CoE202
Fundamentals of Artificial intelligence
<Big Data Analysis and Machine Learning>

Reinforcement learning

Prof. Young-Gyu Yoon
School of EE, KAIST

KAIST EE
Neuro-Instrumentation & Computational Analysis Lab

Contents

- Recap
  - Sequence prediction problem
  - Recurrent neural network
  - Training RNN
  - Advanced unit cells
- Overview of reinforcement learning (Maze example)
- Mathematical backgrounds
  - Random variable, random process
  - Markov process
  - Markov reward process, Markov decision process
  - Return and value function
- Optimal policy

Notations lookup (1)

$$P(X = 3) = 1/6$$

For a random variable X, the probability that its realization equals 3 is 1/6

$$P(S_t = s_t | S_{t-1} = s_{t-1})$$

State transition probability from s_{t-1} to s_t : the probability that a random process S_t at time t equals s_t given that it equals s_{t-1} at time t -1

$$p_{21} = P(S_t = 2 | S_{t-1} = 1)$$

State transition probability from 1 to 2: the probability that a random process S_t at time t equals 2 given that it equals 1 at time t -1

$$R(s) = E[r_t | S_t = s]$$

Reward function (of a state s) is defined as the expected value of immediate reward given that the current state equals s.

Notations lookup (2)

$$(S_t, A_t, R_t)$$

A tuple of S_t, A_t, R_t (state, action, reward). In mathematics, a tuple is a finite ordered list of elements.

$$\pi(a|s) = P(A_t = a|S_t = s)$$

Policy is defined as the probability to choose action as a, given the state s.

$$Q_\pi(s, a; \theta)$$

Value of state-action pair (s,a) for an MDP that follows a policy π that is parameterized by θ.

$$\pi^*(s) = \arg\max_\pi V^\pi(s)$$

The optimal policy is a policy that maximizes the value of a given sate.

$$Q_{\pi*}(s, a)$$

Value of state-action pair (s,a) for an MDP that follows the <u>optimal</u> policy $\pi*$

Types of machine learning

- **Supervised learning**: learning a function that maps an input to an output based on example input-output pairs

- **Unsupervised learning**: looking for previously undetected patterns in a data set with no pre-existing labels and without human supervision

- **Reinforcement learning**: enabling an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences

Types of machine learning

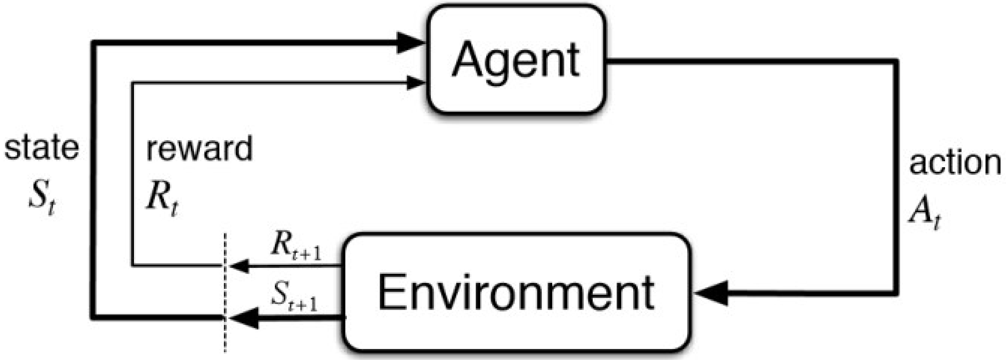
- **Supervised learning**: “I’ll give you some pairs of questions and answers. Learn from these pairs to be able to answer to other questions.”
  - Regression
  - Classification

- **Unsupervised learning**: “I’ll give you some unlabeled data. Try to find if there’s any interesting structure or pattern in the data.”
  - Clustering
  - Dimension reduction

- **Reinforcement learning**: “I cannot teach you what to do, but I can give scores to what you did. Based on the scores you got from what you did, learn what to do.”

Reinforcement learning

- The goal of reinforcement learning is to derive an agent that takes actions in an environment that maximize the cumulative reward



$$(S_0, A_0, R_0), (S_1, A_1, R_1), (S_2, A_2, R_2), \cdots$$

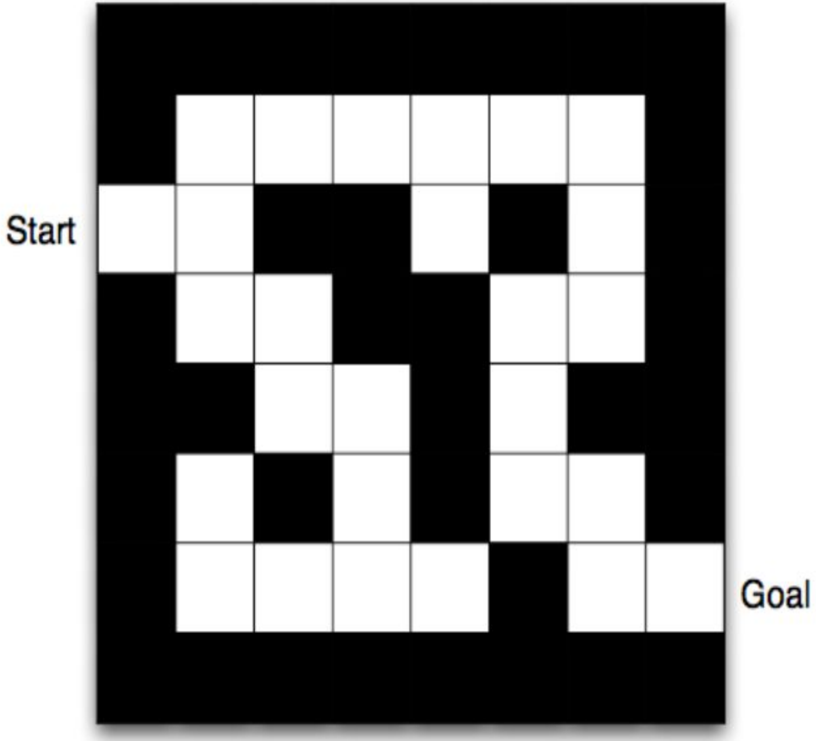
Reinforcement learning

- **Agent** (game player): decides what action to make
- **Environment** (game): provide state to the agent, then takes actions from the agent, then provide next state and reward to the agent
  - The environment is typically modeled as Markov decision process (MDP) → we will talk about this later
- **Reward** (game score): is given to the agent (from the environment) and serves as a criterion for good/bad actions
  - Note that we want to maximize the cumulative reward, not the immediate reward
  - If we only care about the immediate reward, it would be basically the same as supervised learning
  - This delay is what makes RL interesting and difficult

Difficulty in reinforcement learning

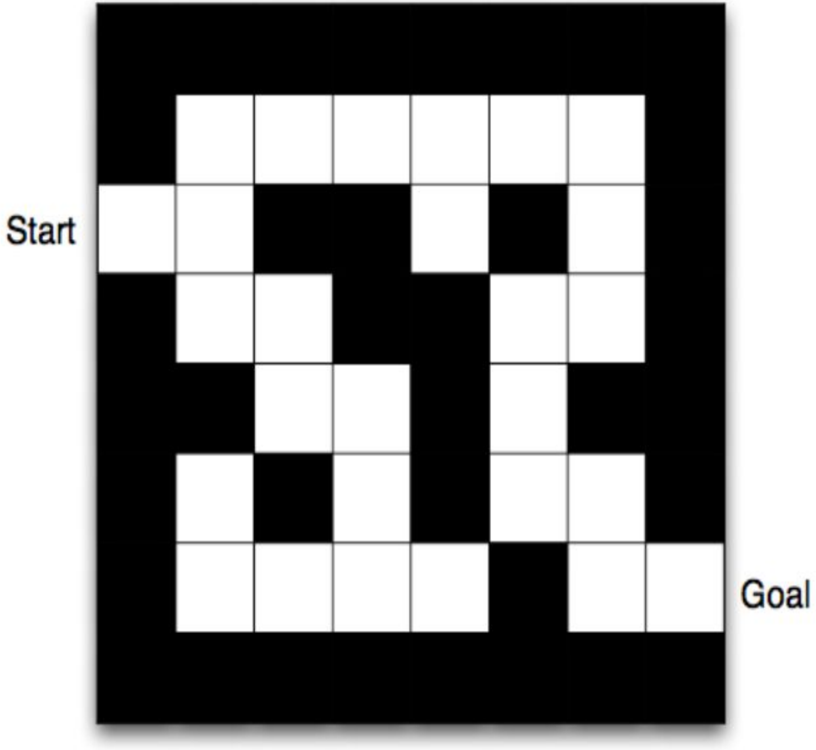
- **Credit assignment problem**
  - If a “good” action always results in an immediate reward, learning would be very easy
    - In fact, this will be basically the same as supervised learning problem
  - However, in most reinforcement learning problems, there is a non-deterministic time delay
  - Moreover, what often determines the reward is a sequence of actions – not just one action at one time point
  - In other words, we do not know which action was responsible for a certain reward
  - Then, how can we tell which was good and which was bad - and how can we reinforce good actions?
- Basically, we need a framework to connect the rewards to the “past” actions
  - Information has to flow backwards (in time)

Reinforcement learning: maze example



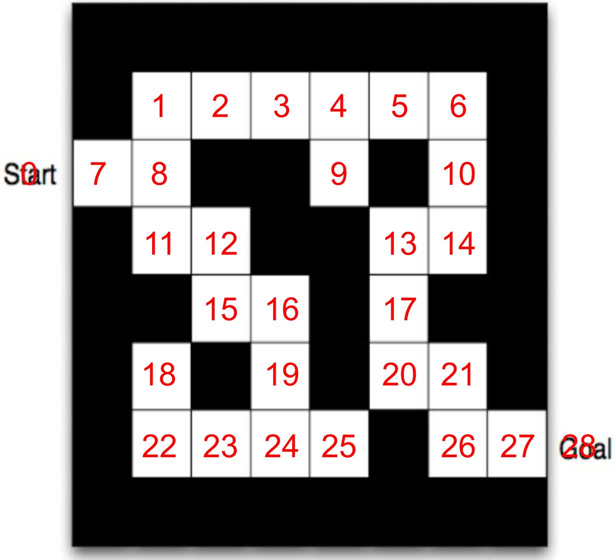
- **Goal**
  - We want to train an agent so that it can escape from the maze as soon as possible
- **Setting**
  - We are not going to "supervise" the agent in terms of what move it should make at each time point
  - We will just give "scores" to the agent, and the agent is supposed to learn from the scores

Reinforcement learning: maze example



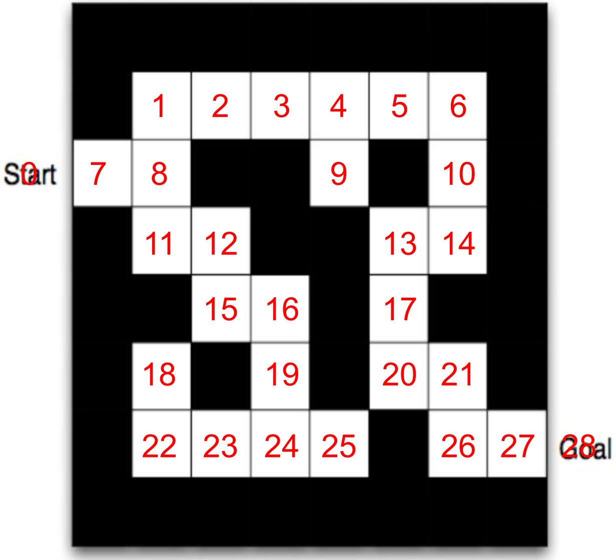
- **State:** agent's current location (or the image itself which shows where the agent is)
- **Action**: up, down, left, right
- **Reward**: -(time taken)

Reinforcement learning: maze example



- **Formulation**
  - **State:**
    an integer number between 0 and 28
  - **Action:**
    an integer number between 0 and 3
    (0:up, 1: down, 2: left, 3: right)
  - **Reward**:
    -1 at each time step
  - **The "agent"**
    - a function f that takes current state as the input and calculates the best action

Reinforcement learning: maze example



- **Sequence of events and decisions**
  - **State 0 → f → Action 3 → Reward -1, State 7**
  - **State 7 → f → Action 3 → Reward -1, State 8**
  - **State 8 → f → Action 0 → Reward -1 , State 1**

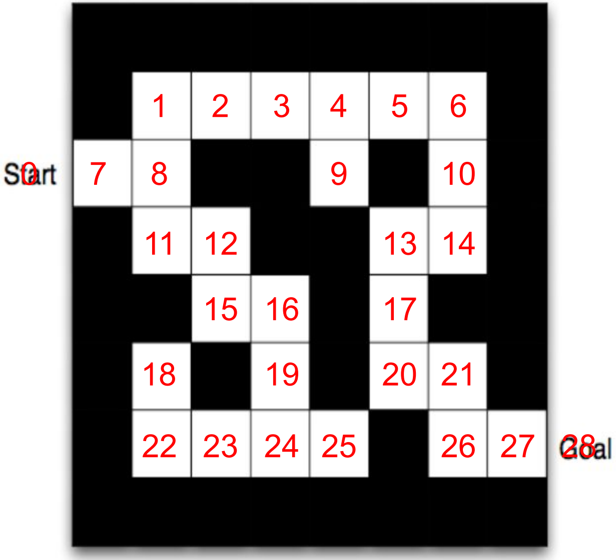
Current state

Action determined by the agent based on the current state

New state

0:up, 1: down, 2: left, 3: right

Reinforcement learning: maze example



- Information flow (backward in time)
  - Through multiple experiences, we can learn that state 27 is a "good" state → we can assign high value to this state.
  - Moreover, we can learn that moving right at state 27 is a good combination.
    → we can assign high value to this state action pair (s=27|a=right)
  - After that, we can learn state 26 can easily lead to state 27, which means state 26 is also good → we can assign high value to this state.
  - Again, we can learn that moving right at state 26 is a good combination.
    → we can assign high value to this state action pair (s=26|a=right)
  - …we can repeat this process
- By repeating this process, we can learn which state is good and which state/action pair is good

Random variable, process, notation

- **Random variable:** a variable whose values depend on outcomes of a random phenomenon

  If X is a random variable from rolling a dice, $P(X = 3) = 1/6$

- **Random process:** a time series of a random variable

  e.g., accumulated sum of dice rolling

$$P(X_t = X_{t-1} + 3) = 1/6$$

- **Upper case** letters such as X or Y denote a **random variable. Lower case** letters like x or y denote the value of a **random variable**

Simply put,
Upper: random variable (not a number)
Lower: a real number (possible outcome)

$$P(X = x) = 1/6$$

1 or 2 or 3 or 4 or 5 or 6

Markov process: weather transition example

- **Markov process:** a memoryless random process* whose future probabilities are determined by its most recent value

$$P(S_t = s_n | S_{t-1} = s_{t-1}, \cdots, S_0 = s_0) = P(S_t = s_t | S_{t-1} = s_{t-1})$$

  - "The future is independent of the past given the present"

- **State transition probability matrix**
  - **Let's assume that there are 3 possible states**

$$\begin{bmatrix} P(S_t = 0) \\ P(S_t = 1) \\ P(S_t = 2) \end{bmatrix} = \begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \end{bmatrix} \begin{bmatrix} P(S_{t-1} = 0) \\ P(S_{t-1} = 1) \\ P(S_{t-1} = 2) \end{bmatrix}$$

Probability of next state being 2 given the current state 1

$$p_{21} = P(S_t = 2 | S_{t-1} = 1)$$

Probability of next state being 2 given the current state 2

*formal definition of random process is somewhat complicated…for now, let's just consider it as a random number that changes over time

Markov process: weather transition example

- If day-by-day weather change is a Markov process*

$$\begin{bmatrix} P(S_t = s) \\ P(S_t = r) \\ P(S_t = c) \end{bmatrix} = \begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix} \begin{bmatrix} P(S_{t-1} = s) \\ P(S_{t-1} = r) \\ P(S_{t-1} = c) \end{bmatrix}$$

Probability of tomorrow being cloud given that today is rainy

Probability of tomorrow being cloud given that today is cloudy

*This is just an illustrative example. It is not to say weather change is indeed a Markov process …

Markov process: weather transition example

- **Saying that something is a Markov process does not mean future is just independent of the past…it is independent of the past <u>given the present</u>**

$$\begin{bmatrix} P(S_t = s) \\ P(S_t = r) \\ P(S_t = c) \end{bmatrix} = \begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix} \begin{bmatrix} P(S_{t-1} = s) \\ P(S_{t-1} = r) \\ P(S_{t-1} = c) \end{bmatrix}$$

$$\begin{bmatrix} P(S_{t-1} = s) \\ P(S_{t-1} = r) \\ P(S_{t-1} = c) \end{bmatrix} = \begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix} \begin{bmatrix} P(S_{t-2} = s) \\ P(S_{t-2} = r) \\ P(S_{t-2} = c) \end{bmatrix}$$

$$\begin{bmatrix} P(S_t = s) \\ P(S_t = r) \\ P(S_t = c) \end{bmatrix} = \begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix} \begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix} \begin{bmatrix} P(S_{t-2} = s) \\ P(S_{t-2} = r) \\ P(S_{t-2} = c) \end{bmatrix}$$

Markov process: weather transition example

- Let's assume the following weather transition matrix

$$\begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix} = \begin{bmatrix} 0.7 & 0.3 & 0.5 \\ 0.1 & 0.4 & 0.2 \\ 0.2 & 0.3 & 0.3 \end{bmatrix}$$

- If today is sunny …tomorrow has 70% chance to be sunny, 10% chance to be rainy, and 20% chance to be cloudy

$$\begin{bmatrix} P(S_t = s) \\ P(S_t = r) \\ P(S_t = c) \end{bmatrix} = \begin{bmatrix} 0.7 & 0.3 & 0.5 \\ 0.1 & 0.4 & 0.2 \\ 0.2 & 0.3 & 0.3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.2 \end{bmatrix}$$

Markov process: weather transition example

- What would be the weather after 10 years?

$$\begin{bmatrix} P(S_t = s) \\ P(S_t = r) \\ P(S_t = c) \end{bmatrix} = \begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix}^{3655} \begin{bmatrix} P(S_{t-3650} = s) \\ P(S_{t-3650} = r) \\ P(S_{t-3650} = c) \end{bmatrix}$$

- Is the predicted weather after 3650 days going to be different from the weather after 3659 days? Not really…hence

$$\begin{bmatrix} P(S_{3650} = s) \\ P(S_{3650} = r) \\ P(S_{3650} = c) \end{bmatrix} = \begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix} \begin{bmatrix} P(S_{t-3649} = s) \\ P(S_{t-3649} = r) \\ P(S_{t-3649} = c) \end{bmatrix}$$

$$\begin{bmatrix} P(S_{3650} = s) \\ P(S_{3650} = r) \\ P(S_{3650} = c) \end{bmatrix} \approx \begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix} \begin{bmatrix} P(S_{t-3650} = s) \\ P(S_{t-3650} = r) \\ P(S_{t-3650} = c) \end{bmatrix}$$

Markov process: weather transition example

- Weather after 10 years

$$\begin{bmatrix} P(S_{3650}=s) \\ P(S_{3650}=r) \\ P(S_{3650}=c) \end{bmatrix} \approx \begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix} \begin{bmatrix} P(S_{t-3650}=s) \\ P(S_{t-3650}=r) \\ P(S_{t-3650}=c) \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0.7 & 0.3 & 0.5 \\ 0.1 & 0.4 & 0.2 \\ 0.2 & 0.3 & 0.3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

$$X = PX \qquad P = \begin{bmatrix} 0.7 & 0.3 & 0.5 \\ 0.1 & 0.4 & 0.2 \\ 0.2 & 0.3 & 0.3 \end{bmatrix} \quad X = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Markov process: weather transition example

- Weather after 10 years

$$X = PX$$

$$(P - I)X = 0$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.3 & 0.3 & 0.5 \\ 0.1 & -0.6 & 0.2 \\ 0.2 & 0.3 & -0.7 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad + \quad a + b + c = 1$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.3 & 0.3 & 0.5 \\ 0.1 & -0.6 & 0.2 \\ 0.2 & 0.3 & -0.7 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \Rightarrow \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0.5806 \\ 0.1774 \\ 0.2419 \end{bmatrix}$$

Markov process: weather transition example

- Let's check if our assumption is correct

$$\begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix}^{3655} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5806 \\ 0.1774 \\ 0.2419 \end{bmatrix}$$

$$\begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix}^{3655} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5806 \\ 0.1774 \\ 0.2419 \end{bmatrix} \quad \Longleftrightarrow \quad \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0.5806 \\ 0.1774 \\ 0.2419 \end{bmatrix}$$

$$\begin{bmatrix} p_{ss} & p_{sr} & p_{sc} \\ p_{rs} & p_{rr} & p_{rc} \\ p_{cs} & p_{cr} & p_{cc} \end{bmatrix}^{3655} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5806 \\ 0.1774 \\ 0.2419 \end{bmatrix}$$

Markov process (again)

- **Markov process** is a sequence of random events that meets the Markov property

$$P(S_t = s_n | S_{t-1} = s_{t-1}, \cdots, S_0 = s_0) = P(S_t = s_t | S_{t-1} = s_{t-1})$$

- To define a Markov process, we need
  - S: a set of states
  - P: state transition probability $P(S_t = s' | S_{t-1} = s)$

$$(S_0), (S_1), (S_2), (S_3), (S_4), \cdots$$

Markov reward process (MRP)

- **Markov reward process:**
  - is a Markov process with a reward received after each state transition

- To define a Markov reward process, we need
  - S: a set of states
  - P: state transition probability $P(S_t = s' | S_{t-1} = s)$
  - R: reward function $R(S_t = s) = E[r_t | S_t = s]$
  - Discount factor γ

Reward may be stochastic

To quantify the value of future reward (compared to the immediate reward)

$$(S_0, R_0), (S_1, R_1), (S_2, R_2), \cdots$$

Return

- **Horizon:** number of time steps until the end of state transitions

- **Return**: discounted sum of reward from time step t to horizon

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

  - To incorporate “If it’s a good thing, it’s better to receive early”
  - $\gamma=0$: I don’t care about future reward (YOLO?)
  - $\gamma=1$: future reward is as good as immediate reward

Remember that reward and return are different!
We will also talk about ‘value’ which is basically <u>expected return</u>

State value function

- **State value function (for a MRP):**
  expected return from starting in state s

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

$$= E[r_t|S_t = s] + \gamma E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots|S_t = s]$$

$$= R(s) + \gamma V(G_{t+1}|S_t = s)$$

$$= R(s) + \gamma V(\text{next state}|S_t = s)$$

$$= R(s) + \gamma \sum_{s'} P(s'|s)V(s')$$

$$\boxed{V(s) = R(s) + \gamma \sum_{s'} P(s'|s)V(s')}$$

Bellman equation:
value = present value + future value

Immediate (expected) reward **Discounted sum of future (expected) reward**

State value function (for finite state MRP)

$$V(s) = R(s) + \gamma \sum_{s'} P(s'|s)V(s')$$

$$\begin{bmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_N) \end{bmatrix} = \begin{bmatrix} R(s_1) \\ R(s_2) \\ \vdots \\ R(s_N) \end{bmatrix} + \begin{bmatrix} P_{11} & P_{21} & \cdots & P_{1N} \\ P_{21} & P_{22} & \cdots & P_{2N} \\ \vdots & \vdots & \cdots & \vdots \\ P_{N1} & P_{N2} & \cdots & P_{NN} \end{bmatrix} \begin{bmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_N) \end{bmatrix}$$

$$V = R + \gamma P V$$

$$(I - \gamma P)V = R$$

$$V = (I - \gamma P)^{-1} R$$

We can calculate V, if P, R, γ are given

Markov decision process (MDP)

- **Markov decision process:**
  - is a Markov reward process whose state transition probabilities and the reward depend on the present action

- To define a Markov decision process, we need
  - S: a set of states
  - A: a set of actions
  - P: state transition probability $P(S_t = s' | S_{t-1} = s, a_t = a)$
  - R: reward function $R(S_t = s, a_t = a) = E[r_t | S_t = s, a_t = a]$
  - Discount factor γ

$$(S_0, A_0, R_0), (S_1, A_1, R_1), (S_2, A_2, R_2), \cdots$$

Policy in MDP

- MDP requires an actions at each state
- We can consider a function that tell us what to do at each state
  - Input: state
  - Output: probability to perform each action
- **Policy**:

$$\pi(a|s) = P(A_t = a|S_t = s)$$

$$(S_0, A_0, R_0), (S_1, A_1, R_1), (S_2, A_2, R_2), \cdots$$

Now, both actions and rewards depend on the policy π

Policy in MDP

- **Policy:** $\pi(a|s) = P(A_t = a|S_t = s)$

$$(S_0, A_0, R_0), (S_1, A_1, R_1), (S_2, A_2, R_2), \cdots$$

both actions and rewards depend on the policy π

$$P^\pi(s'|s) = \sum_a \pi(a|s)P(s'|s, a)$$

$$R^\pi(s) = \sum_a \pi(a|s)R(s, a)$$

State value function of MDP

- **State value function (for a MDP):**
  expected return from starting in state s

$$V(s) = R(s) + \gamma \sum_{s'} P(s'|s)V(s')$$

$$\Downarrow$$

In MDP, state value depends on π

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

Immediate reward depends on action and hence on policy

Transition probability depends on action and hence on policy

Optimal policy in MDP

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

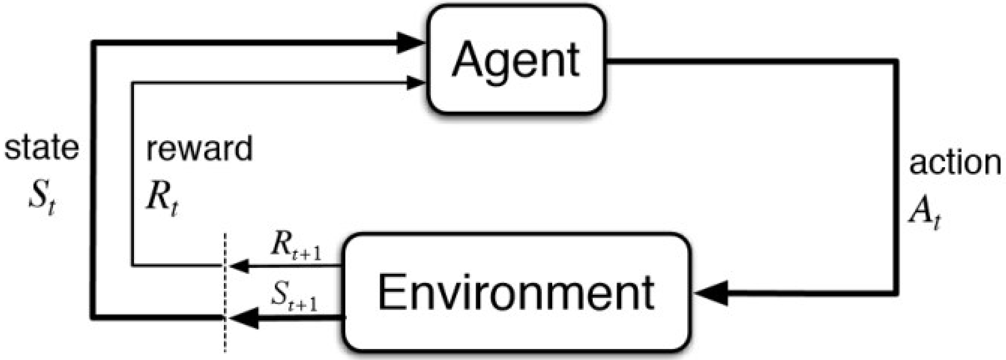
- Goal is to maximize the expected return by choosing a sequence of actions
- This is equivalent to finding a policy that maximize the value function
- Hence, we want to find the optimal policy such that

$$\pi^{*}(s) = \arg\max_{\pi} V^{\pi}(s)$$

Thus, one way of reinforcement learning is to directly find the optimal policy π^*
(which means there are other ways, too)

Reinforcement learning

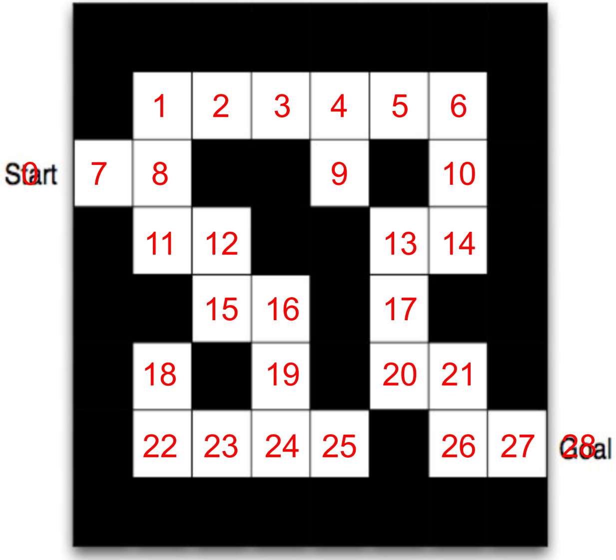
- The goal of reinforcement learning is to learn to generate the <u>best sequence of actions</u> for a given Markov decision process



$$(S_0, A_0, R_0), (S_1, A_1, R_1), (S_2, A_2, R_2), \cdots$$

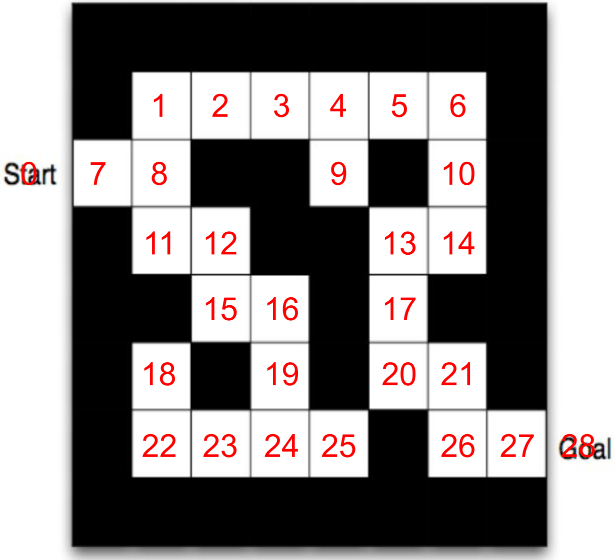
Again, how do we define the best?

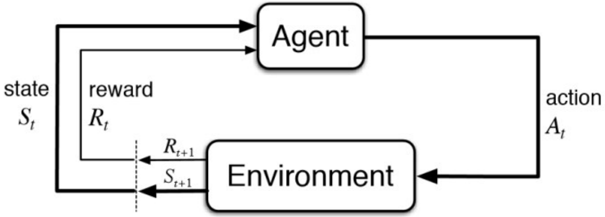
Maze example (revisited)



- **Formulation**
  - **State:**
    an integer number between 0 and 28
  - **Action:**
    an integer number between 0 and 3
    (0:up, 1: down, 2: left, 3: right)
  - **Reward**:
    -1 at each time step
  - **The “agent”**
    - a function f that takes current state as the input and calculates the best action

Maze example (revisited)





$$
\begin{aligned}
&(S_0 = 0, A_0 = 3, R_0 = -1),\\
&(S_1 = 7, A_1 = 3, R_1 = -1),\\
&(S_2 = 8, A_2 = 0, R_2 = -1),\\
&\ldots\\
&(S_k = 27, A_k = 3, R_k = -1)
\end{aligned}
$$

0:up, 1: down, 2: left, 3: right

The Agent

- …is essentially a function
- It can be a **policy function** that determines the action probability given state (input: state, output: action probability)

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

  - If we have a policy function, we can take action by following the action probability
- It can also be a **state action value function** (input: state & action, output: value)

$$Q(s, a) = E[G_t|S_t = s, A_t = a]$$

  - If we have a value function, we can pick the action with the highest value for the given state (among all possible actions)

- It can also be something else …
- Now, let's say our goal is to find either <u>the policy that maximize the cumulative reward</u> or find <u>the state-action value function in the given environment</u>

Bellman equation: "clue" for learning

$$\boxed{V(s) = R(s) + \gamma \sum_{s'} P(s'|s)V(s')}$$

Bellman equation (for MRP):
value = present value + future value

$$\boxed{Q_\pi(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s)Q_\pi(s',a')}$$

Bellman equation (for MDP):
value = present value + future value

If the policy $\boldsymbol{\pi}$ is optimal (i.e., $\boldsymbol{\pi} = \boldsymbol{\pi}^*$),
the following equation holds

$$\sum_{s'} P(s'|s)Q_{\pi*}(s',a') = \max_{a'} Q_{\pi*}(s',a')$$

hence

$$\boxed{Q_{\pi*}(s,a) = R(s,a) + \gamma \max_{a'} Q_{\pi*}(s',a')}$$

Bellman equation for optimal policy

Bellman equation: “clue” for learning

Now we know that the following equation should be met for optimal policy

$$Q_{\pi*}(s,a) = R(s,a) + \gamma \max_{a'} Q_{\pi*}(s',a')$$

$$\Downarrow$$

$$Q_{\pi*}(s,a) - R(s,a) - \gamma \max_{a'} Q_{\pi*}(s',a') = 0$$

In other words, we can consider
any deviation from this condition as an error

$$\epsilon = Q_{\pi}(s,a) - R(s,a) - \gamma \max_{a'} Q_{\pi}(s',a')$$

Updating the state-action value function (Q-function)

$$\epsilon = Q_\pi(s,a) - R(s,a) - \gamma \max_{a'} Q_\pi(s',a')$$

Let’s subtract the error (after multiplying with a learning rate α)

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) - \alpha\epsilon$$

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha(R(s,a) + \gamma \max_{a'} Q_\pi(s',a') - Q_\pi(s,a))$$

Repeating this update will give us more accurate Q-function!

Finding the Q-function

- Ok, we have the equation for iterative updates. Are we done?

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha(R(s,a) + \gamma \max_{a'} Q_\pi(s',a') - Q_\pi(s,a))$$

- The problem is that this Q-function takes both state and action as the input

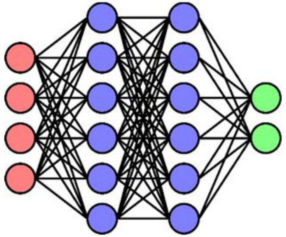
$$Q : S \times A \rightarrow \mathcal{R}$$

- We have to update the value for all possible state-action combinations
  → learning becomes infeasible when state-action space is large

Neural network as an agent

- Neural network is a powerful method to approximate an arbitrary function

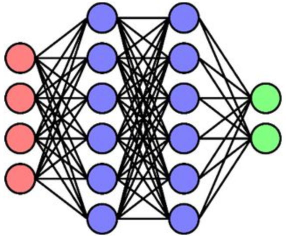
input: state (as an array)



output: probability to take each action

policy function

input: state & action

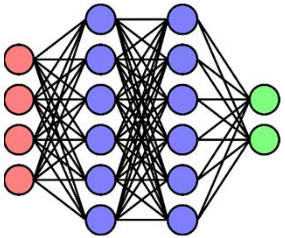


output: value of the state & action

state-action value function (Q-function)

Deep Q-Network (DQN)

input: **state & action**



output: **value of the state & action**

state-action value function (Q-function)

- Instead of trying to find the true Q function (which is infeasible when state-action space is large), we may choose to find an approximated Q function
- Now, the problem has changed to finding a set of network parameters θ that approximates the true Q function

Updating DQN

- This is something we want to reduce

$$\epsilon = Q_\pi(s,a) - R(s,a) - \gamma \max_{a'} Q_\pi(s',a')$$

- Q-function is now represented as a network (parameterized!)

$$\epsilon = Q_\pi(s,a;\boxed{\theta}) - R(s,a) - \gamma \max_{a'} Q_\pi(s',a';\boxed{\theta})$$

- We can think of a loss function as follows

$$\mathcal{L}(\theta) = \frac{1}{2}\epsilon^2 = \frac{1}{2}(Q_\pi(s,a;\theta) - R(s,a) - \gamma \max_{a'} Q_\pi(s',a';\theta))^2$$

- How can we minimize it? Gradient descent?

Updating DQN

- Small modification before we go further

$$\mathcal{L}(\theta) = \frac{1}{2}\epsilon^2 = \frac{1}{2}(Q_\pi(s,a;\theta) - (R(s,a) + \gamma \max_{a'} Q_\pi(s',a';\theta)))^2$$

Our Q-function

Target Q-function

  - We will change the “target” part to be non-parametric component that we want our Q-function to match*

$$\mathcal{L}(\theta) = \frac{1}{2}(Q_\pi(s,a;\theta) - (R(s,a) + \gamma \max_{a'} Q_\pi(s',a';\theta^-)))^2$$

This can be considered as parameters from earlier steps..

$$\nabla\mathcal{L}(\theta) = (Q_\pi(s,a;\theta) - (R(s,a) + \gamma \max_{a'} Q_\pi(s',a';\theta^-)))\nabla Q_\pi(s,a;\theta)$$

$$\theta^{(k+1)} = \theta^{(k)} - \alpha\nabla\mathcal{L}(\theta^{(k)})$$

This is not true gradient descent. This type of algorithm is called semi-gradient algorithm.

Updating DQN

- Q) Can we just directly minimize the following?

$$\mathcal{L}(\theta) = \frac{1}{2}\epsilon^2 = \frac{1}{2}(Q_\pi(s,a;\theta) - (R(s,a) + \gamma \max_{a'} Q_\pi(s',a';\theta)))^2$$

- …by calculating its true gradient

$$\nabla\mathcal{L}(\theta) = (Q_\pi(s,a;\theta) - (R(s,a) + \gamma \max_{a'} Q_\pi(s',a';\theta)))(\nabla Q_\pi(s,a;\theta) - \gamma \nabla Q_\pi(s',a';\theta))$$

- …and then performing gradient descent update?

$$\theta^{(k+1)} = \theta^{(k)} - \alpha \nabla\mathcal{L}(\theta^{(k)})$$

- A) It works, but turns out to be slower than semi-gradient method*
    - Conceptually speaking, it is about whether we want to update only Q(s,a) using R+Q(s',a'), or both Q(s,a) and Q(s',a'). The short answer is, we may not want to update Q(s',a')

*See 'Residual Algorithms: Reinforcement Learning with Function Approximation' for more information

Updating DQN

- Another way to interpret the algorithm
  - We want our approximated Q-function to be close to the true Q-function

$$\epsilon = Q_\pi(s, a; \theta) - Q_{true}(s, a)$$

  - Then we can define a loss function as follows

$$\mathcal{L}(\theta) = \frac{1}{2}\epsilon^2 = \frac{1}{2}(Q_\pi(s, a; \theta) - Q_{true}(s, a))^2$$

  - Then, calculate the gradient

$$\nabla\mathcal{L}(\theta) = (Q_\pi(s, a; \theta) - \boxed{Q_{true}(s, a)})\nabla Q_\pi(s, a; \theta)$$

  - Since we do not know the true Q-function, we may just use a target function (which is supposedly better than the current one)

$$\nabla\mathcal{L}(\theta) = (Q_\pi(s, a; \theta) - \boxed{(R(s, a) + \gamma \max_{a'} Q_\pi(s', a'; \theta^-))})\nabla Q_\pi(s, a; \theta)$$

Updating DQN

- Another way to interpret the algorithm
  - Temporal difference learning

$$\nabla \mathcal{L}(\theta) = (\underbrace{\boxed{Q_\pi(s,a;\theta)}}_{\text{Our Q-function}} - \underbrace{\boxed{(R(s,a) + \gamma \max_{a'} Q_\pi(s',a';\theta^-))}}_{\text{Target Q-function}})\nabla Q_\pi(s,a;\theta)$$

  - Our Q-function is from the newest parameters where the target Q-function uses old parameters. How can it be possibly any better?
    - It is from the “next state” which has actual reward in it (at least the present value is accurate)
    - Let’s say today is Tuesday. It would be easier to predict the weather of Thursday, once we know the weather of Wednesday

Training DQN

- Ok, now we have the equation for the parameter update
- But, this is assuming that we have all training data

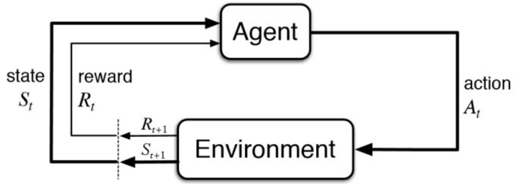
$$(S_0, A_0, R_0), (S_1, A_1, R_1), (S_2, A_2, R_2), \cdots$$

- Training data in RL comes from "interacting with environment"

- We are still missing a few things to implement RL
  - Interacting with environment
  - Exploration strategy
  - …and more

Interacting with environment

- So far, we have neglected the importance of interacting with the environment



- What we have discussed requires the following sequence (called episode)

$$(S_0, A_0, R_0), (S_1, A_1, R_1), (S_2, A_2, R_2), \cdots$$

- Our agent has to actually interact with the environment to generate the sequence

$$(S_0, A_0, R_0), (S_1, A_1, R_1), (S_2, A_2, R_2), \cdots$$

These can be obtained from the agent

Exploration: ε-greedy algorithm

$$(S_0, A_0, R_0), (S_1, A_1, R_1), (S_2, A_2, R_2), \cdots$$

These can be obtained from the agent

- If the agent keeps choosing poor actions (which obviously occurs at the early stage of learning), how can we learn the good actions?
  → Need to generate (at least some) actions that are different from the agent's choice
- **Simple solution**: with a probability ε, let the agent choose a random action (rather than the optimal action generated by Q-network) for the sake of generating data for training

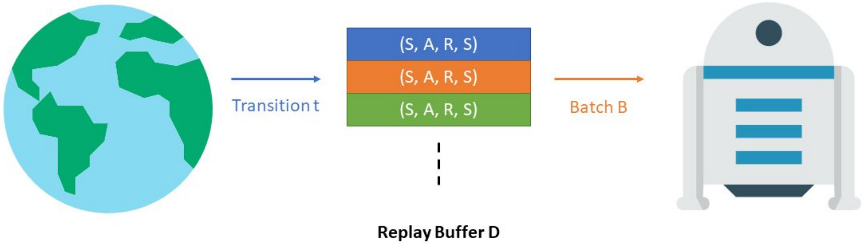
Replay buffer

- When we generate an episode …

$$(S_0, A_0, R_0), (S_1, A_1, R_1), (S_2, A_2, R_2), \cdots$$

These are highly correlated

- Therefore, it is better to collect data from multiple episodes, shuffle them, and then use for training



$$
\begin{array}{l}
(S_{15}, A_{15}, R_{15}, S_{16}) \\
(S_{88}, A_{88}, R_{88}, S_{89}) \\
(S_0, A_0, R_0, S_1) \\
(S_3, A_3, R_3, S_4) \\
\vdots \\
(S_{35}, A_{35}, R_{35}, S_{36}) \\
(S_{64}, A_{64}, R_{64}, S_{65})
\end{array}
$$

DQN: putting things together

```
Initialize replay buffer
Initialize Q network with random weights θ
Initialize target network (Q') with θ⁻ = θ
for episode in range(n_episode):
        Initialize sequence s_1
        for t in range(t_max):
                if random variable < ε:
                        select random action a_t
                else:
                        select a_t = argmax_a Q(s_t, a;θ)
                perform action a_t and take r_t and s_{t+1}
                store transition (s_t, a_t, r_t, s_{t+1}) in buffer

                sample random mini batch of transitions (s_k, a_k, r_k, s_{k+1}) from buffer
                if episode terminates at step k+1:
                        y_j = r_j
                else:
                        y_j = r_j + γ argmax_a Q(s_j, a;θ⁻)
                Update θ ← θ - α∇(y_j - Q(s_j, a_j;θ⁻))²
                Update θ⁻ ← θ @ every C steps
```

Summary

- Overview of reinforcement learning (Maze example)
  - Credit assignment problem
- Mathematical backgrounds
  - Random variable, random process
  - Markov process
  - Markov reward process, Markov decision process
  - Return and value function
- Q-function & Optimal policy
- Bellman equation
- Deep Q learning

References

- Lecture notes
  - Berkeley CS285
    - http://rail.eecs.berkeley.edu/deeprlcourse/
  - Stanford CS234
    - https://web.stanford.edu/class/cs234/