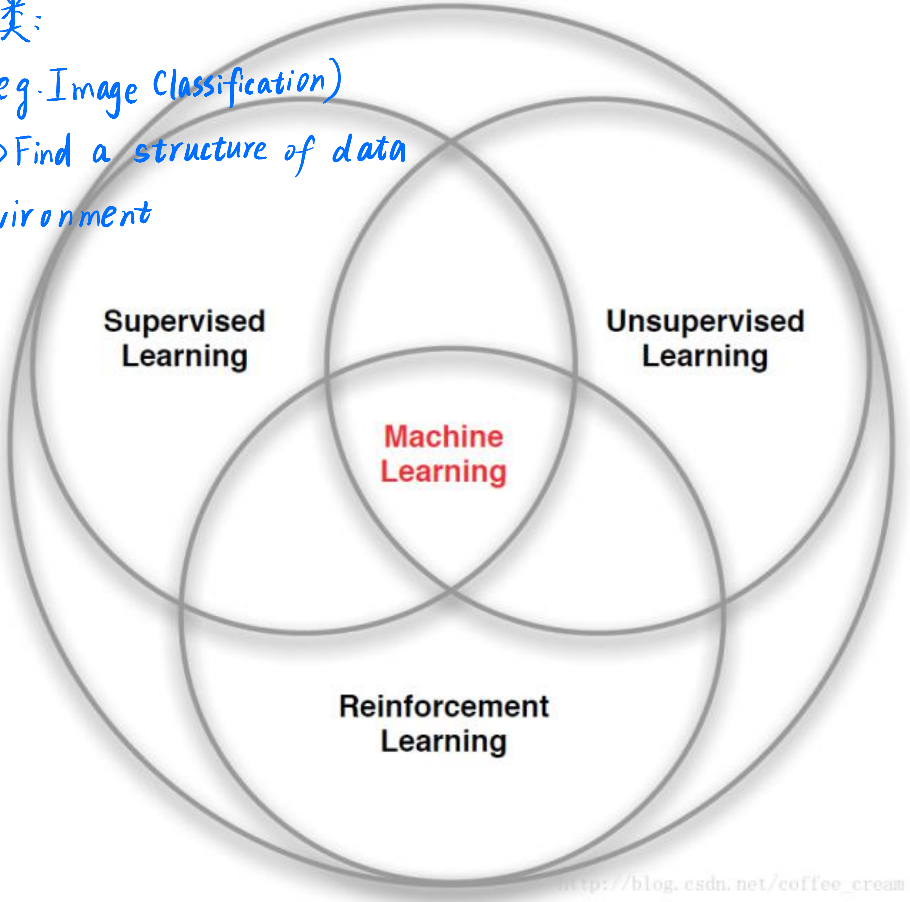
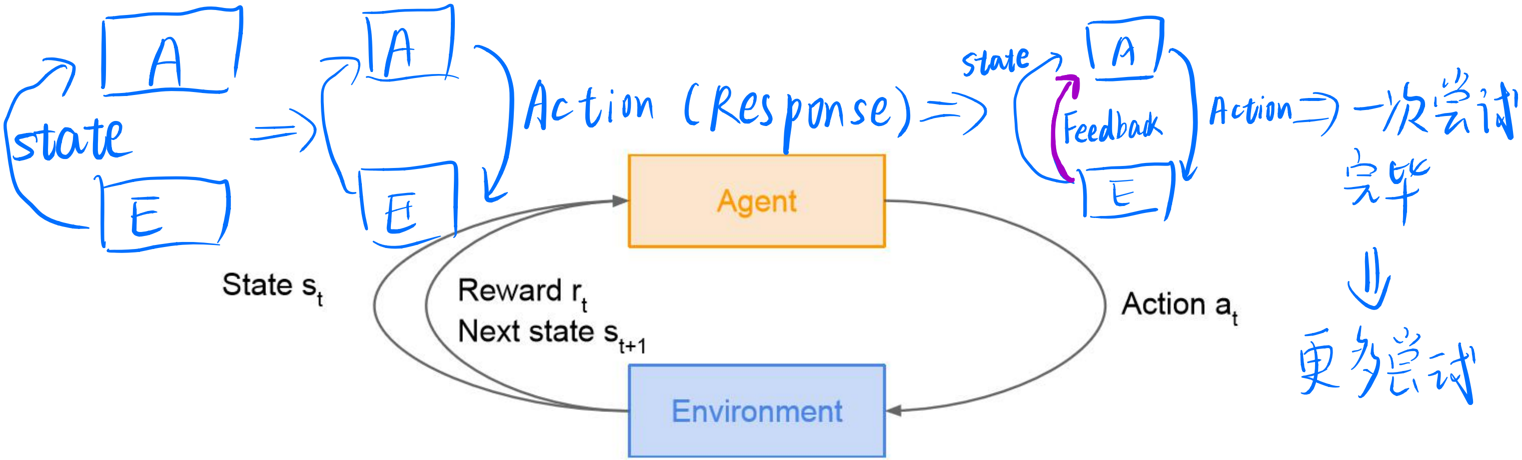
Reinforcement Learning

机器学习大致上分三类:

1. 有监督学习: X → Y (eg. Image Classification)
2. 无监督学习: Only X ⇒ Find a structure of data
3. 强化学习: Agent & Environment



Typical Scenario：



Markov Decision Process

- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

S:状态集 A:动作响应集 R:反馈Return (Reward或Penalty)

$\mathbb{P}$:转移概率 γ:衰减率[为什么要引入呢?
1.直观上:现在(短时间)+未来(长时间)
2.数学上:易于计算,将反馈有界化,并且更具过程性]

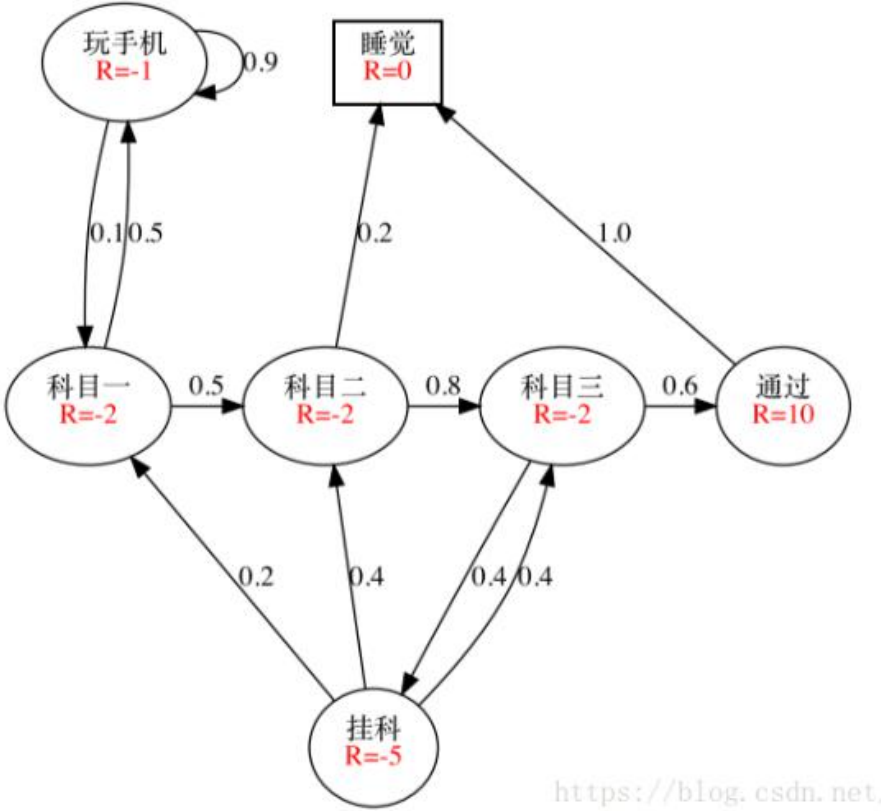
$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(. | s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. | s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

- A policy **π** is a function from S to A that specifies what action to take in each state
- **Objective**: find policy **π*** that maximizes cumulative discounted reward: $\sum_{t\geq0} \gamma^t r_t$

A Simple Illustration Example about MDP:

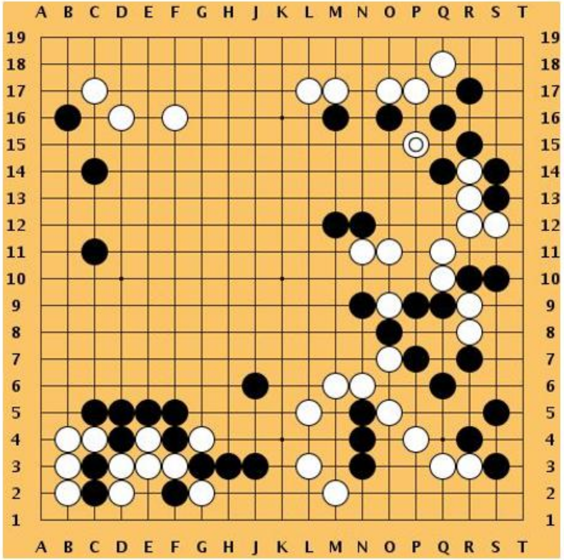
一个简例:



Applications: 在AlphaGo，Atari Game（常用于评估一个RL模型的效果）

小游戏（Flappy Bird）中，都有广泛应用

Go



Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

Relative or Advanced Algorithms:

Basic learning:

- Policy Gradient
- Q Learning

Advanced learning:

表 5 典型的深度强化学习算法特点及性能比较

Table 5 Characteristic and performance comparison of classical deep reinforcement learning algorithms

算法	算法特点	Atari游戏表现
DQN	经验回放技术, 异步更新目标网络	100%(DQN表现作为基准)
Dueling DQN	竞争型网络结构, 提升网络更新效率	151.72%
A3C	异步多线程优势函数作用网络更新	163.07%
TRPO	理论保证单调提升, 但训练耗时较长	实验游戏数量较少, 且表现性能较差
ACKTR	使用K-FAC因式分解, 降低梯度计算复杂度, 提升算法样本利用率	353.87%
PPO	具有TRPO算法的稳定性和可靠性, 算法复杂度较低	46.26%