(s,a) = (f_1(s,a), f_2(s,a), f_n(s,a))$$

- Represent state-action value function with a weighted linear combination of features

$$\widehat{\mathrm{Q}}^{\pi}(s,a;w) = \boldsymbol{f}(s,a)^T\boldsymbol{w} = \sum_{j=1}^{n} f_j(s,a)w_j$$

- Objective function:

$$J(w) = E_{\pi}[(Q^{\pi}(s,a) - \widehat{\mathrm{Q}}^{\pi}(s,a;\boldsymbol{w}))^2]$$

- Stochastic gradient descent update

$$\Delta\boldsymbol{w} = \alpha * ((\mathrm{Q}^{\boldsymbol{\pi}}(\boldsymbol{s},\boldsymbol{a}) - \boldsymbol{f}(s,a)^T\boldsymbol{w}) * \boldsymbol{f}(s,a))$$

Incremental Model-Free Control Approaches

- Similar to policy evaluation, true state-action value function is unknown and need to substitute a target value
- In Monte Carlo methods, use a return G_t as substitute target

$$\Delta\boldsymbol{w} = \alpha * \left(\left(G_t - \hat{Q}^{\pi}(s_t, a_t; \boldsymbol{w})\right) * \nabla_w \hat{Q}^{\pi}(s_t, a_t; \boldsymbol{w})\right)$$

- For SARSA, use a TD target $r + \gamma\hat{Q}(s', a'; \boldsymbol{w})$ which leverages the current function approximation value

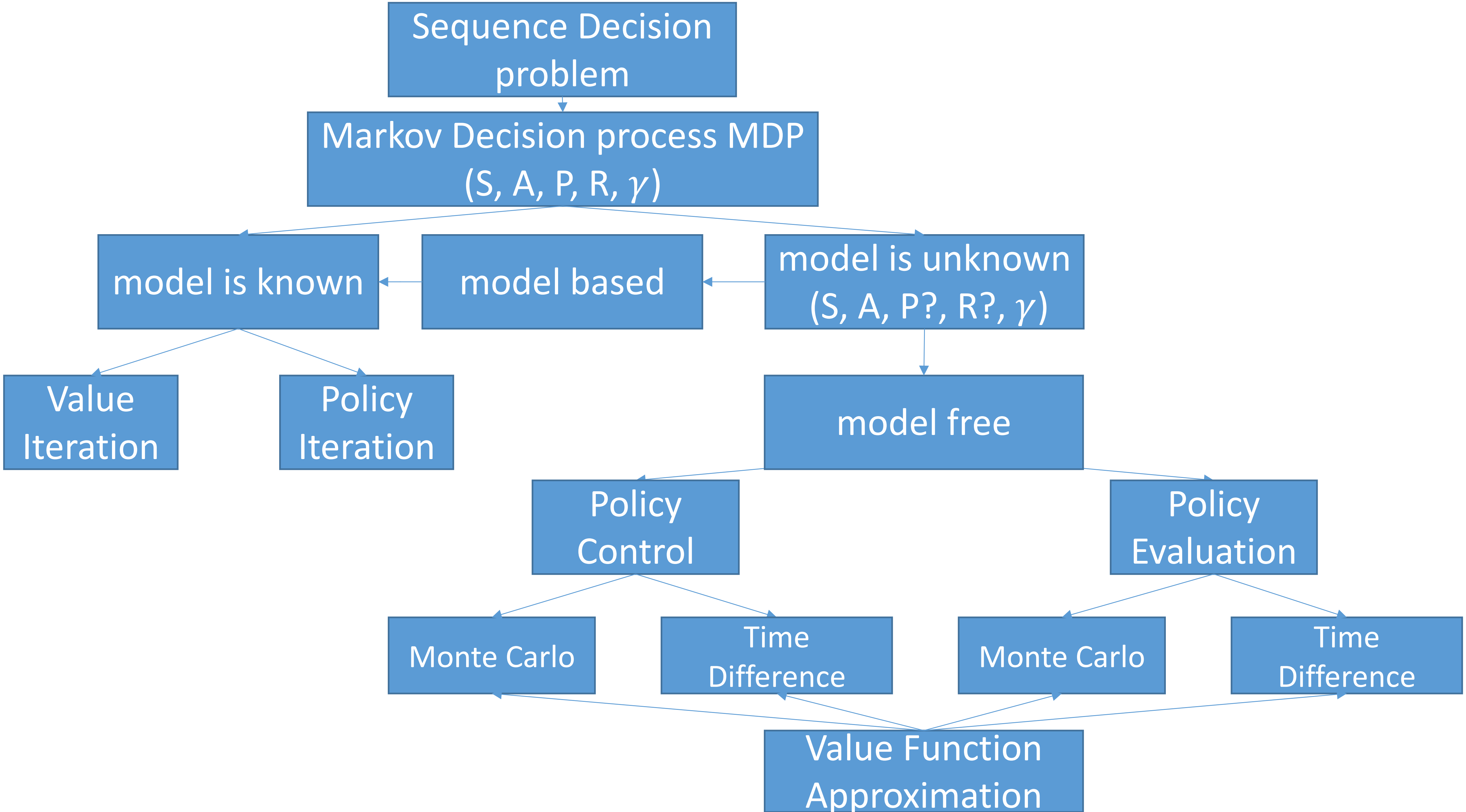
$$\Delta\boldsymbol{w} = \alpha * \left(\left(r + \gamma\hat{Q}^{\pi}(s_{t+1}, a_{t+1}; \boldsymbol{w}) - \hat{Q}^{\pi}(s_t, a_t; \boldsymbol{w})\right) * \nabla_w \hat{Q}^{\pi}(s_t, a_t; \boldsymbol{w})\right)$$

- For Q-learning, use a TD target $r + \gamma \max_a Q^{\pi}(s_{t+1}, a_{t+1}; \boldsymbol{w})$ which leverages the max of the current function approximation value

$$\Delta\boldsymbol{w} = \alpha * \left(\left(r + \gamma \max_a Q^{\pi}(s_{t+1}, a_{t+1}; \boldsymbol{w}) - \hat{Q}^{\pi}(s_t, a_t; \boldsymbol{w})\right) * \nabla_w \hat{Q}^{\pi}(s_t, a_t; \boldsymbol{w})\right)$$

Summary of Reinforcement Learning

強化學習大致框架



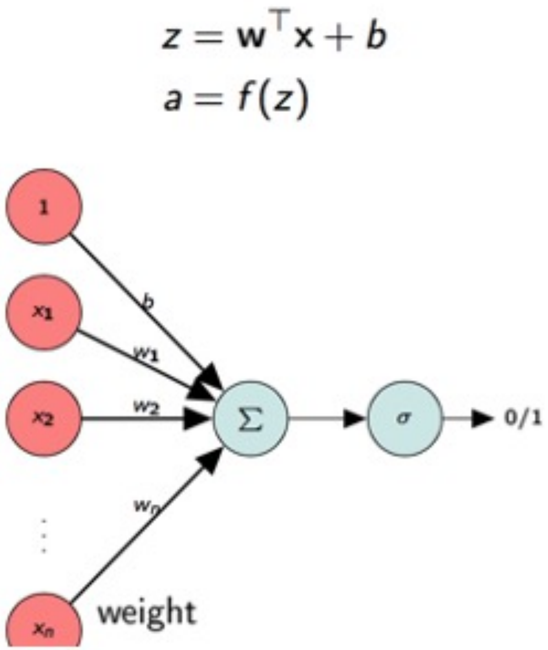
Outline

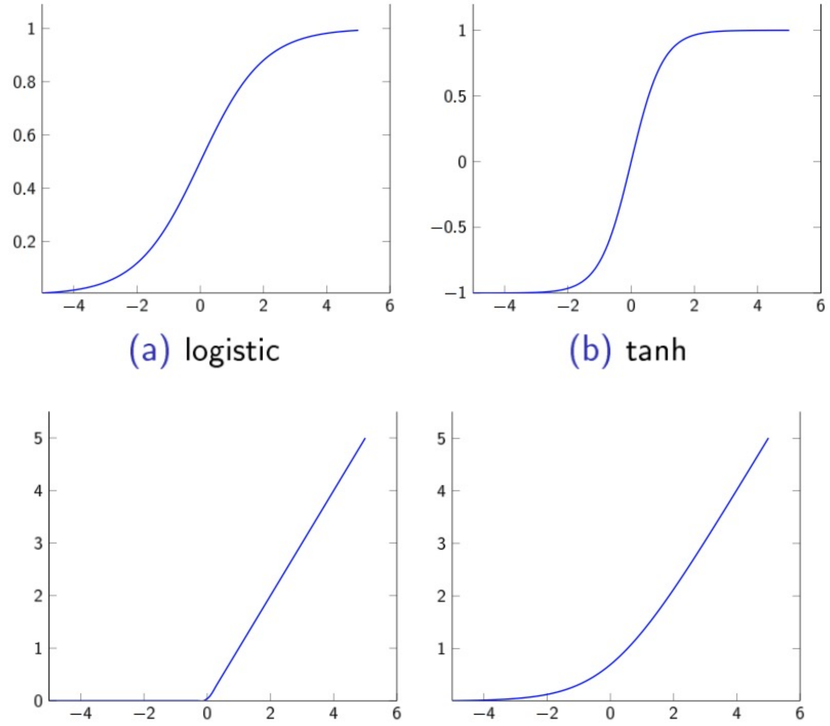
- States Generalization
- Function Approximation Preliminaries
- Value Function Approximation (VFA) for Policy Evaluation
- MC and TD for VFA in Policy Evaluation
- Value Function Approximation (VFA) for Policy Control
- Deep Q-Learning Network

Artificial Neuron

- An artificial neuron is a mathematical function to simulate biological neurons. Input: $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$

- State: **z**
- Output: **a**
- Activation Function: **f**



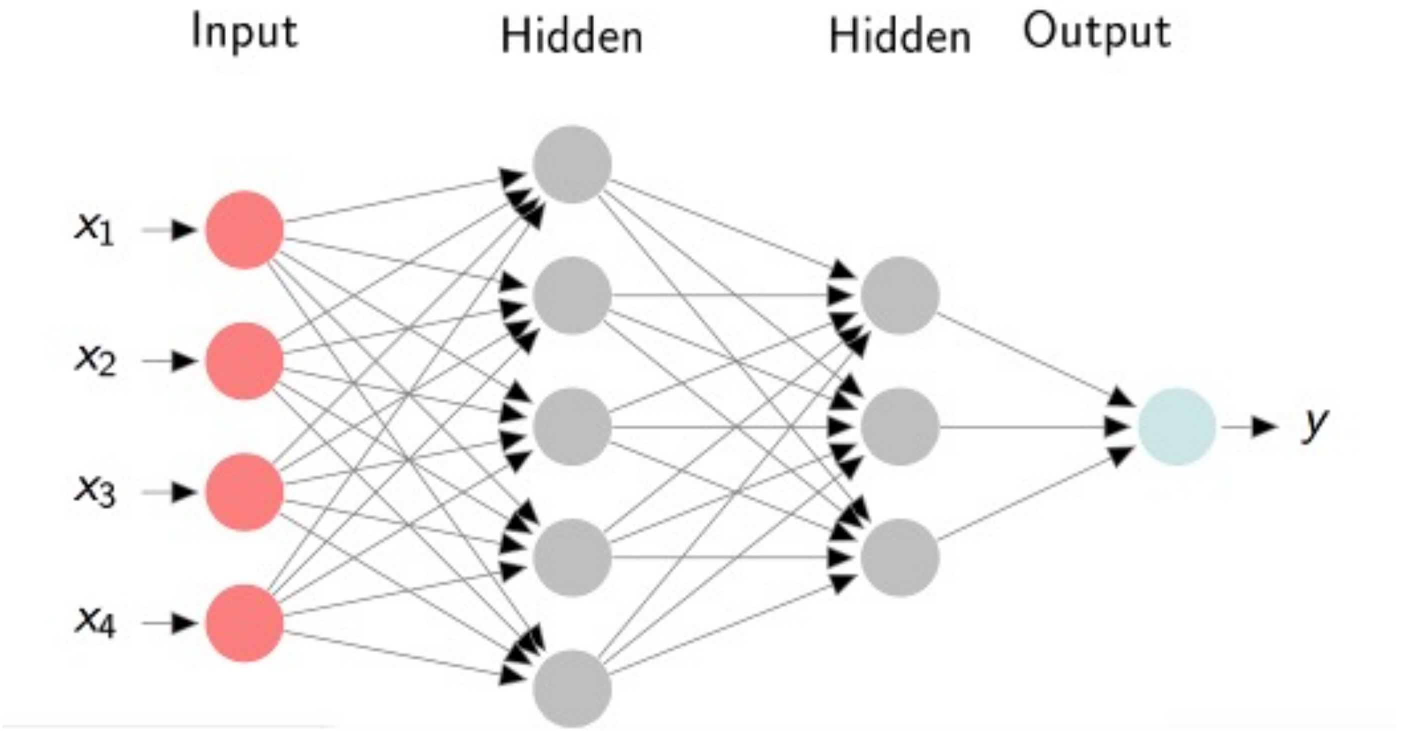


(a) logistic (b) tanh

(c) rectifier (d) softplus

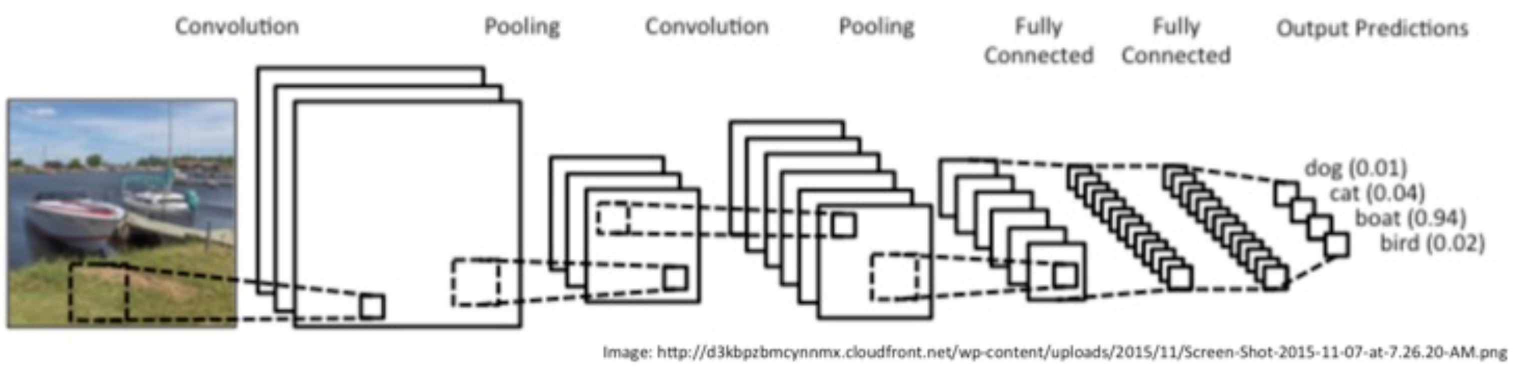
Feedforward Neural Network

- In feedforward neural network, the information moves in only one direction forward: from the input nodes data goes through the hidden nodes and to the output nodes.



- Composition of multiple functions
- Can use a chain rule to backpropagate the gradient

Convolutional Neural Network



捲機神經網路

transformer神經
網路將替代掉這個

- Consider local structure and common extraction of features
- Not fully connected
- Locality of processing
- Weight sharing for parameter reduction
- Learn the parameters of multiple convolutional filter banks
- Compress to extract salient features & favor generalization

The Benefit of Deep Neural Network Approximators

- Linear value function approximators assume value function is a weighted combination of a set of features, where each feature is a function of the state.

- Linear VFA often work well given the right set of features
- But can require carefully hand designing that feature set

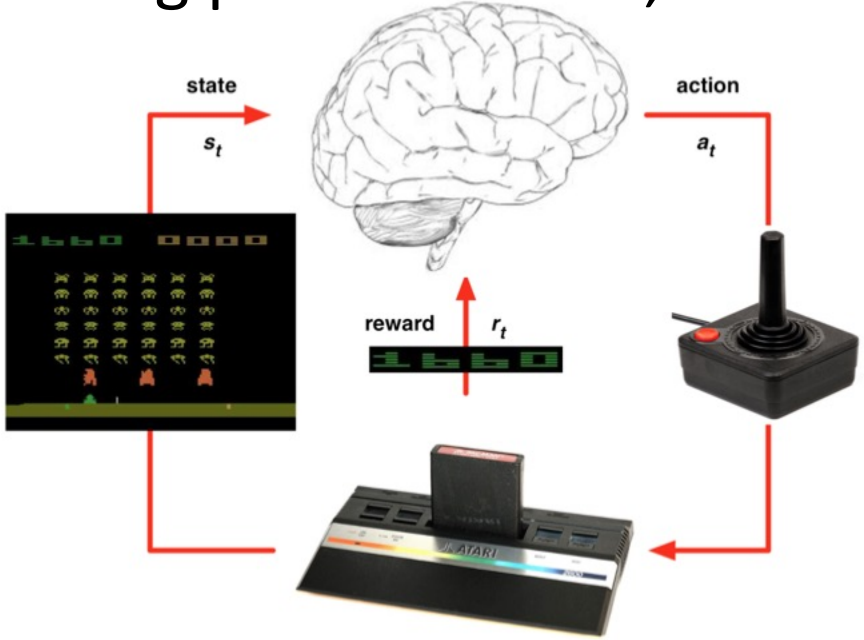
- Alternative: Deep neural networks
  - Uses distributed representations instead of local representations
  - Universal function approximator
  - Can potentially need exponentially less nodes/parameters, (compared to a shallow net) to represent the same function
  - Can learn the parameters using stochastic gradient descent

Using DQN to do RL in Atari 很多小遊戲的平台

用強化學習的方法建構起來

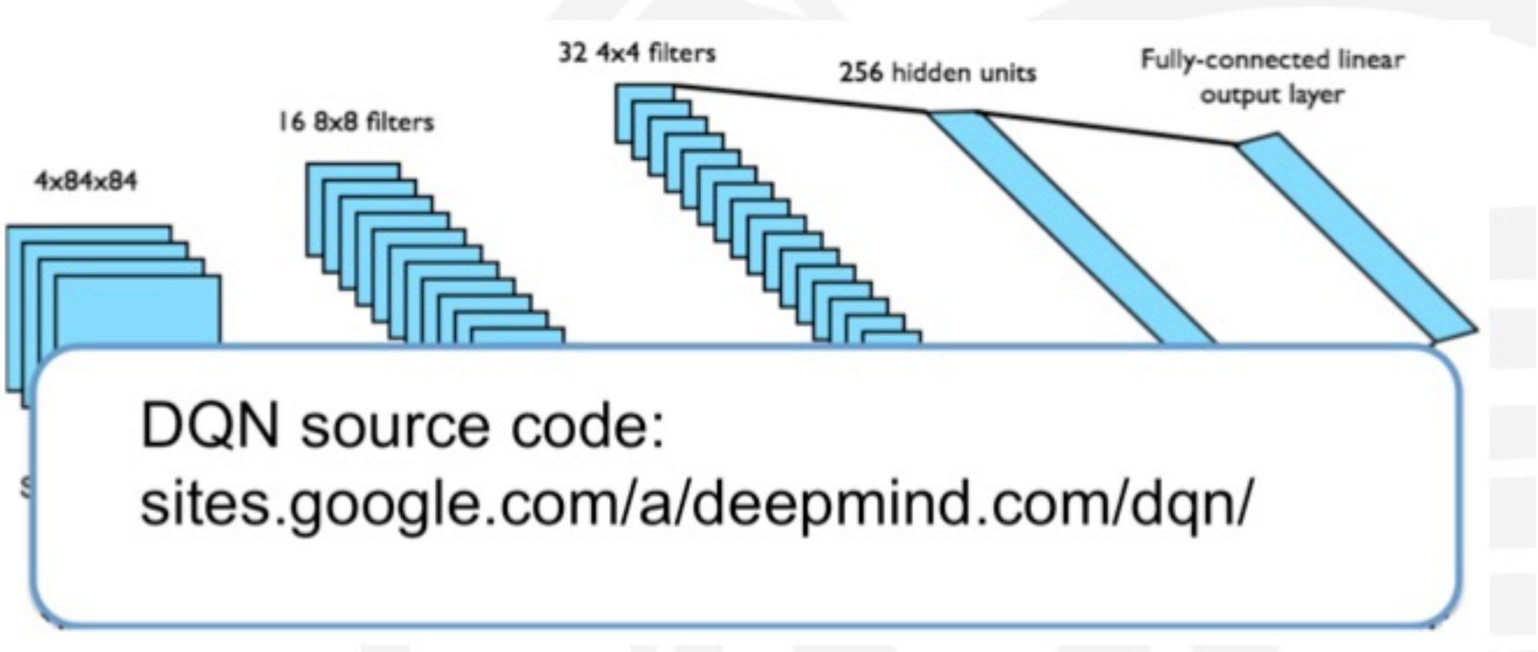
- **Atari is one the most recognized and celebrated brands in the world.**

- Founded in 1972, Atari played an integral role in the development of the arcade game, game console and personal computer industries. Atari's iconic games, including Pong®, Asteroids®, Centipede® Missile Command®, have been played by many millions, and the brand continues to bring joy to gamers with its expanding portfolio of PC, console and mobile games.



DQNs in Atari

- End to end learning of values Q(s, a) from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is Q(s, a) for 18 joystick/button positions
- Reward is change in score for that step



- Network architecture and hyperparameters fixed across all games

Outline

- States Generalization
- Function Approximation Preliminaries
- Value Function Approximation (VFA) for Policy Evaluation
- MC and TD for VFA in Policy Evaluation
- Value Function Approximation (VFA) for Policy Control
- Deep Q-Learning Network
- Issues in Value Function Approximation

Issues in Value Function Approximation

- Q-learning converges to the optimal $Q*(s,a)$ using table lookup representation
- In value function approximation Q-learning we can minimize MSE loss by stochastic gradient descent using a target Q estimate instead of true Q (as we saw with linear VFA)

- But Q-learning with VFA can diverge and two issues causing problems:
  - Correlations between samples
  - Non-stationary targets

- Deep Q-learning (DQN)
  - Experience replay
  - Fixed Q-targets

DQNs: Experience Replay

- To help remove correlations, store dataset D from prior experience

s_1, a_1, r_2, s_1
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
…
$s_t, a_t, r_{t+1}, s_{t+1}$

$\longrightarrow \quad s, a, r, s'$

- To perform experience replay, repeat the following:
    - (s,a,r,s') : sample an experience tuple from the dataset D
    - Compute the sampled target value $s : r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$
    - Use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha(r + \gamma max_{a'} \hat{Q}(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}))\nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w})$$

- Can treat the target as a scalar, but the weights will get updated on the next round, changing the target value

DQNs: Fixed Q-Targets

- To help improve stability, fix the target weights used in the target calculation for multiple updates
- Target network uses a different set of weights than the weights being updated
- Let parameters $\boldsymbol{w}^{-}$ be the set of weights used in the target, and $\boldsymbol{w}$ be the weights that are being updated
- Slight change to computation of target value:
  - (s,a,r,s′) : sample an experience tuple from the dataset
  - Compute the sampled target value s : $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^{-})$
  - Use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha(r + \gamma max_{a'} \hat{Q}(s', a'; \mathbf{w}^{-}) - \hat{Q}(s, a; \mathbf{w}))\nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w})$$

DQNs Pseudocode

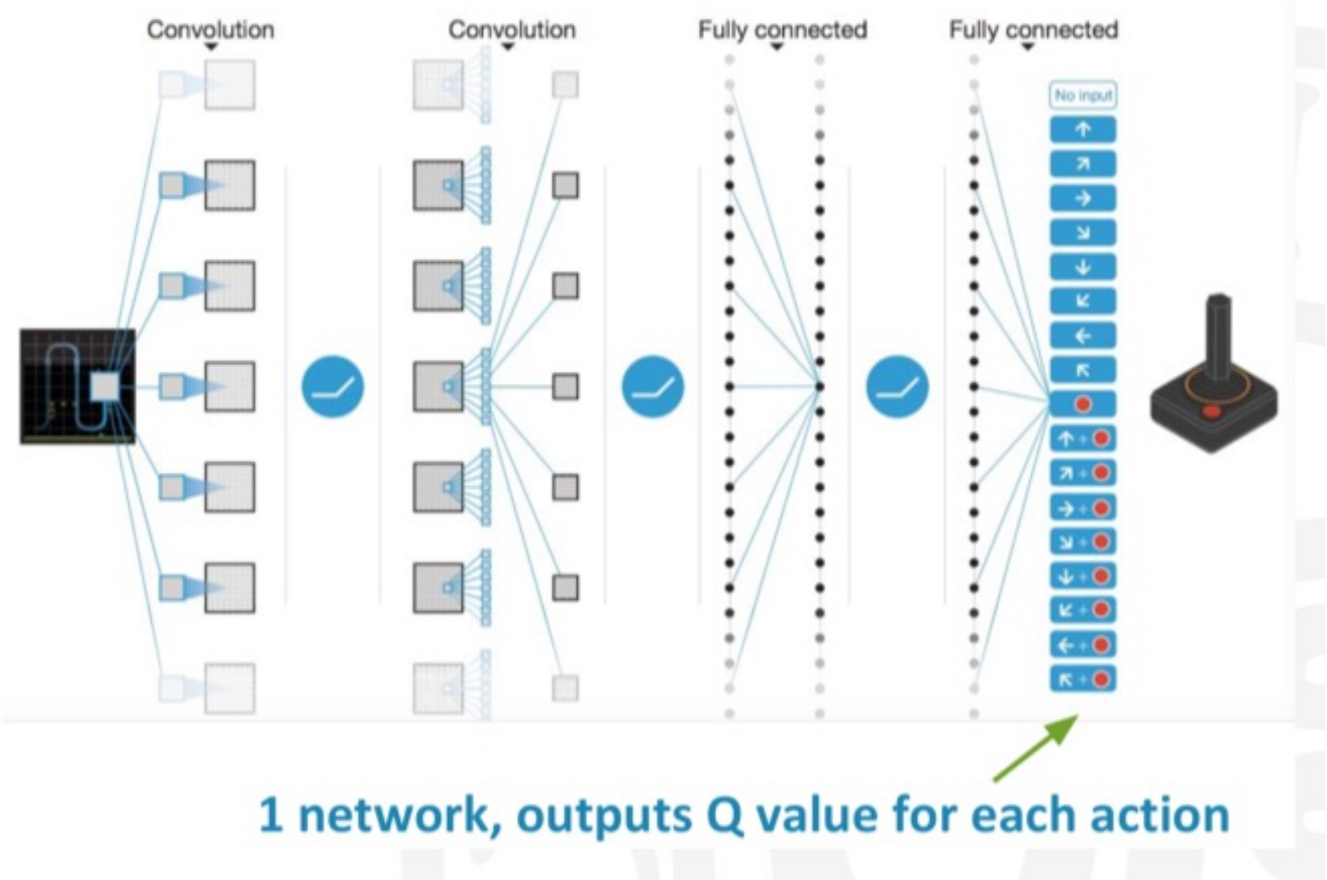
Input C, α, $D = \{\}$, Initialize $\boldsymbol{w}$, $\boldsymbol{w}^- = \boldsymbol{w}$, $t = 0$
Get initial state s_0
loop
    Sample action a_t given ϵ-greedy policy for current $\hat{Q}(s_t, a; \boldsymbol{w})$
    Observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer D
    Sample random minibatch of tuples (s_i, a_i, r_i, s_{i+1}) from D
    for j in minibatch **do**
        if episode terminated at step $i+1$ **then**
            $y_i = r_i$
        else
            $y_i = r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; \boldsymbol{w}^-)$
        end if
        Do gradient descent step on $(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))^2$ for parameters $\boldsymbol{w}$: $\Delta \boldsymbol{w} = \alpha(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_i, a_i; \boldsymbol{w})$
    end for
    $t = t + 1$
    if mod(t,C) == 0 **then**
        $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$
    end if
end loop

- One needs to choose the **neural network architecture**, the **learning rate**, and **how often to update the target network**.
- Often a fixed **size replay buffer** is used for experience replay, which introduces a parameter to control the size, and the need to decide how to populate it.

DQNs Summary

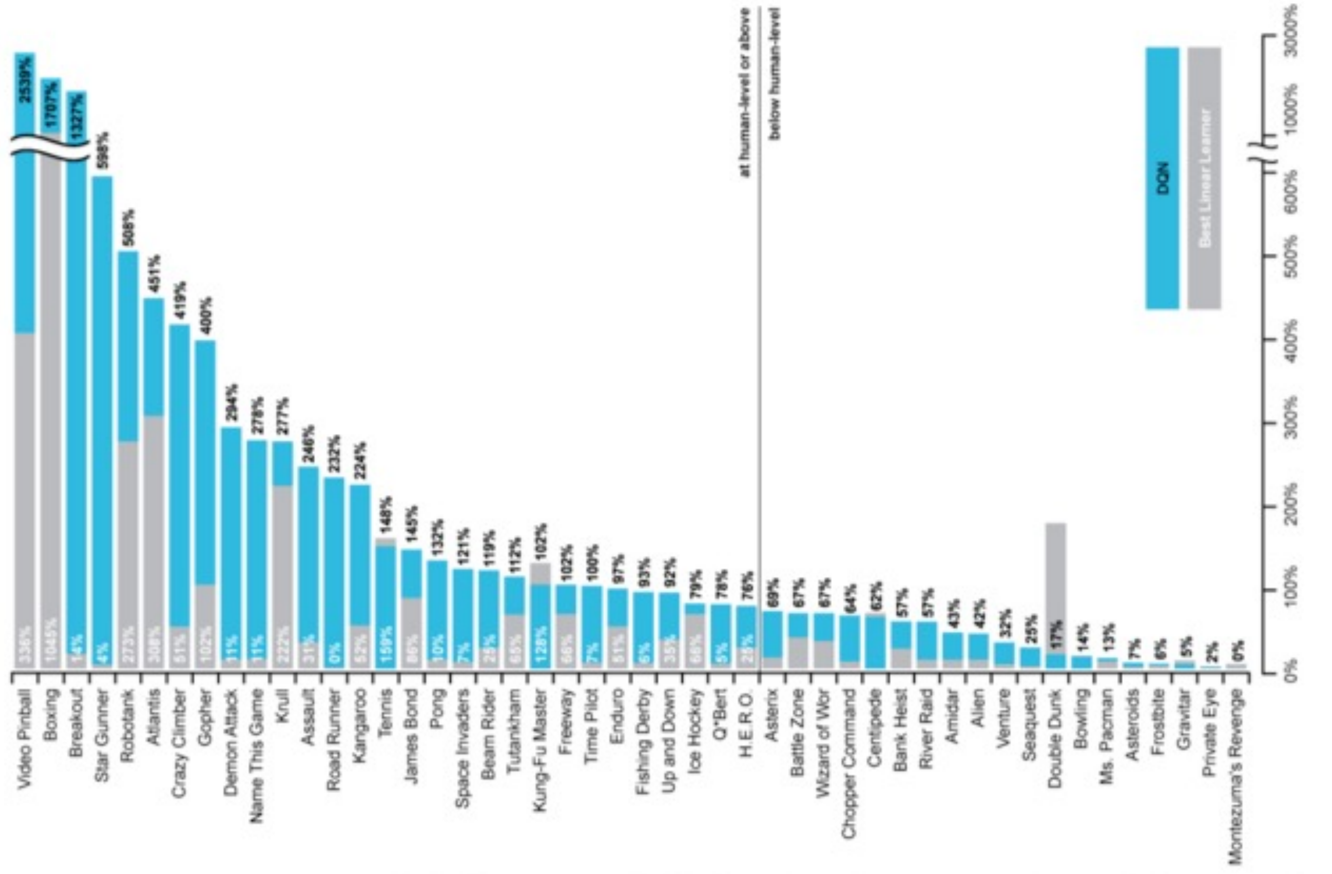
- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- Sample random mini-batch of transitions (s, a, r, s') from D
- Compute Q-learning targets w.r.t. old, fixed parameters w
- Optimizes MSE between Q-network and Q-learning targets
- Uses stochastic gradient descent

DQN



- Figure 1: Human-level control through deep reinforcement learning, Mnih et al, 2015

DQN Results in Atari



- Figure 2: Human-level control through deep reinforcement learning, Mnih et al, 2015

Which Aspects of DQN were Important for Success?

強化學習:神經網路+DQN alpha go

加上經驗回放

Game	Linear	Deep Network	DQN w/fixed Q	DQN w/reply	DQN w/ both
Breakout	3	3	10	241	317
Enduro	62	29	141	831	1006
River Raid	2345	1453	2868	4102	7447
Seaquest	656	275	1003	823	2894
Space Invaders	301	302	373	826	1089

- Just using a deep NN actually hurts performance sometimes
- Replay is hugely important
- Why? Beyond helping with correlation between samples, what does replaying do?

More about experience replay

Experience Replay Does this	T or F
Involves using a bank of prior (s,a,r,s) tuples and doing Q-learning updates on the tuples in the bank.	T
Always uses ~~the most recent~~ history of tuples 循環	F
~~Reduces~~ the data efficiency of DQN	F
Increases the computational cost	T

More Readings

- Reinforcement Learning: An introduction, Richard S. Sutton and Andrew G. Barto
- CS234, Reinforcement Learning
- 《神经网络与深度学习》， 邱锡鹏， Chapter 14

CS188