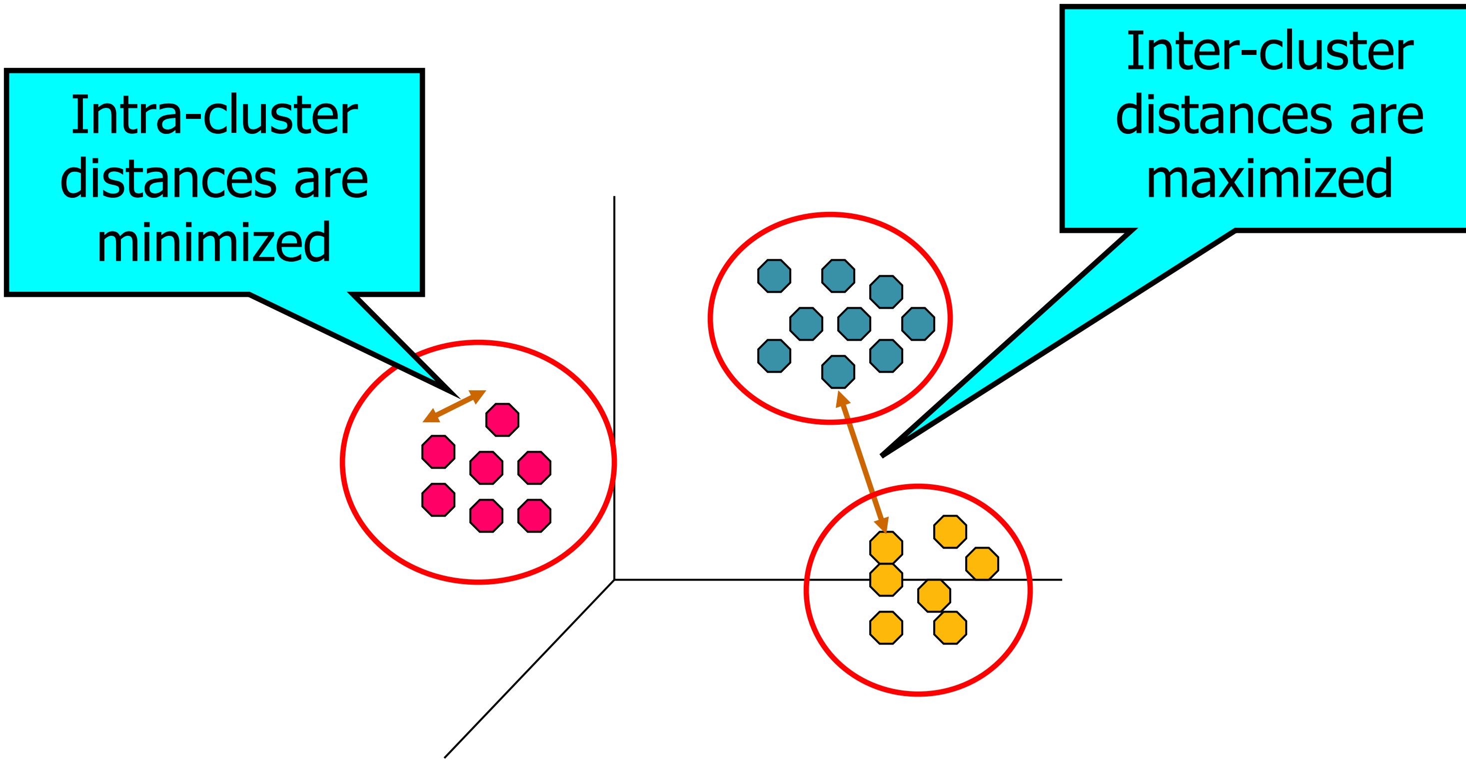
人工智能：机器学习 VI

饶洋辉
计算机学院,
中山大学
raoyangh@mail.sysu.edu.cn
http://cse.sysu.edu.cn/node/2471

课件来源：中山大学刘咏梅教授；陈川、余超副教授等

聚类

- 目标：基于距离度量，将对象集合聚类到簇(cluster)中，使得簇内对象的距离尽量小，且簇之间对象的距离尽量大。



簇的数量

How many clusters?

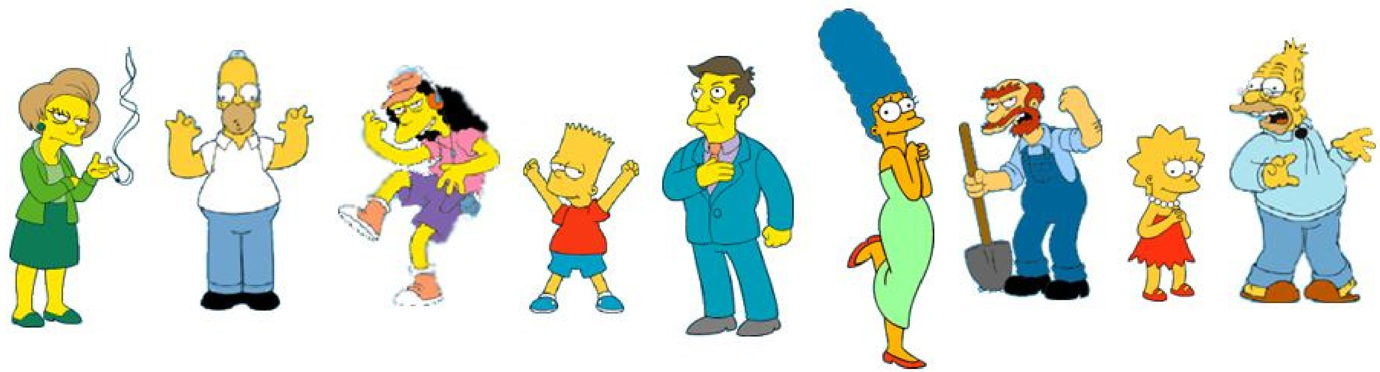
Six Clusters

Two Clusters

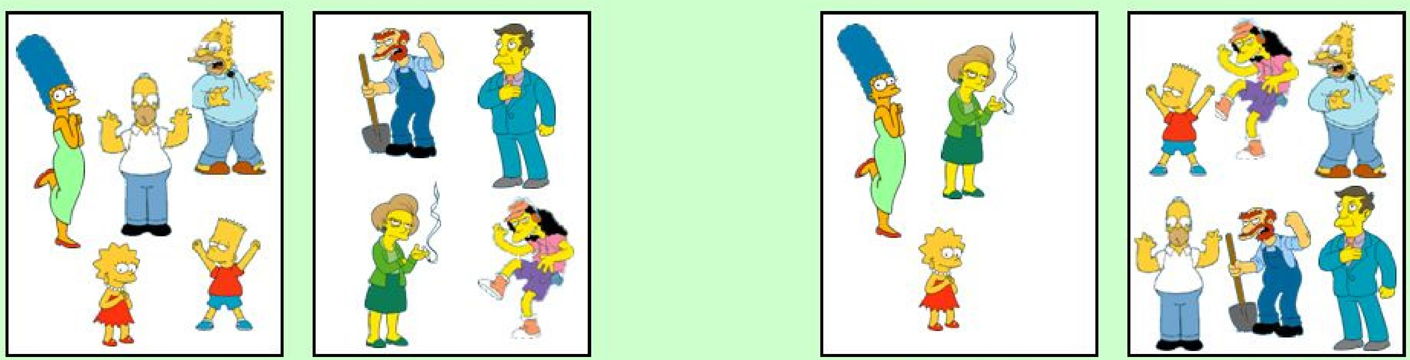
Four Clusters

簇的数量

What is a natural grouping among these objects?



Clustering is subjective



Simpson's Family School Employees Females Males

聚类的类型

- 划分式聚类
  - Divide data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset
  - # clusters is needed, e.g., *k*-Means….
- 基于密度的聚类
  - A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density
  - Used when the clusters are irregular or intertwined (不规则或纠缠), and when noise and outliers are present
- 层次聚类
  - A set of nested clusters organized as a hierarchical tree
  - # clusters is not needed

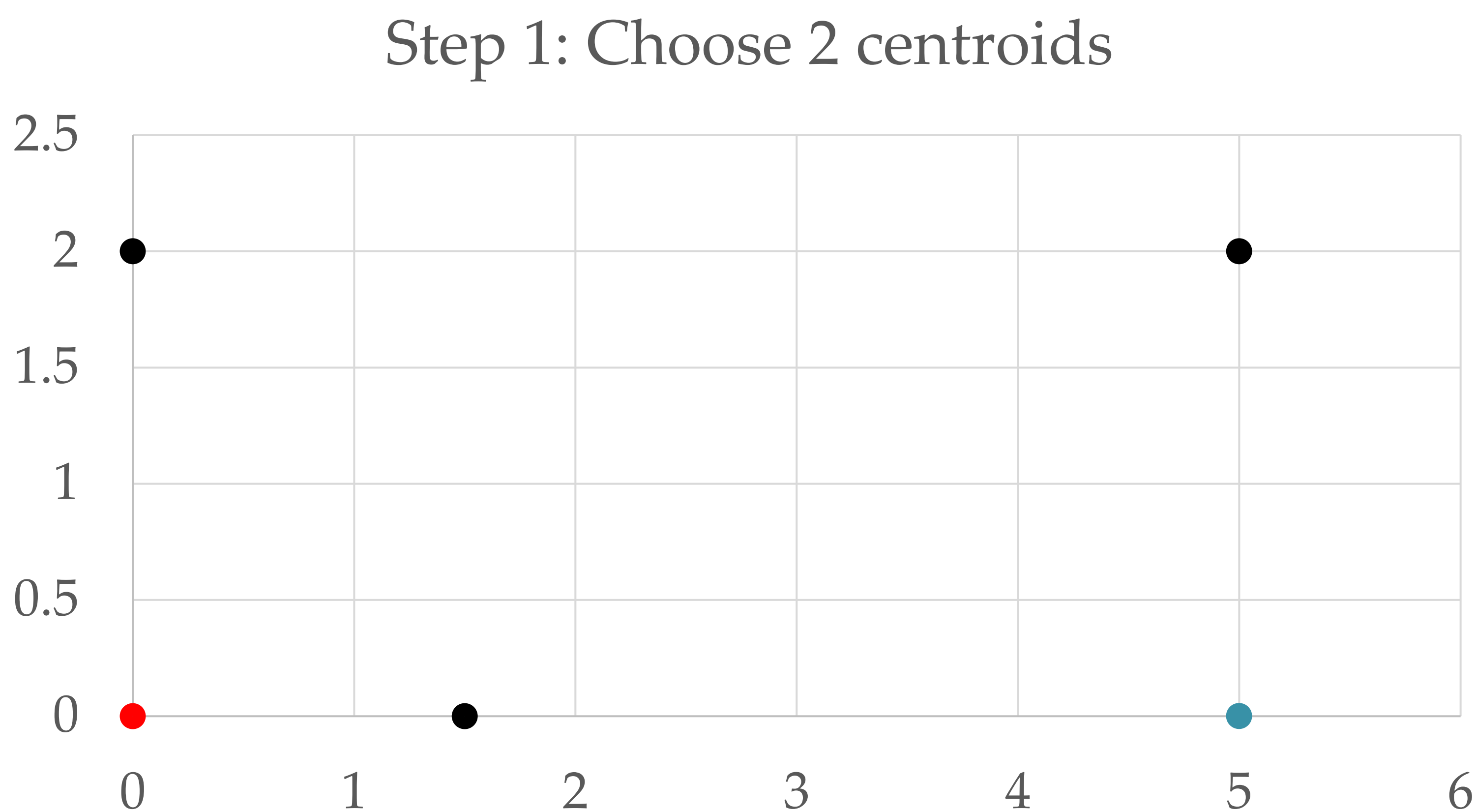
划分式聚类

- *k*-Means: 重复如下步骤…
  - 选择任意 *k* 个质心(centroids)
  - 将每个文档分配到最近的质心
  - 重新计算质心

- *k*-Means (划分法) 示例：
  - x1 = (0, 2), x2 = (0, 0), x3 = (1.5, 0), x4 = (5, 0), x5 = (5, 2)
  - $k = 2$

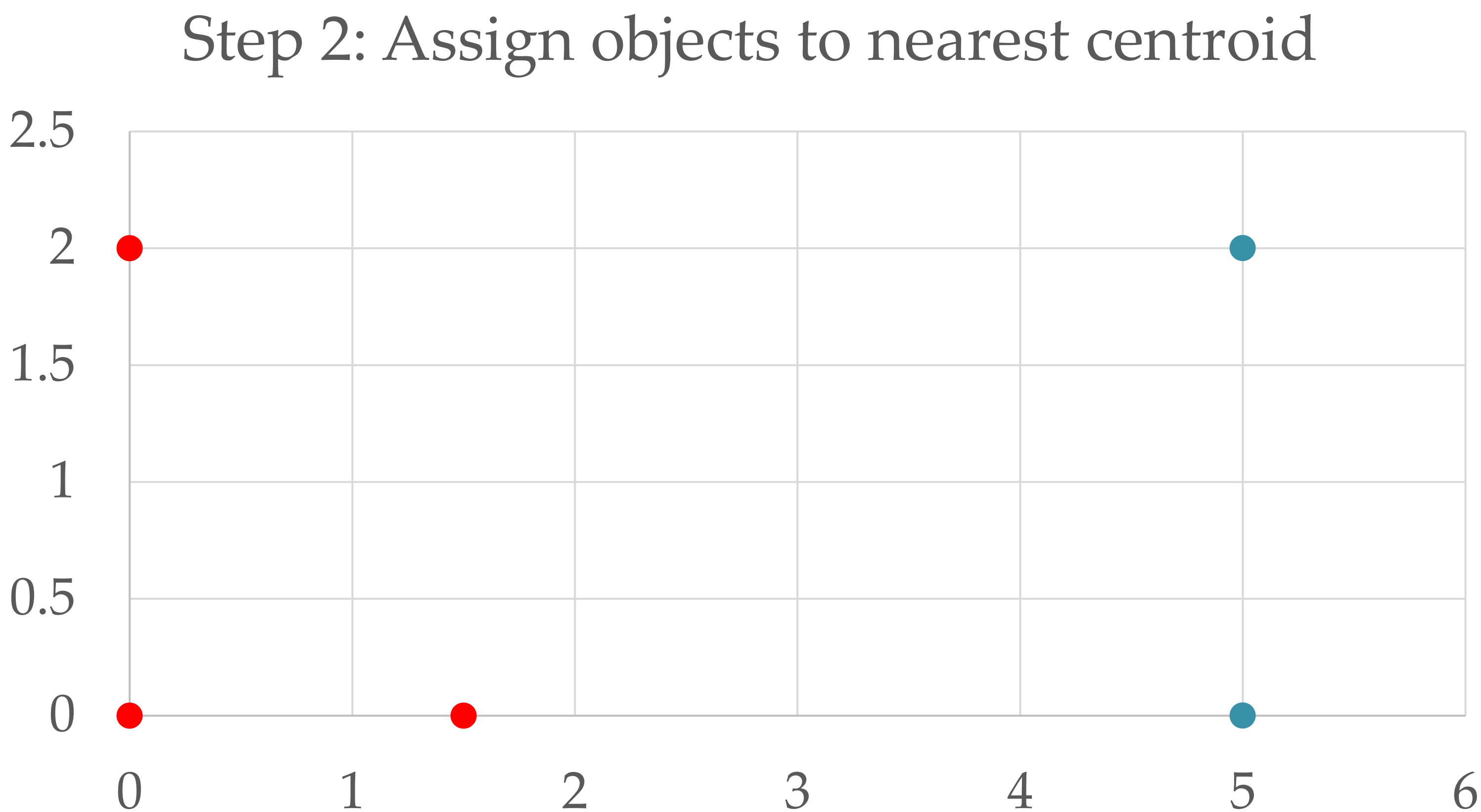
k-Means

x1 = (0, 2), x2 = (0, 0), x3 = (1.5, 0), x4 = (5, 0), x5 = (5, 2); $k = 2$



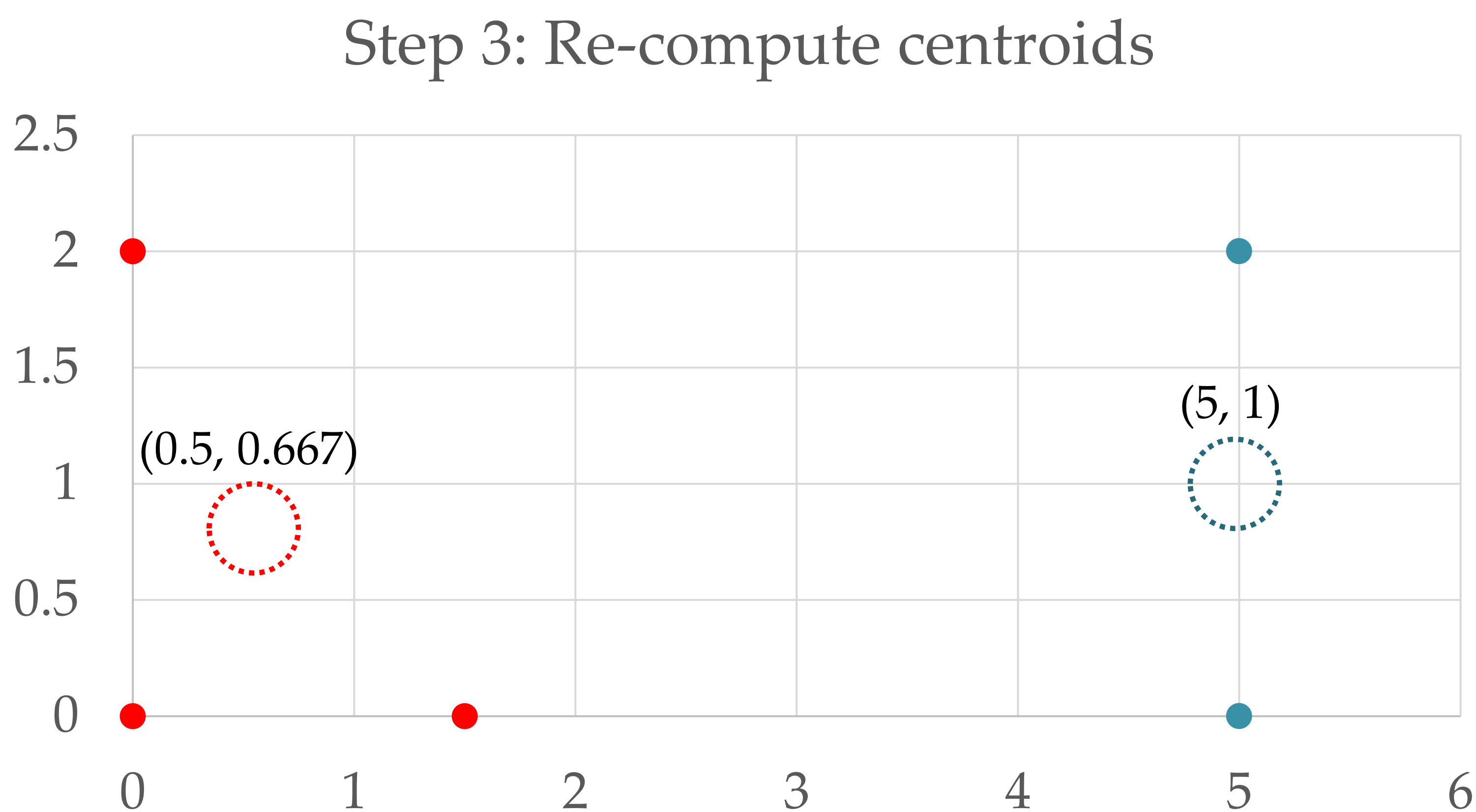
k-Means

x1 = (0, 2), x2 = (0, 0), x3 = (1.5, 0), x4 = (5, 0), x5 = (5, 2); $k = 2$



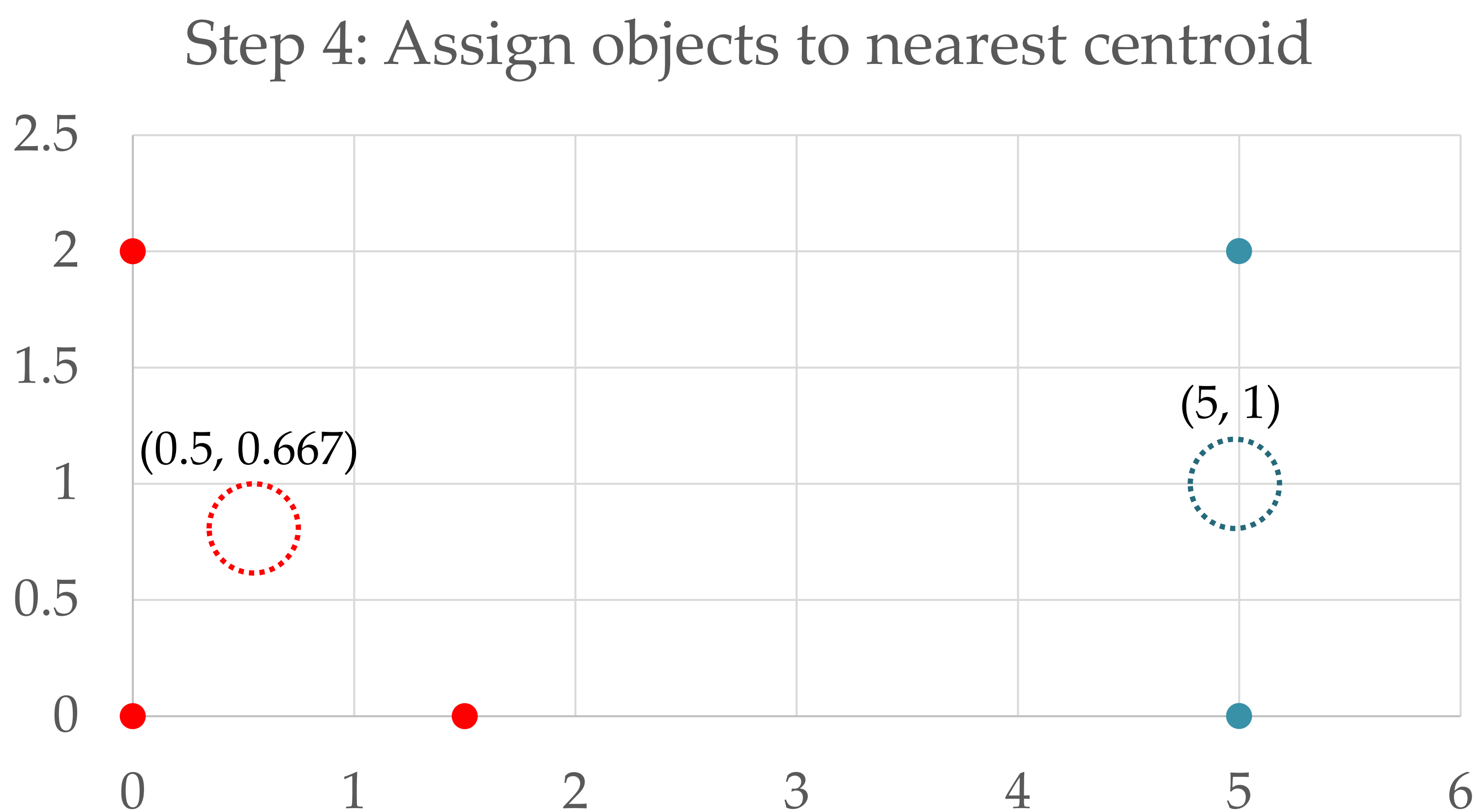
k-Means

x1 = (0, 2), x2 = (0, 0), x3 = (1.5, 0), x4 = (5, 0), x5 = (5, 2); $k = 2$



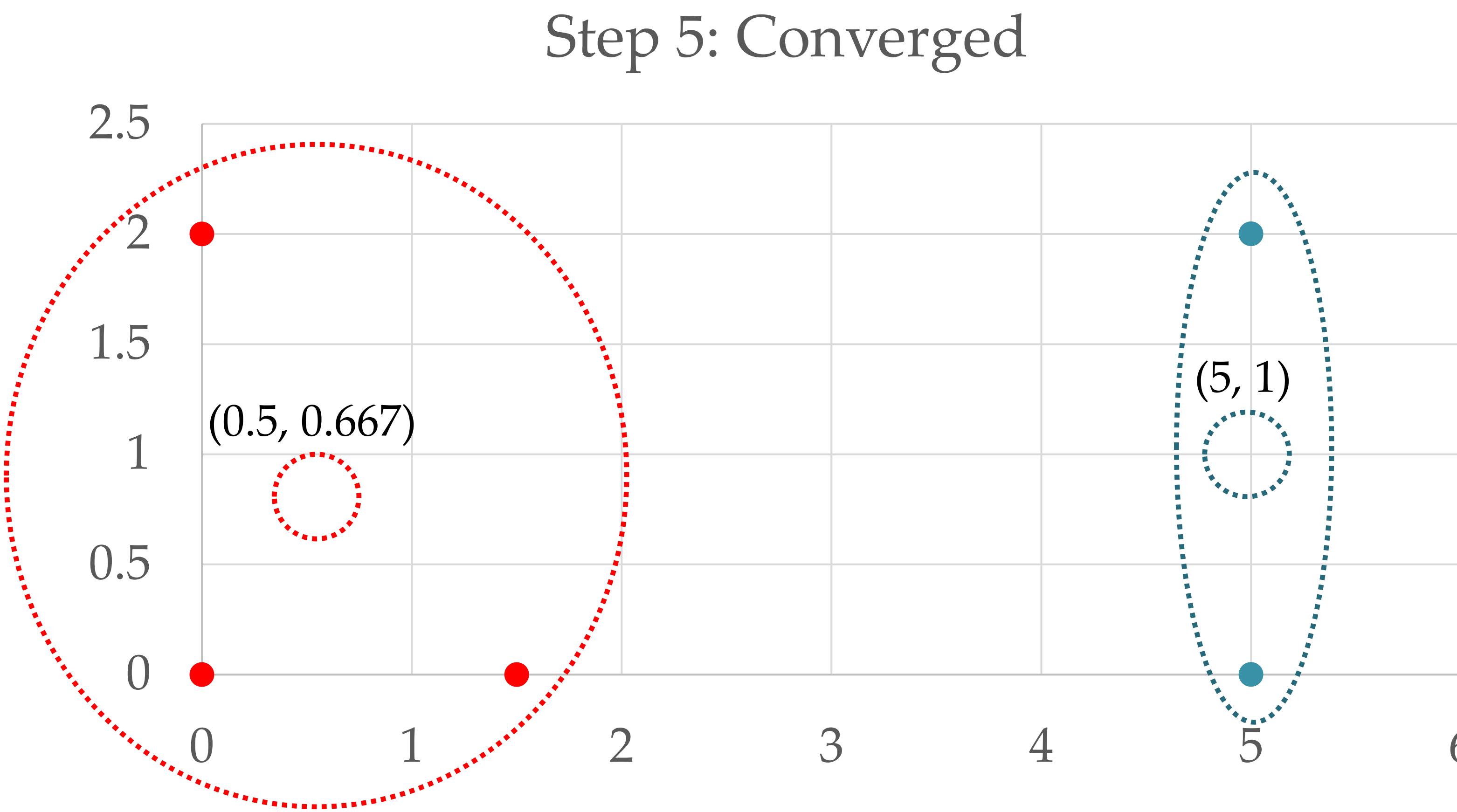
k-Means

x1 = (0, 2), x2 = (0, 0), x3 = (1.5, 0), x4 = (5, 0), x5 = (5, 2); *k* = 2



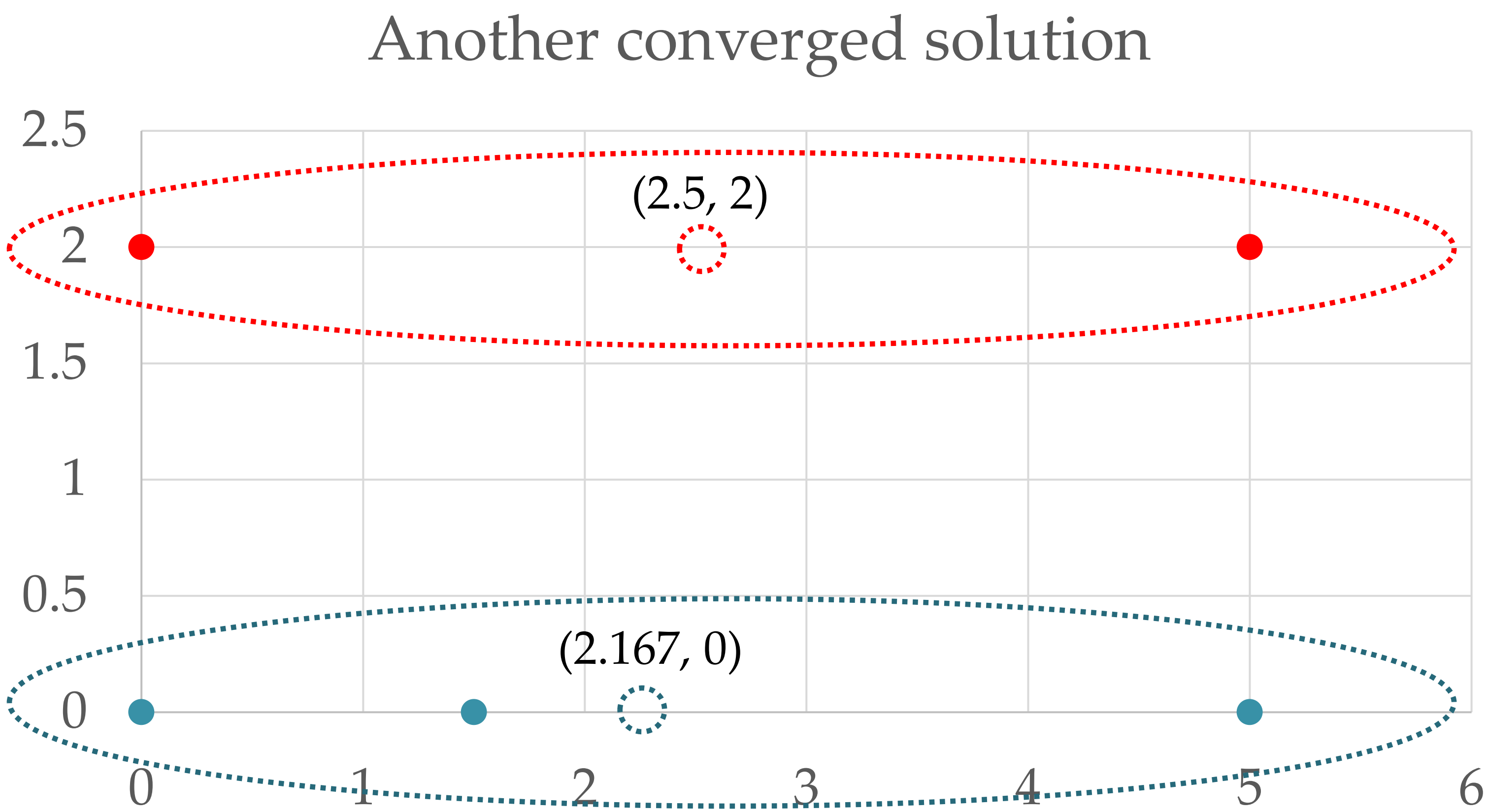
k-Means

x1 = (0, 2), x2 = (0, 0), x3 = (1.5, 0), x4 = (5, 0), x5 = (5, 2)； $k = 2$



k-Means

x1 = (0, 2), x2 = (0, 0), x3 = (1.5, 0), x4 = (5, 0), x5 = (5, 2); $k = 2$



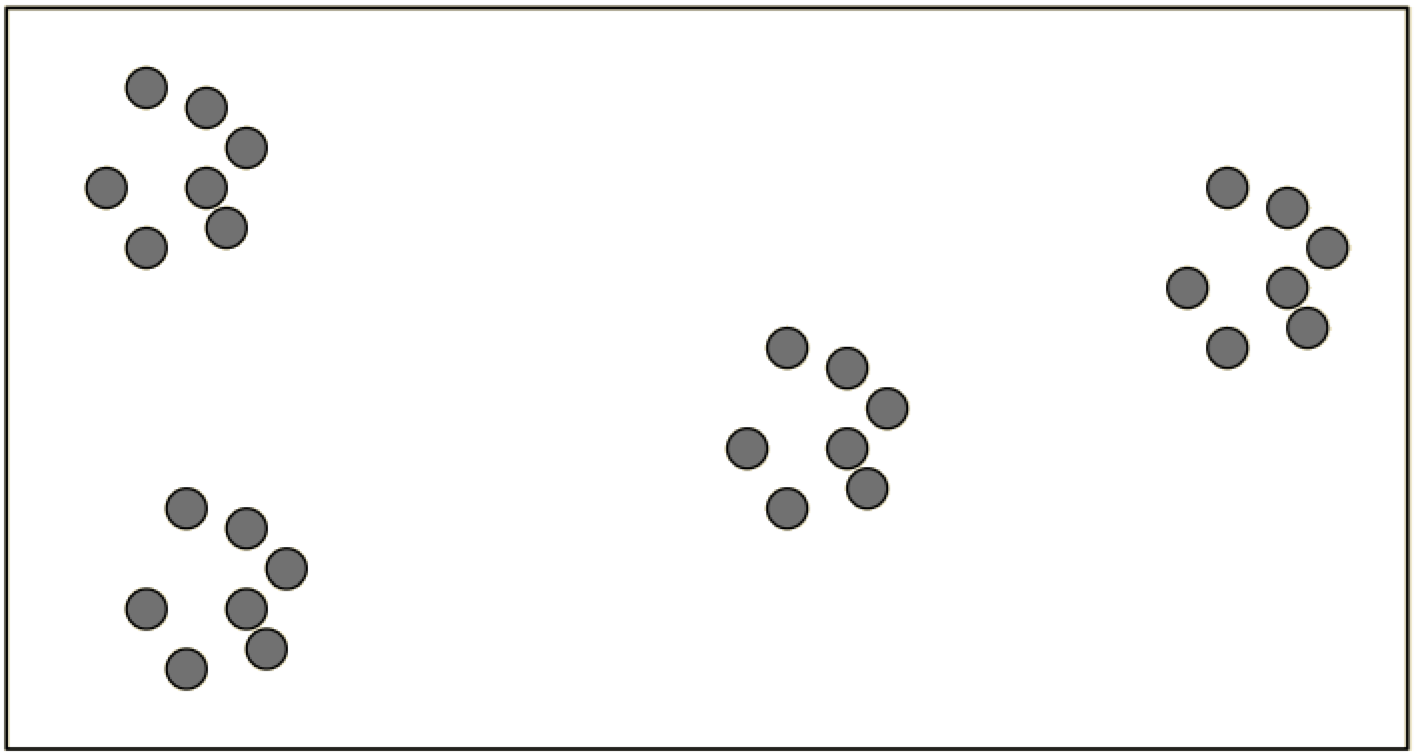
k-Means++

- Spreads out the centers
- Choose first center, x_1, uniformly at random from the data set
- Repeat for $2 \le i \le k$:
  - Choose x' to be equal to a data point sampled from the distribution:

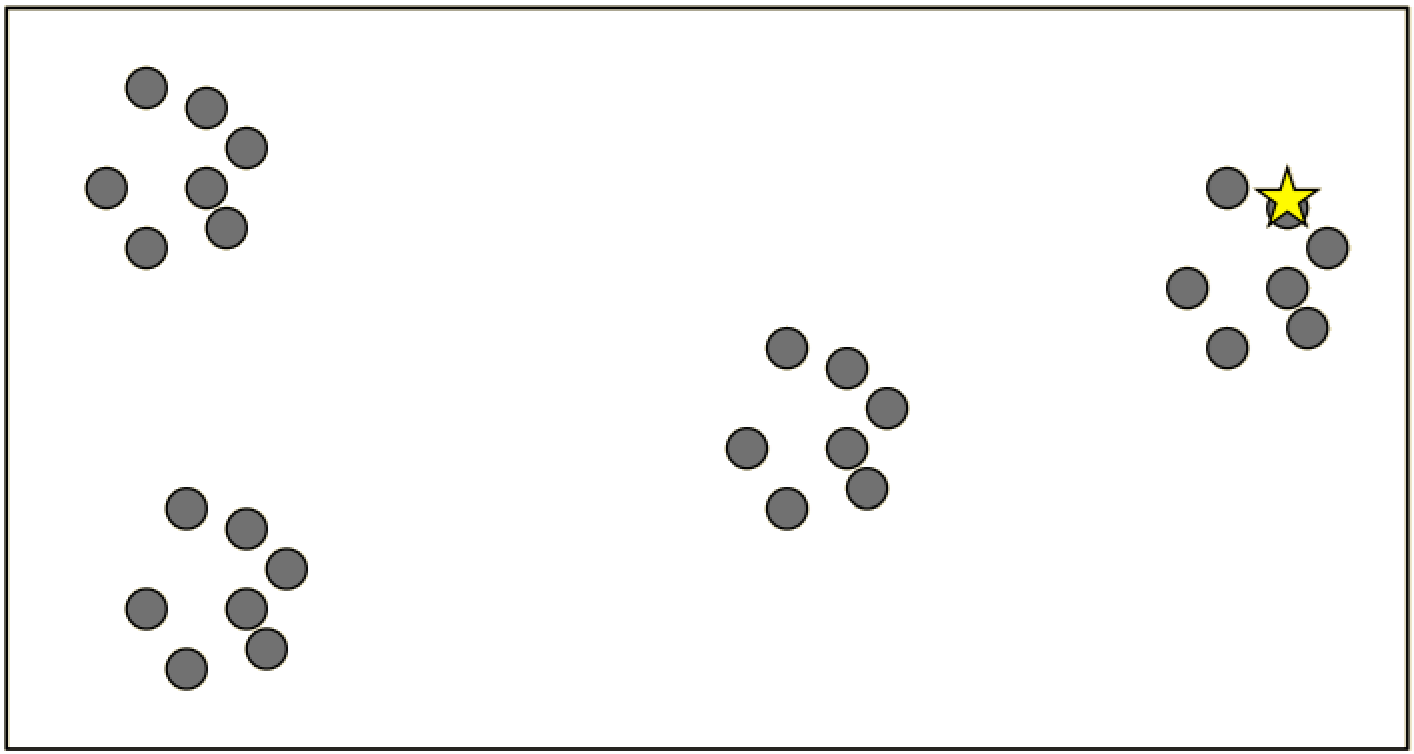
$$\frac{D(x')^2}{\sum_{x \in \mathcal{X}} D(x)^2}$$

- $D(x)$: the shortest distance from a data point x to the closest center we have already chosen

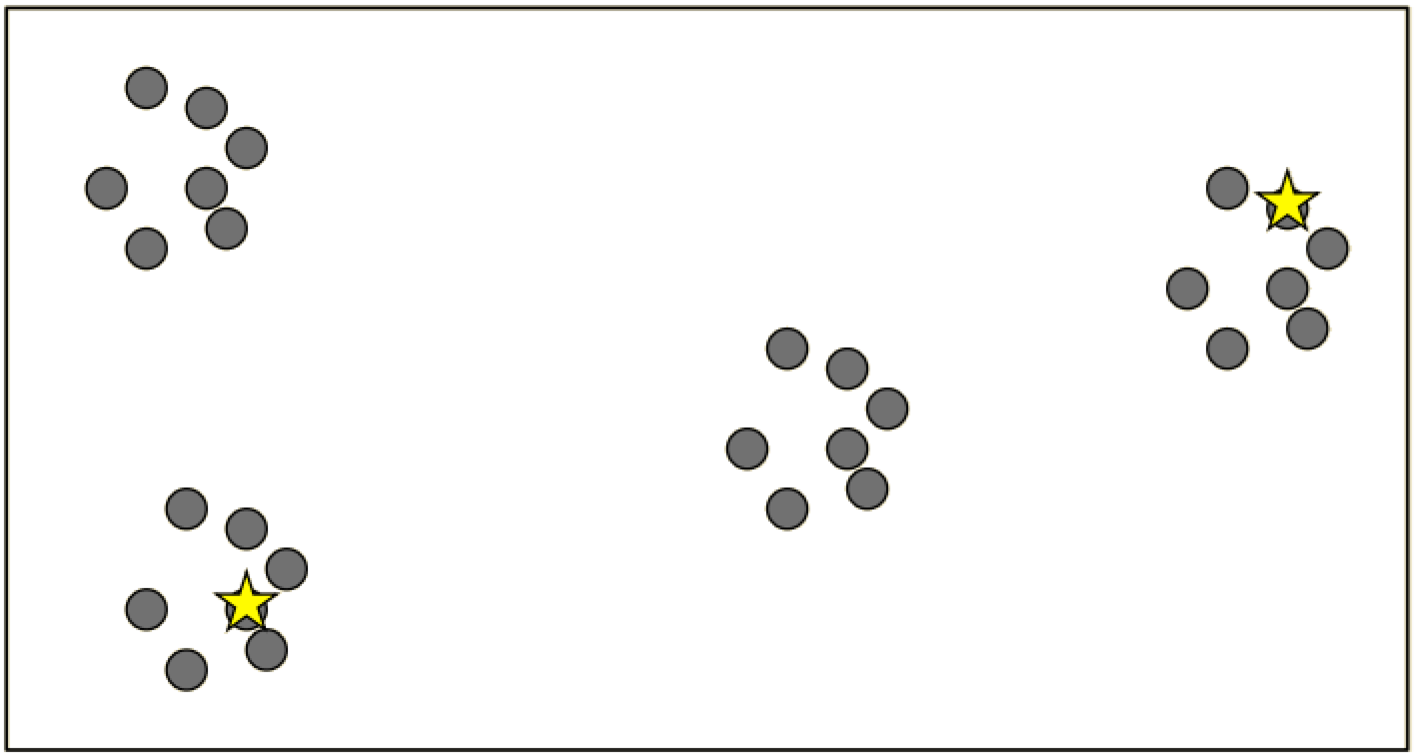
k-Means++



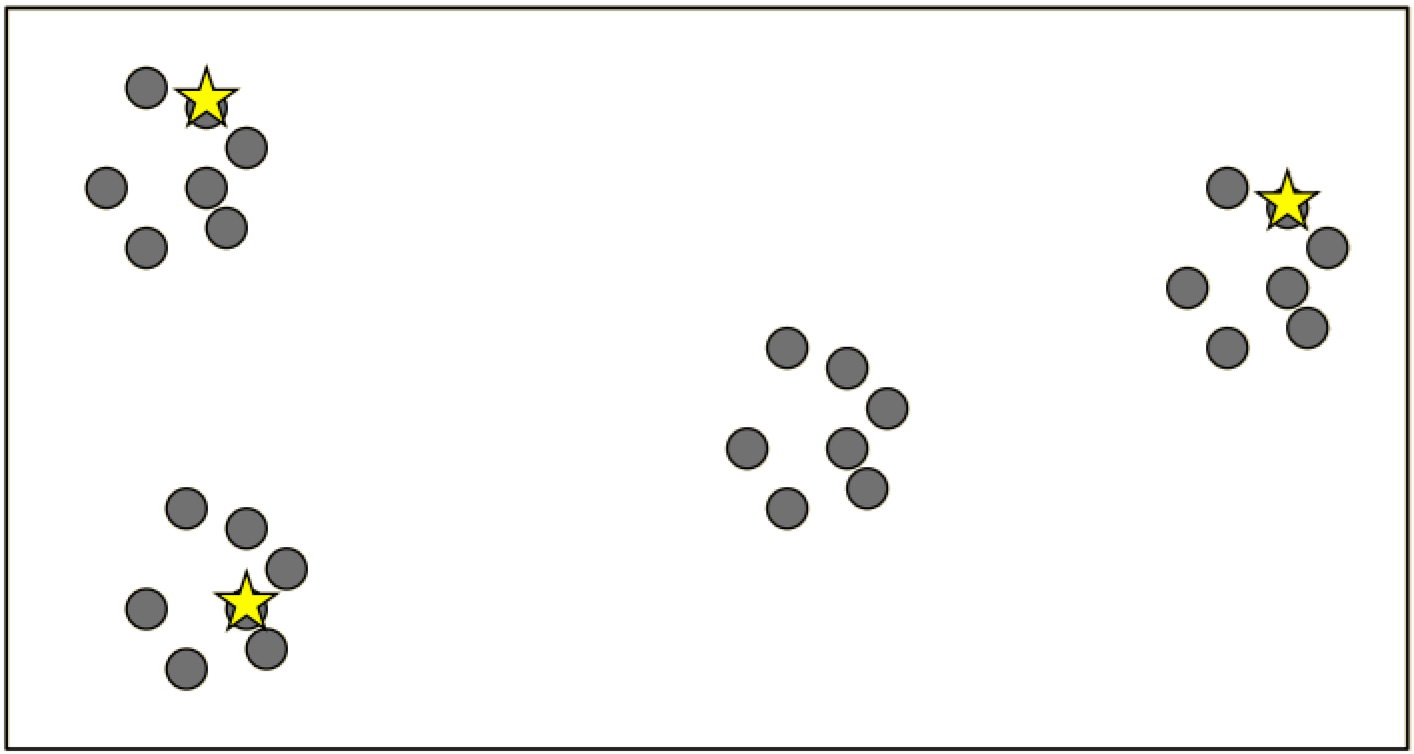
k-Means++



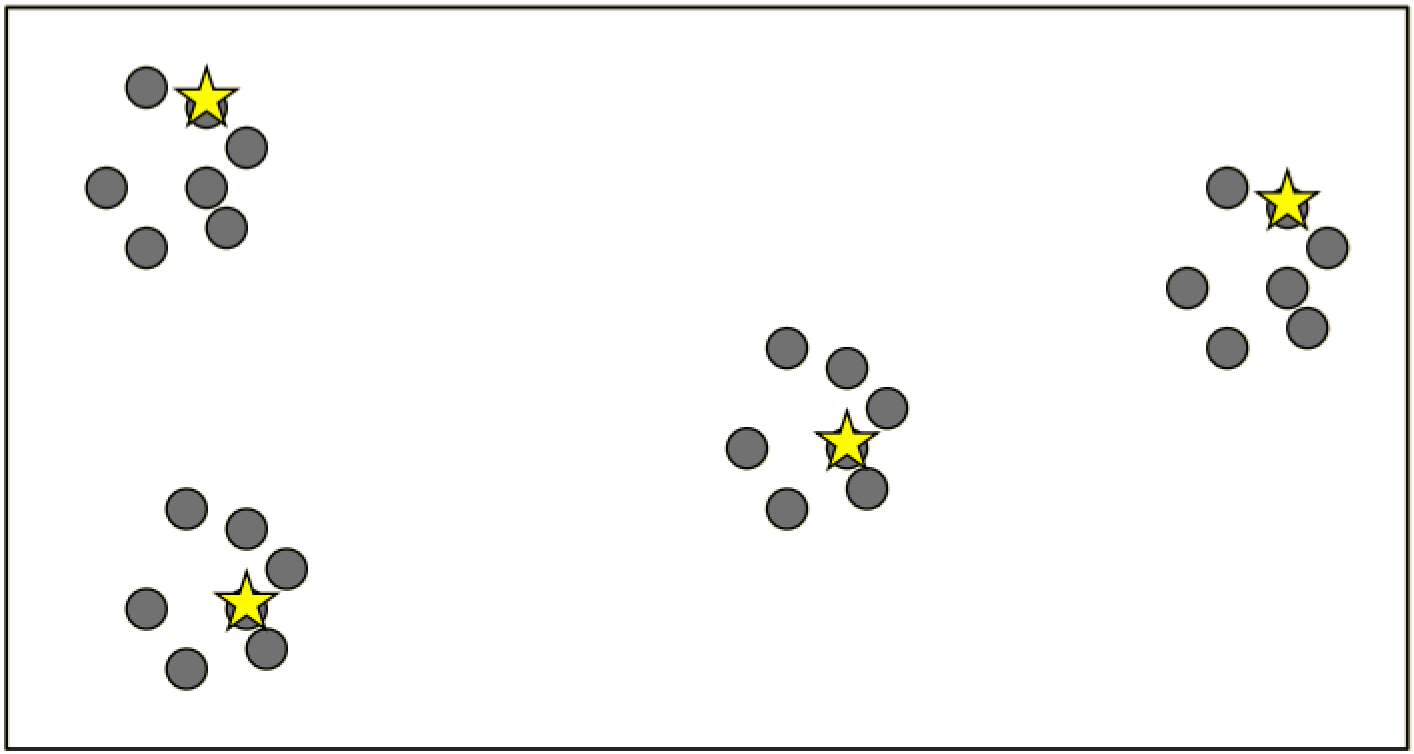
k-Means++



k-Means++



k-Means++



k-Means：*k*值的选择

- 主要是“问题驱动”（problem driven）
- 也可以是“数据驱动”（data driven），但条件如下：
  - 数据不稀疏
  - 输入的属性没有太多噪音

k-Means：*k*值的选择

- Most common measure is Sum of Squared Error (SSE)
  - For each point, the error is the distance to the nearest cluster
  - To get SSE, we square these errors and sum them.

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} dist^2(m_i, x)$$

  - x is a data point in cluster C_i and m_i is the representative point for cluster C_i
    - can show that m_i corresponds to the center (mean) of the cluster
  - Given two clusters, we can choose the one with the smallest error
  - One easy way to reduce SSE is to increase the number of clusters
    - A good clustering with smaller k can have a lower SSE than a poor clustering with higher k

k-Means：*k*值的选择

- **Elbow method:** plot a line chart of the SSE for each value of *k*. If the line chart looks like an arm, then the "elbow" on the arm is the value of *k* that is the best.

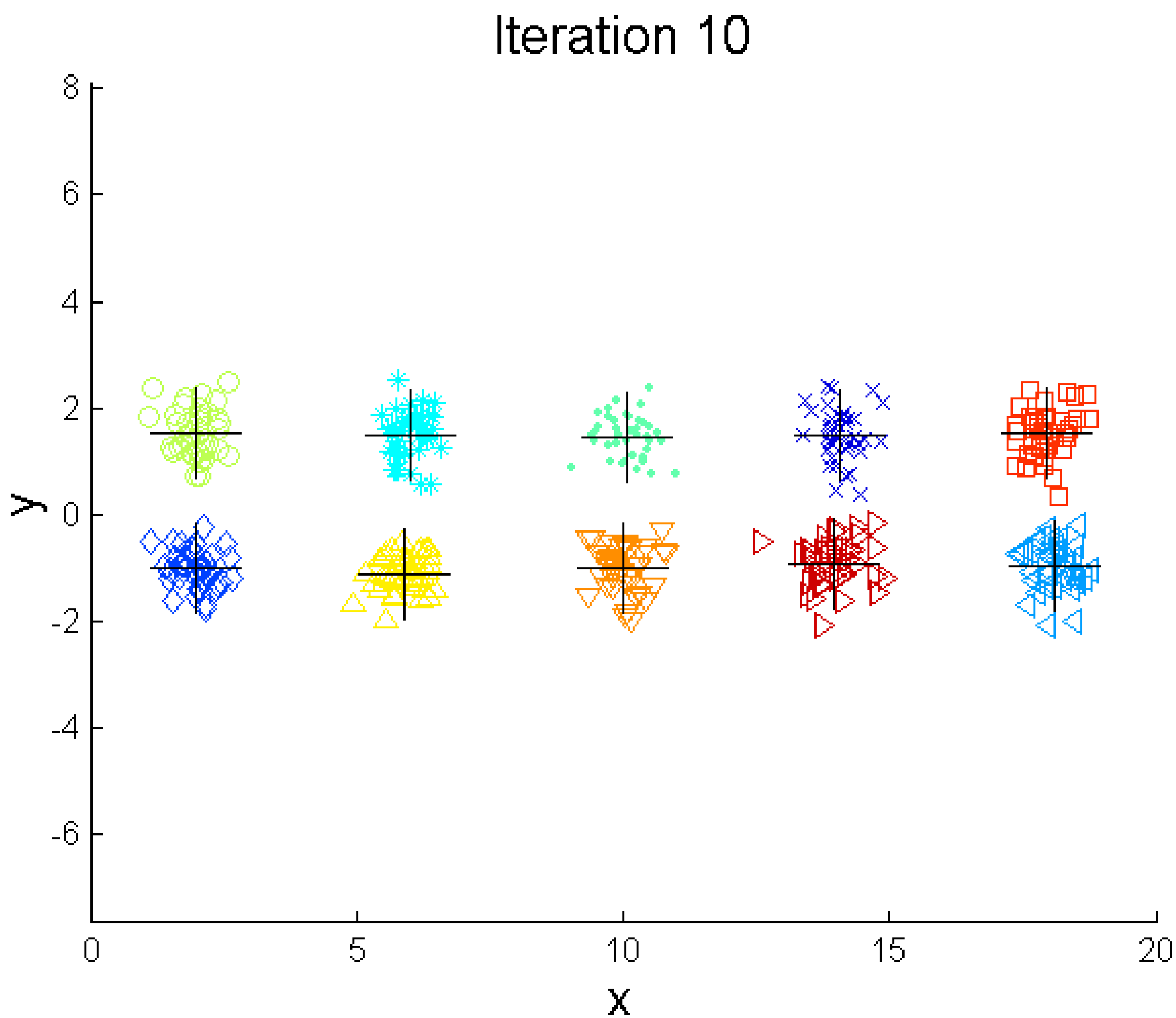
Bisecting *k*-Means

- Bisecting *k*-Means algorithm
  - Variant of *k*-Means that can produce a partitional or a hierarchical clustering

---

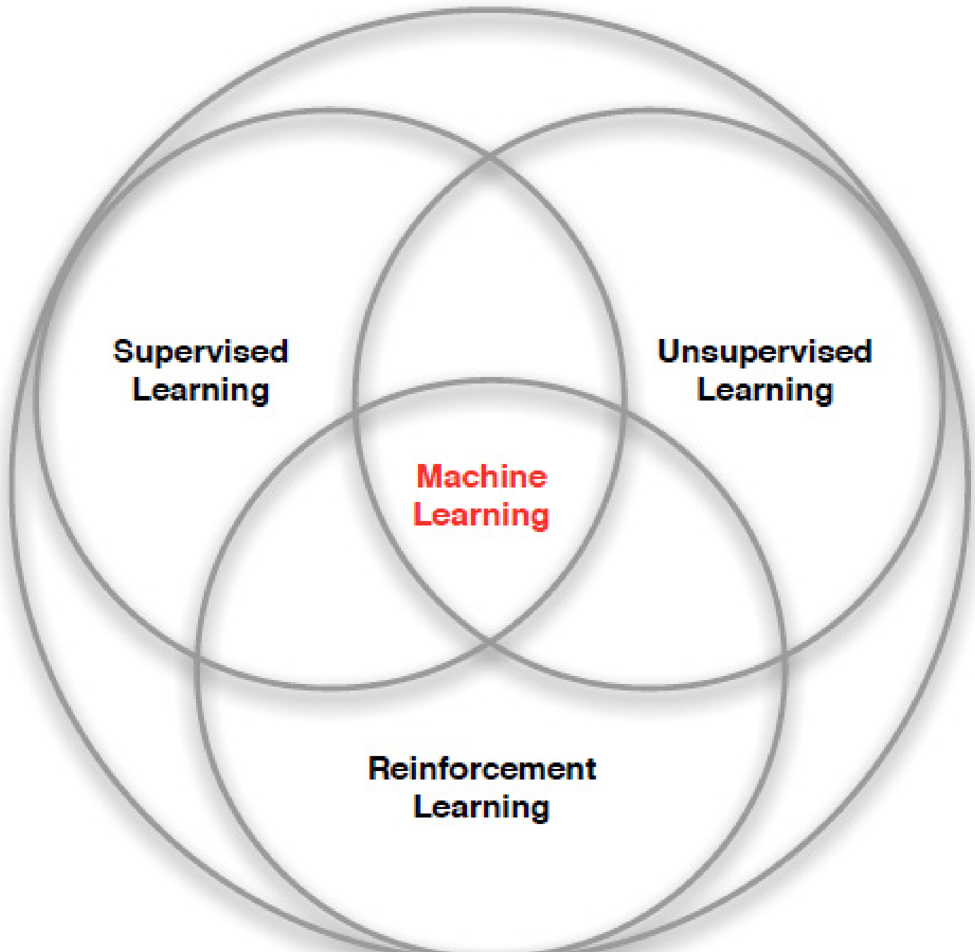
Initialize the list of clusters to contain the cluster containing all points.
repeat
    Select a cluster from the list of clusters
    for $i = 1$ to $number\_of\_iterations$ **do**
        Bisect the selected cluster using basic K-means
    end for
    Add the two clusters from the bisection with the lowest SSE to the list of clusters.
until Until the list of clusters contains K clusters

Bisecting *k*-Means



机器学习

- Three fundamental problems in machine learning



机器学习

- Three fundamental problems in machine learning
- Supervised learning
  - *Classification or prediction from labeled* (*action*, *outcome*) *pairs*
  - *No interactions*
  - *No sequential decisions*
  - *No explorations*
- Unsupervised learning
  - *Discover inherent correlations among data*
  - *No interactions*
  - *No sequential decisions*
  - *No explorations*

在很多应用场景中，有监督学习可能行不通。比如我们通过有监督学习来训练一个围棋模型，就需要将当前棋盘的状态作为输入，其对应的最佳落子位置（动作）作为标签。训练一个好的模型就需要收集大量的不同棋盘状态以及对应动作。这种做法实践起来比较困难，一是对于每一种棋盘状态，即使是专家也很难给出“正确”的动作，二是获取大量数据的成本往往比较高。

机器学习

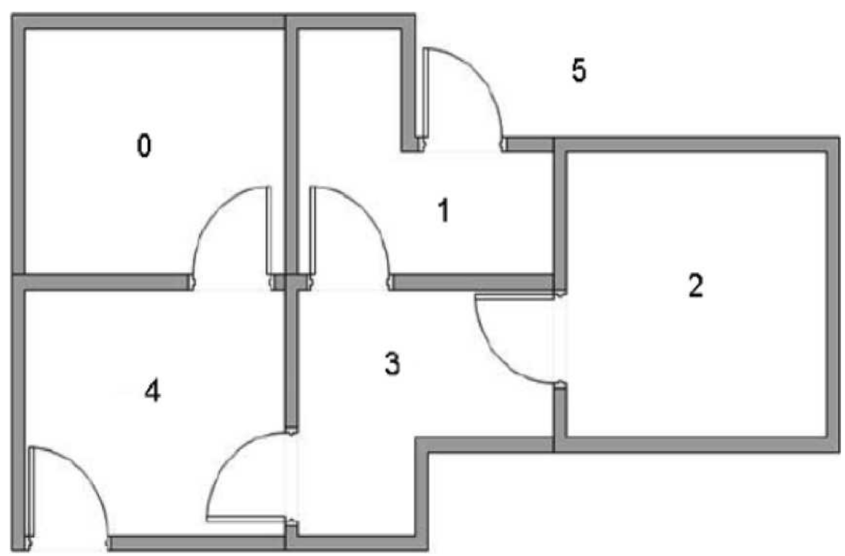
- Three fundamental problems in machine learning
- Reinforcement learning
  - *Reinforcement learning* (*RL*), *in a nutshell, is to "learn to make good sequences of decisions through trail-and-errors"*
- Thus, there are four basic aspects in RL:
  - *Optimization* (*good decisions*)
  - *Delayed consequences* (*sequential*)
  - *Exploration* (*trail-and-error*)
  - *Generalization* (*learn*)
- The evaluation of optimality can be explicitly measured or provided in terms of utility functions, e.g.,
  - *the shortest path between two cities given a network of roads*
  - *the fastest speed that a robot is able to run*
  - *the maximum area for a multi-robot system to cover*
  - *the least time for a group of vehicles to pass a crossroad*

强化学习

- 强化学习（Reinforcement learning）是一种序贯决策方法，它研究如何让计算机与环境交互，从中学会最优决策。智能体在和环境的交互过程中，根据当前的状态、环境的奖惩而采取相应的动作，通过学习使环境的回报最大化。
- 强化学习的基本要素：
  - 状态S、动作A
  - 奖励（reward）R：智能体动作好坏的评价
  - 策略（policy）$\pi(a|s)$：状态到动作的映射$P(A = a \mid S = s)$
  - 值函数（value function）：动作价值函数$Q^{\pi}(s, a)$和状态价值函数$V^{\pi}(s)$，分别是在状态s采取动作a后，执行策略π的期望回报；以及从状态s起，执行策略π的期望回报
- 强化学习的学习目标：找到最大化累计奖励的策略。

示例

- 假设建筑中有5个房间，编号为0-4，房间之间通过门相连（每个门都有两个方向），屋子外可视为一个大房间，编号为5。
- 将agent置于建筑中的任意一个房间，目标是走到房间5。每一条边关联一个reward值，直接连接到目标房间的门的reward值为100，其他门的reward值为0。



强化学习的基本要素：状态

- State is the information used to determine what happens next
- The environment state is its private representation
  - *whatever data to pick the next observation/reward*
  - *not usually visible to the agent*
  - *May contain irrelevant information*
- The agent state is the agent's internal representation
  - *whatever information the agent uses to pick the next action*
  - *it is the information used by RL algorithms*
- An Markov state contains all useful information from the history, i.e., future is independent of past given present

A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

强化学习的基本要素：奖励

- A reward R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximise cumulative reward
- The goal reward and the intermediate reward
  - defeat the world champion at Go
    +1/-1 reward for winning/losing a game
  - Make a humanoid robot walk
    +1 reward for forward motion
    -1 reward for falling over
  - Manage an investment portfolio
    +v reward for each $ in bank
- Reward is the most fundamental component in RL

强化学习的基本要素：策略

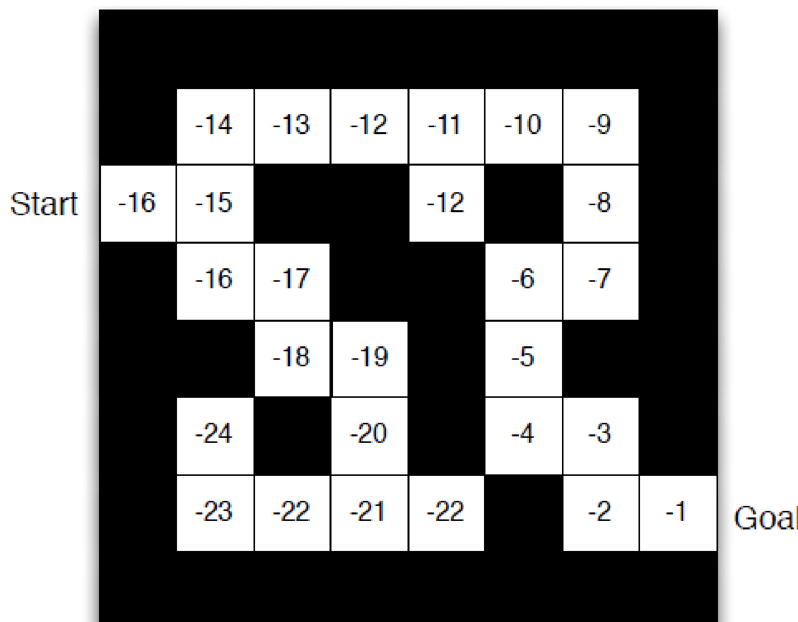
- Policy: an agent's behaviour function, i.e., a mapping from state to action
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

强化学习一般会使用随机性的策略。随机性的策略具有很多优点。比如在学习时可以通过引入一定的随机性来更好地探索环境，且使得策略更加多样性。以围棋游戏为例，确定性策略总是在同一个位置上下棋，这会导致你的策略很容易被对手预测。

强化学习的基本要素：值函数

- Value functions: how good is each state and/or action
  - *Value function is a prediction of future reward*
  - *Used to evaluate the goodness/badness of states*
  - *And therefore used to select between actions*

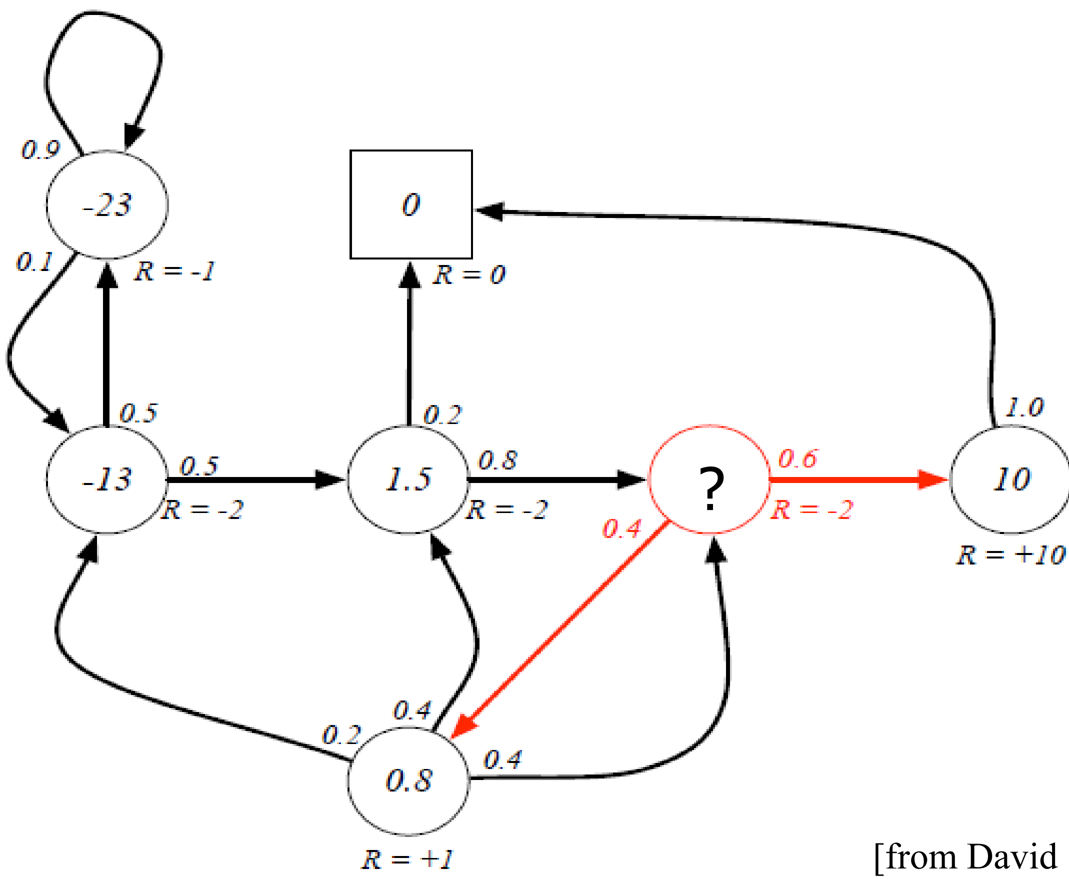
$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s\right]$$



[from David Silver's course]

强化学习的基本要素：值函数

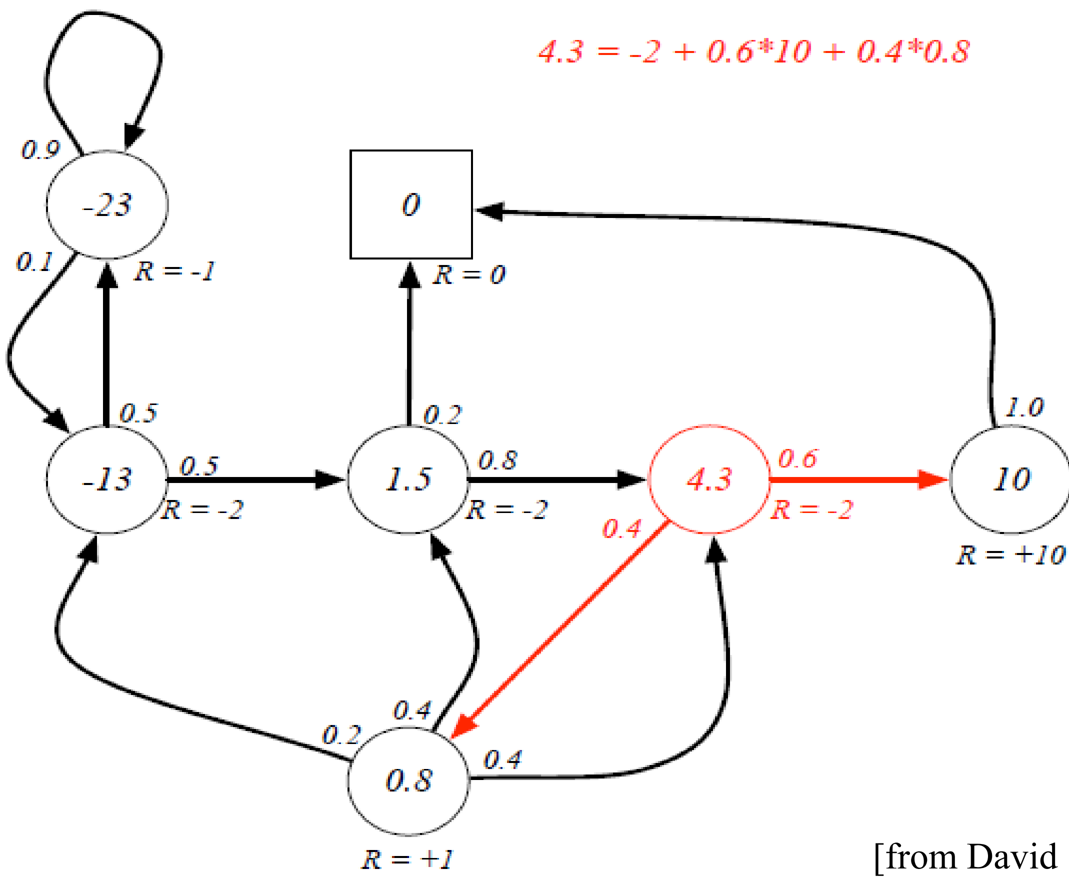
$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s)V(s')}_{\text{Discounted sum of future rewards}}$$



[from David Silver's course]

强化学习的基本要素：值函数

$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s)V(s')}_{\text{Discounted sum of future rewards}}$$

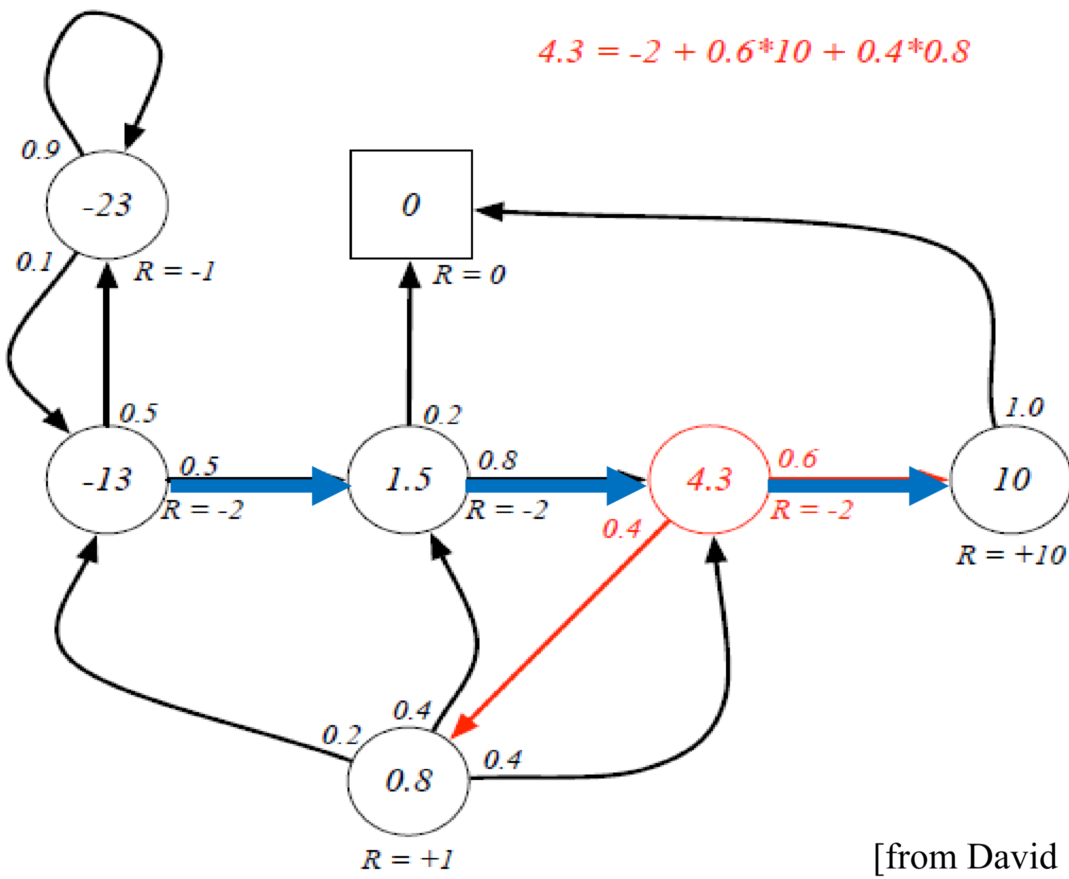


[from David Silver's course]

强化学习的基本要素：值函数

给定折扣因子（discount factor）$\gamma = 0.5$，蓝色轨迹的回报值如下：

$(-2) + 0.5 * (-2) + 0.5^2 * (-2) + 0.5^3 * 10 = -2.25$



[from David Silver's course]

形式化定义

单智能体强化学习问题的形式化定义：

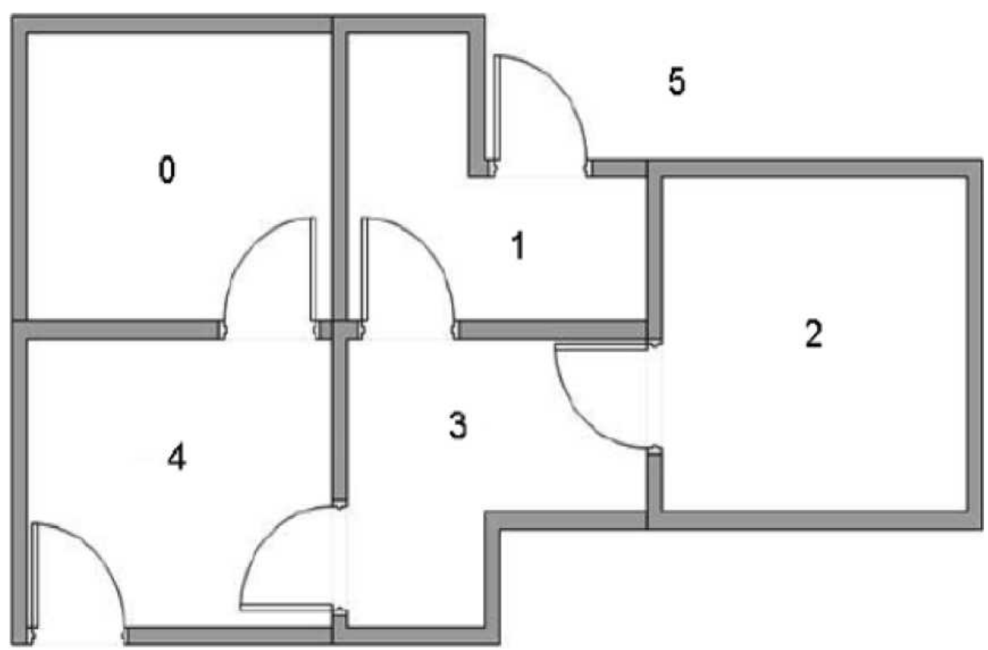
由六元组构成的马尔可夫决策过程，具体如下：

Markov Decision Process (MDP) $(S, A, R, T, P_0, \gamma)$

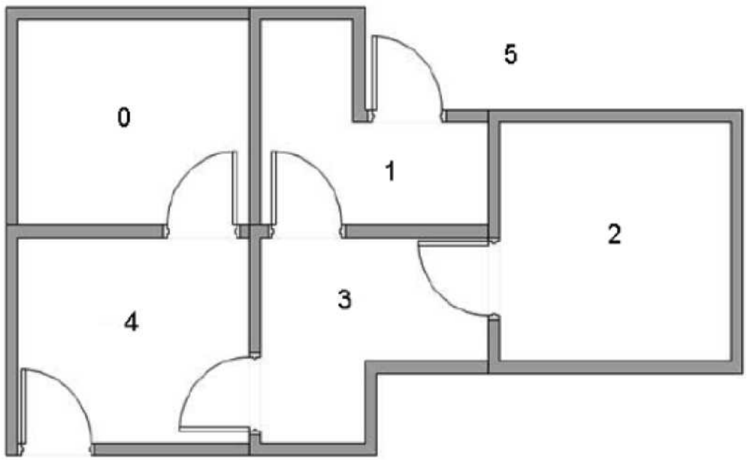
- S denotes the state space
- A is the action space
- $R = R(s, a)$ is the reward function
- $T: S \times A \times S \to [0,1]$ is the state transition function
- P_0 is the distribution of the initial state
- γ is a discount factor

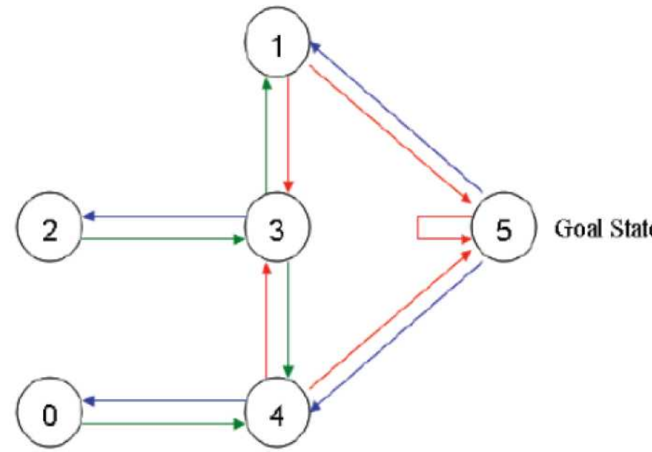
基于值函数的方法：Q-learning

- 先学习值函数，再基于值函数选择动作
- Q-learning：使用最大化Q值的动作来更新Q值
- 建筑中有5个房间，编号为0-4，房间之间通过门相连，屋子外被视为一个大房间，编号为5



基于值函数的方法：Q-learning

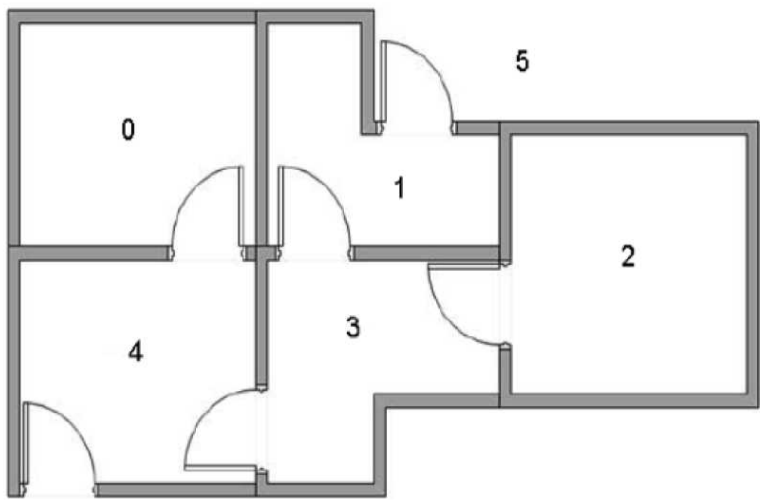


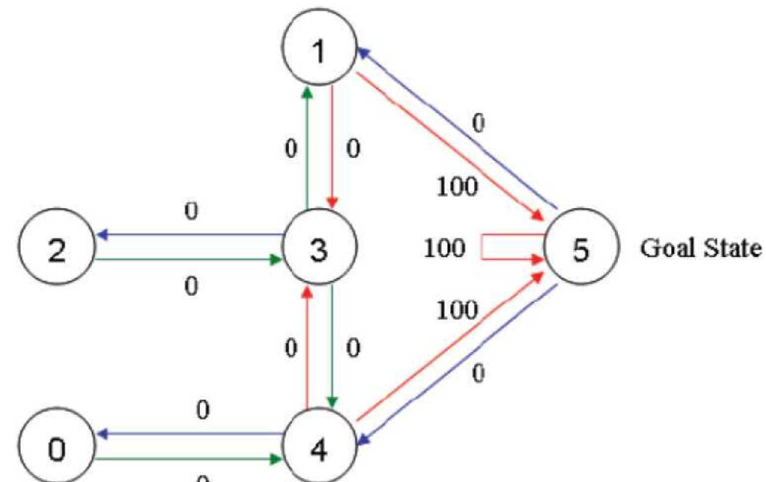


房间作为图的节点，两房间之间若有门相连则对应节点间的一条边（每个门都有两个方向）。

- ✓ State：房间（节点）
- ✓ Action：从一个房间走到另一个房间（箭头）

基于值函数的方法：Q-learning

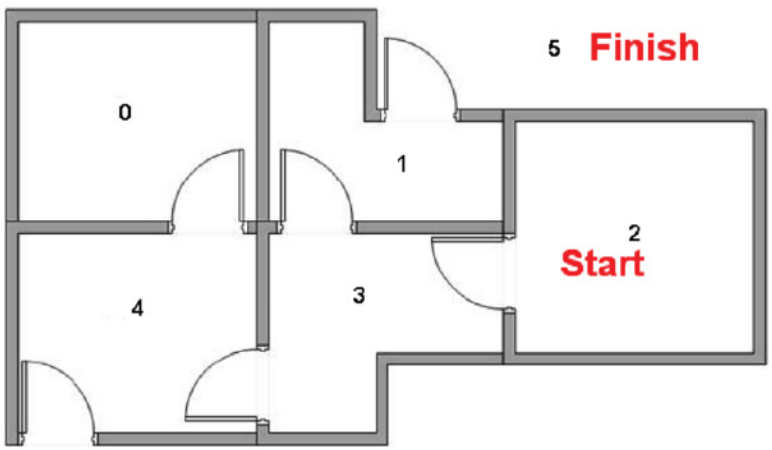


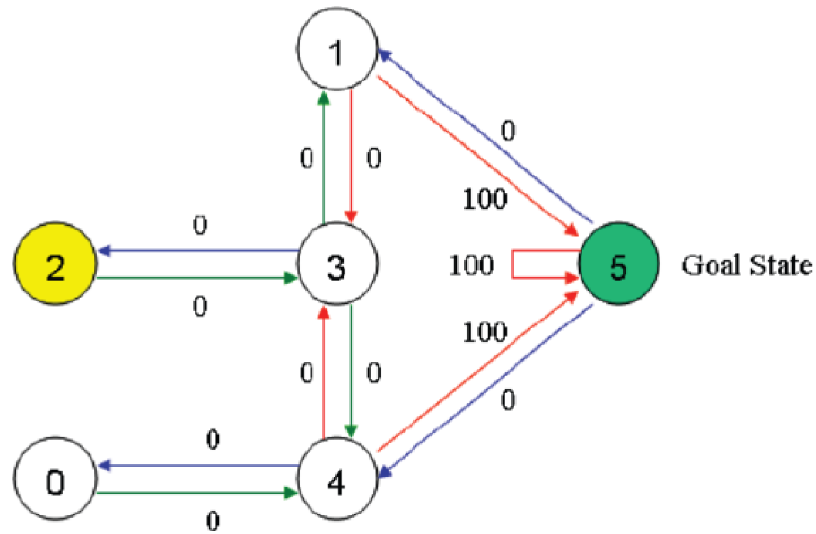


将agent置于建筑中的任意一个房间，目标是走到屋子外，即房间5。每一条边关联一个reward值，直接连接到目标房间的门的reward值为100，其他门的reward值为0。

房间5有一个指向自己的箭头，reward值也为100。

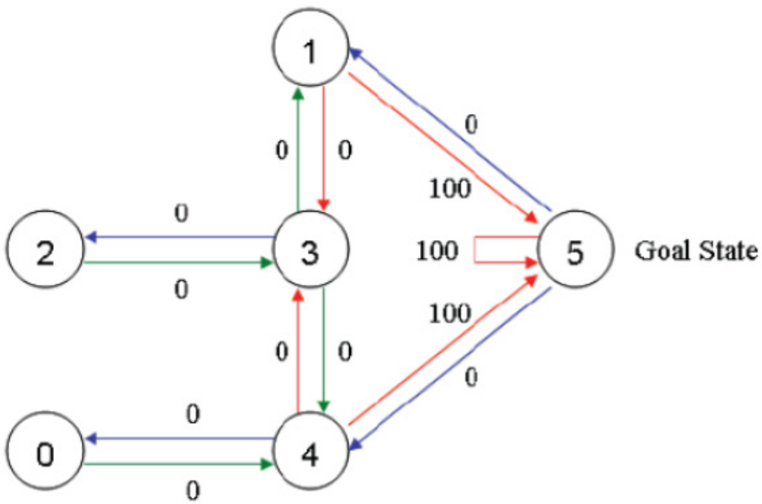
基于值函数的方法：Q-learning





假设agent从状态2开始，我们希望通过学习到达状态5。

基于值函数的方法：*Q*-learning



$$R=\begin{array}{c c} & \text{Action} \\ \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0\\1\\2\\3\\4\\5 \end{array} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

以state为行，action为列构建reward值矩阵*R*，其中-1表示空值，即节点间没有边相连。

基于值函数的方法：*Q*-learning

- ✓ 类似地，构建一个与R同阶的矩阵Q，表示agent已经从经验中学到的知识。由于agent刚开始对外界环境一无所知，Q初始化为零矩阵。
- ✓ 本例中的状态数目是已知的（等于6），对于状态数目未知的情形，可以让Q从一个元素出发，每发现一个新的状态就增加相应的行列。
- ✓ Q学习算法的状态转移规则：

$$Q(s,a) = R(s,a) + \gamma \cdot \max_{\tilde{a}}\{Q(\tilde{s},\tilde{a})\} \quad (1.1)$$

其中s, a表示当前的状态和动作，$\tilde{s},\tilde{a}$ 表示s的下一个状态及动作。学习参数γ为满足 $0 \le \gamma < 1$ 的常数。

基于值函数的方法：Q-learning

Step 1 给定参数 γ 和 *reward* 矩阵 R.

Step 2 令 $Q := \mathbf{0}$.

Step 3 *For each episode:*

3.1 随机选择一个初始的状态 s.

3.2 若未达到目标状态，则执行以下几步

(1) 在当前状态 s 的所有可能行为中选取一个行为 a.

(2) 利用选定的行为 a，得到下一个状态 $\widetilde{s}$.

(3) 按照 *(1.1)* 计算 $Q(s, a)$.

(4) 令 $s := \widetilde{s}$.

agent利用该算法从经验中学习，每一个episode相当于一个training epoch。agent不断探索外界环境，并接收外界环境的reward，直至达到目标状态。训练得越多，*Q*被优化得更好，agent就能根据训练后的*Q*更容易地找到到达目标状态的最快路径。

基于值函数的方法：*Q*-learning

得到充分训练的Q之后：

1. 令当前状态 $s := s_0$.

2. 确定 a, 它满足 $Q(s, a) = \max_{\widetilde{a}}\{Q(s, \widetilde{a})\}$.

3. 令当前状态 $s := \widetilde{s}$ ($\widetilde{s}$ 表示 a 对应的下一个状态).

4. 重复执行步 2 和步 3 直到 s 成为目标状态.

基于值函数的方法：*Q*-learning

设学习参数为0.8，初始状态为房间1，Q初始化为0矩阵：

$$Q=\begin{array}{c}\\0\\1\\2\\3\\4\\5\end{array}\begin{array}{c}\begin{array}{cccccc}0&1&2&3&4&5\end{array}\\\begin{bmatrix}0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{bmatrix}\end{array}$$

$$\begin{array}{c}\text{State}\\ \\ \end{array}\quad\begin{array}{c}\text{Action}\end{array}$$

$$R=\begin{array}{c}\\0\\1\\2\\3\\4\\5\end{array}\begin{array}{c}\begin{array}{cccccc}0&1&2&3&4&5\end{array}\\\begin{bmatrix}-1&-1&-1&-1&0&-1\\-1&-1&-1&0&-1&100\\-1&-1&-1&0&-1&-1\\-1&0&0&-1&0&-1\\0&-1&-1&0&-1&100\\-1&0&-1&-1&0&100\end{bmatrix}\end{array}$$

若agent随机地转移到状态5，更新Q矩阵，得到一次episode后的Q矩阵：

$$\begin{aligned}Q(1,5)&=R(1,5)+0.8*\max\{Q(5,1),Q(5,4),Q(5,5)\}\\&=100+0.8*\max\{0,0,0\}\\&=100.\end{aligned}$$

$$Q=\begin{array}{c}\\0\\1\\2\\3\\4\\5\end{array}\begin{array}{c}\begin{array}{cccccc}0&1&2&3&4&5\end{array}\\\begin{bmatrix}0&0&0&0&0&0\\0&0&0&0&0&100\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{bmatrix}\end{array}$$

基于值函数的方法：Q-learning

第二次episode：随机选择一个初始状态，此处选状态3。让agent执行走到状态1的action，更新Q：

$$
\begin{aligned}
Q(3,1) &= R(3,1) + 0.8 * \max\{Q(1,3), Q(1,5)\} \\
&= 0 + 0.8 * \max\{0, 100\} \\
&= 80.
\end{aligned}
$$

$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 80 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

现在状态1变成了当前状态，因为状态1还不是目标状态，仍需继续探索，故随机选择可能的action。假定agent选择了走到状态5的action，更新Q：

$$
\begin{aligned}
Q(1,5) &= R(1,5) + 0.8 * \max\{Q(5,1), Q(5,4), Q(5,5)\} \\
&= 100 + 0.8 * \max\{0, 0, 0\} \\
&= 100.
\end{aligned}
$$

此处更新并没有引起矩阵Q的变化，Q保持不变。

基于值函数的方法：Q-learning

执行更多的episode，矩阵Q最终收敛为：

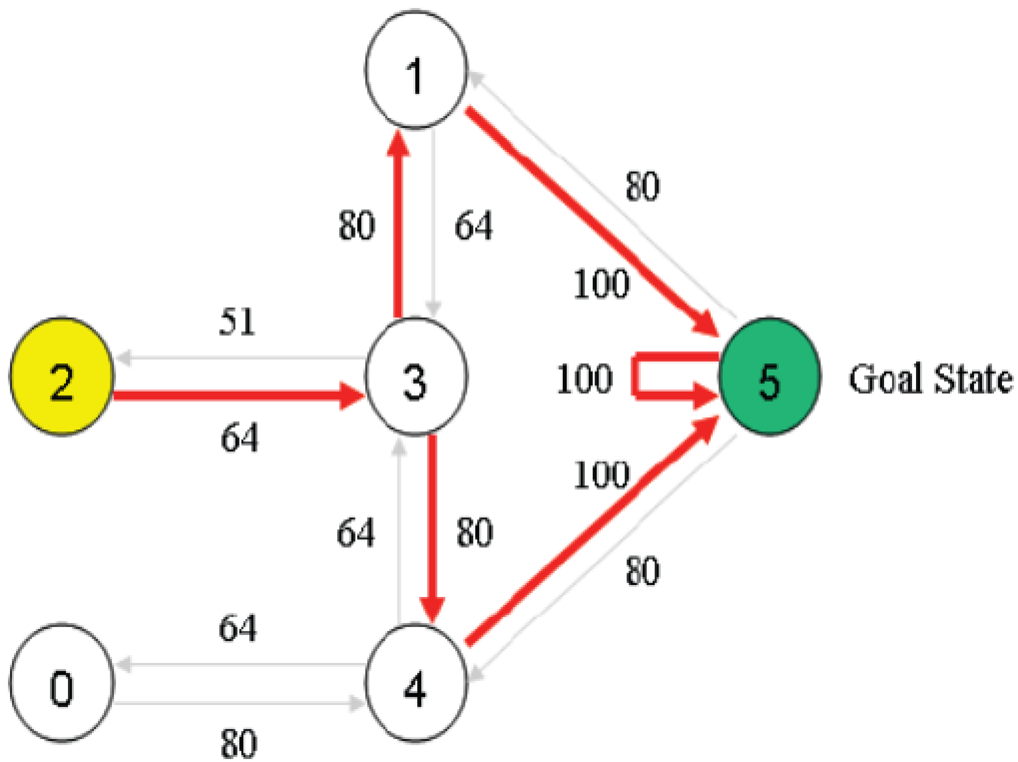
$$
Q=\begin{array}{c}
 \\ 0\\1\\2\\3\\4\\5
\end{array}
\begin{array}{c}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array}\\
\begin{bmatrix}
0 & 0 & 0 & 0 & 400 & 0\\
0 & 0 & 0 & 320 & 0 & 500\\
0 & 0 & 0 & 320 & 0 & 0\\
0 & 400 & 256 & 0 & 400 & 0\\
320 & 0 & 0 & 320 & 0 & 500\\
0 & 400 & 0 & 0 & 400 & 500
\end{bmatrix}
\end{array}
$$

每个元素都除以5，得到：

$$
Q=\begin{array}{c}
 \\ 0\\1\\2\\3\\4\\5
\end{array}
\begin{array}{c}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array}\\
\begin{bmatrix}
0 & 0 & 0 & 0 & 80 & 0\\
0 & 0 & 0 & 64 & 0 & 100\\
0 & 0 & 0 & 64 & 0 & 0\\
0 & 80 & 51 & 0 & 80 & 0\\
64 & 0 & 0 & 64 & 0 & 100\\
0 & 80 & 0 & 0 & 80 & 100
\end{bmatrix}
\end{array}
$$

基于值函数的方法：*Q*-learning

当矩阵*Q*接近收敛状态，agent便学习到了转移至目标状态的最佳路径：



基于值函数的方法：*Q*-learning

- Suppose the agent has an experience $\langle s, a, r, s' \rangle$
- This provides one piece of data to update $Q[s, a]$.
- An experience $\langle s, a, r, s' \rangle$ provides a new estimate for the value of $Q^*(s, a)$:

$$r + \gamma \max_{a'} Q[s', a']$$

which can be used in the TD formula giving:

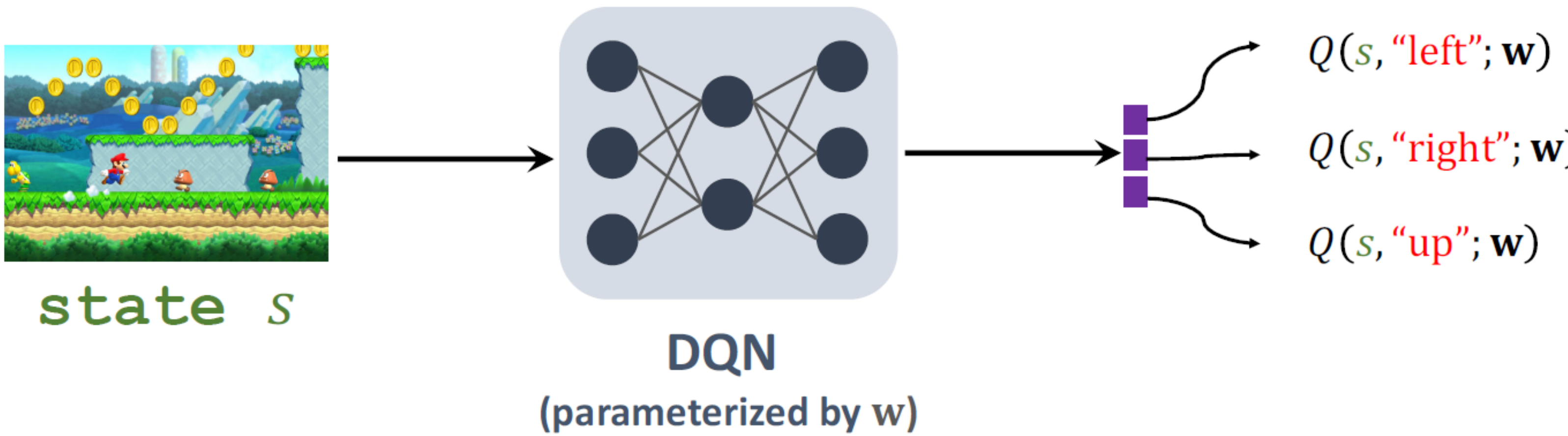
$$Q[s, a] \leftarrow Q[s, a] + \alpha \left(r + \gamma \max_{a'} Q[s', a'] - Q[s, a] \right)$$

基于值函数的方法：*Q*-learning

initialize $Q[S, A]$ arbitrarily
observe current state s
repeat forever:
  select and carry out an action a
  observe reward r and state s'
  $Q[s, a] \leftarrow Q[s, a] + \alpha \left(r + \gamma \max_{a'} Q[s', a'] - Q[s, a] \right)$
  $s \leftarrow s'$

Deep Q Network (DQN)

Approximate the optimal action-value function, $Q^\star(s,a)$, by $Q(s,a;\mathbf{w})$.



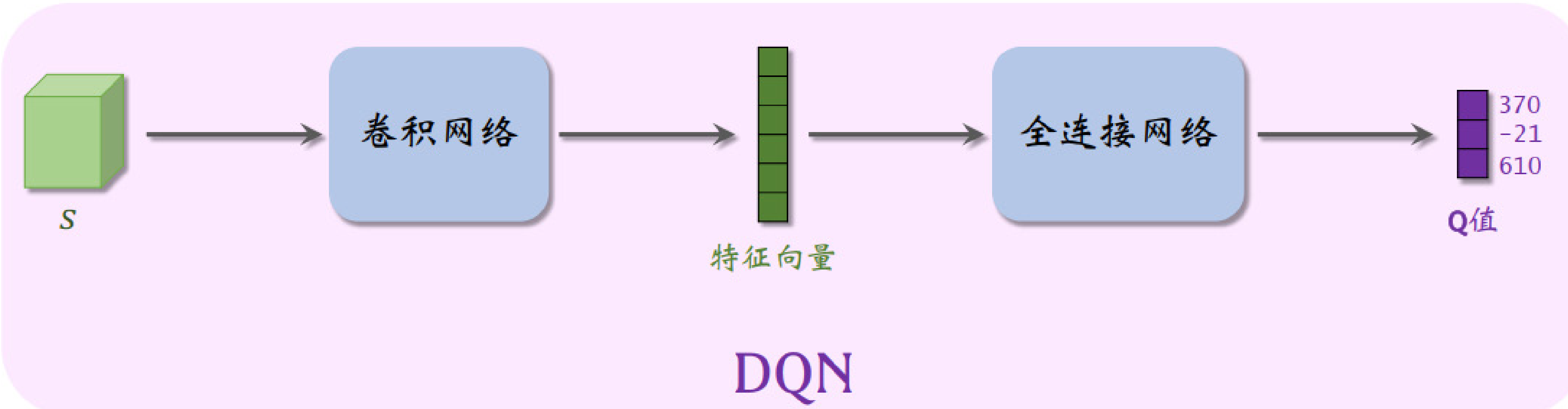
$$\underbrace{(\widehat{q}_t - \widehat{y}_t)}_{\text{TD 误差 } \delta_t}$$

$$\underbrace{Q(s_t, a_t;\, \boldsymbol{w})}_{\text{预测 } \widehat{q}_t}$$

$$\underbrace{r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q(s_{t+1}, a;\, \boldsymbol{w})}_{\text{TD 目标 } \widehat{y}_t}$$

$$\boldsymbol{w} \;\leftarrow\; \boldsymbol{w} - \alpha \cdot \delta_t \cdot \nabla_{\boldsymbol{w}} Q(s_t, a_t;\, \boldsymbol{w}).$$

Deep Q Network (DQN)



- 以CNN为例，假设输入图像（状态*s*）的维度是：3 * 6 * 6，其中3为输入单元通道数，图片的高和宽均为6；
- 假设卷积核大小为3，步长为1，填充大小为0（即无填充），输出单元通道数为2，经过一层卷积操作后，输出图像的维度是：2 * 4 * 4（Why?），总的卷积核参数数量为：3 * 2 * 3 * 3 = 54；

$$\lfloor 1 + [\,\text{高（或宽）} + 2 * \text{填充大小} - \text{卷积核大小}\,] / \text{步长} \rfloor$$

$$\lfloor 1 + (6 + 2 * 0 - 3) / 1 \rfloor = 4$$

- 假设池化窗口的大小为2（即池化层的卷积核大小和步长均为2），经过一层池化操作后，输出图像的维度是：2 * 2 * 2。

基于策略的方法

- 学习一个参数化的策略$\boldsymbol{\pi}(a|s)$，不需要基于值函数选择动作。
- 使用策略网络$\boldsymbol{\pi}(a|s;\boldsymbol{\theta})$来近似$\boldsymbol{\pi}(a|s)$，其中$\boldsymbol{\theta}$为策略网络的可训练参数。

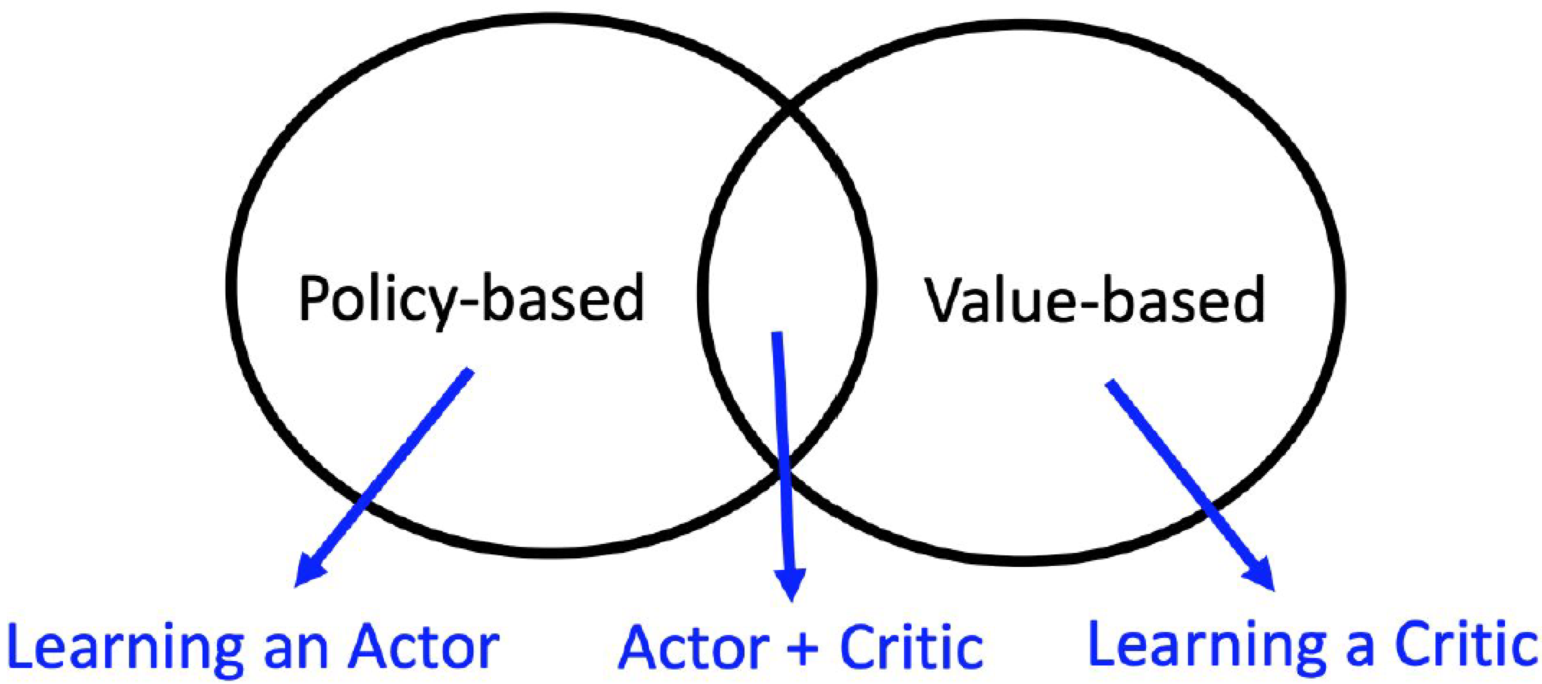
Learn $\boldsymbol{\theta}$ that maximizes $J(\boldsymbol{\theta}) = \mathbb{E}_S[V(S;\boldsymbol{\theta})]$

- 基于策略的方法主要包括随机性策略梯度和确定性策略梯度等。例如，随机性策略梯度方法基于$J(\boldsymbol{\theta})$的梯度来学习$\boldsymbol{\theta}$：

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$$

Actor-Critic算法

Actor-Critic（演员-评委）：基于值函数和基于策略的融合算法。其中，策略π控制智能体，因此被看作"演员"；而Q^π评价π的表现，帮助改进π，因此Q^π被看作"评委"。



异策略与同策略学习

- 在强化学习中，我们让智能体与环境交互，记录下观测到的状态、动作、奖励，用这些经验来学习一个策略函数。在这一过程中，控制智能体与环境交互的策略被称作**行为策略** μ。行为策略的作用是收集经验(Experience)，即观测的环境、动作、奖励。
- 训练的目的是得到一个**目标策略**函数 π，在结束训练之后，用这个策略函数来控制智能体。
- **Off-policy（异策略） learning**
  - Learn about policy π from experience sampled from μ
- **On-policy（同策略） learning**
  - Learn about policy π from experience sampled from π

异策略学习

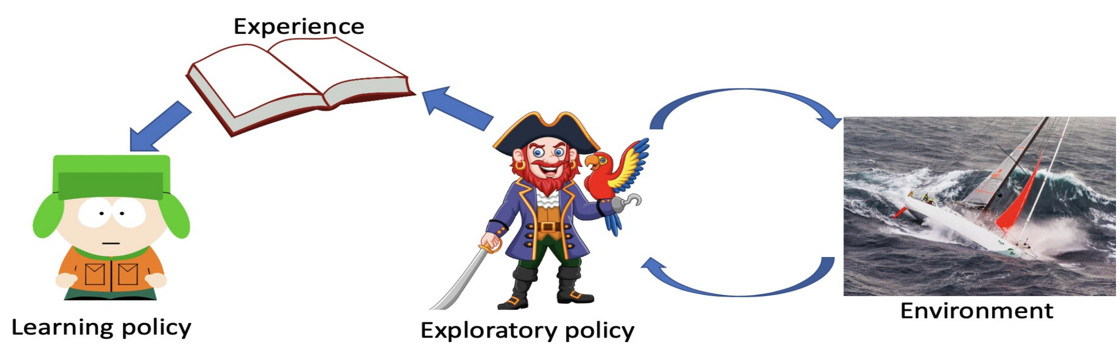
- 通过行为策略$\mu(a|s)$来收集经验数据：

$$S_1, A_1, R_2, ..., S_T \sim \mu$$
$$\text{Update } \pi \text{ using } S_1, A_1, R_2, ..., S_T$$

- 最常用的行为策略是ϵ-*greedy*:

$$a_t = \begin{cases} \text{argmax}_a Q(s_t, a; \boldsymbol{w}), & \text{以概率 } (1-\epsilon); \\ \text{均匀抽取 } \mathcal{A} \text{ 中的一个动作}, & \text{以概率 } \epsilon. \end{cases}$$

- 优势：
  - Learn about optimal policy while following exploratory policy
  - Learn from observing humans or other agents
  - Re-use experience generated from old policies $\pi_1, \pi_2, \dots, \pi_{t-1}$



Off-policy control with *Q*-learning

- We allow both behavior and target policies to improve
- The target police π is greedy on $Q(s, a)$

$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a')$$

- The behavior policy μ could be totally random, but we let it improve following *ϵ-greedy* on $Q(s, a)$
- Thus *Q*-learning target

$$\begin{aligned} R_{t+1} + \gamma Q(S_{t+1}, A') &= R_{t+1} + \gamma Q(S_{t+1}, \arg\max Q(S_{t+1}, a')) \\ &= R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') \end{aligned}$$

- Thus the *Q*-learning update

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

Q-learning和Sarsa的对比

☐ *Q*-learning: Off-Policy

Choose action A_t from S_t using policy derived from Q with ϵ-*greedy*
Take action A_t, observe R_{t+1} and S_{t+1}
Then 'imagine' A_{t+1} as argmax $Q(S_{t+1}, a')$ in the update target

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

☐ State-Action-Reward-State-Action (Sarsa): On-Policy

Choose action A_t from S_t using policy derived from Q with ϵ-*greedy*
Take action A_t, observe R_{t+1} and S_{t+1}
Choose action A_{t+1} from S_{t+1} using policy derived from Q with ϵ-*greedy*

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

强化学习参考资料

- 王树森、张志华. 深度强化学习（初稿）. 北京大学, 2021.
- 邱锡鹏. 神经网络与深度学习. 复旦大学, 2019.
- Reinforcement Learning: An Introduction (Second Edition), Richard S. Sutton and Andrew G. Barto, MIT Press, Cambridge, MA, 2018.
- Reinforcement Learning: State-of-the-Art, Wiering M.A., Springer, 2016.
- Algorithms for Reinforcement Learning, Csaba Szepesvári, Morgan & Claypool Publishers, 2010.
- https://github.com/wangshusen/DRL
- https://github.com/zhoubolei/introRL