深度强化学习

未来新机器学习范式

杨文昊

目前研究方向有视频监控下的异常检测、Model-based 强化学习

2020 年 11 月 17 日

AlphaGo

- 我想这应该是让人工智能迎来转折点的一项应用，也让 AI 引起了全世界范围的关注。而 AlphaGo 的核心算法正是深度强化学习 (Deep Reinforcement Learning)，到今天为止，DRL 的恐怖功能还未被全部开发出来。
- 训练 AlphaGo 的网络花费了 1920 CPUs 和 280 GPUs，还有 $3000 电费。

图: 人工智能 AlphaGo

什么是强化学习?

监督学习与无监督学习

从本质而言，监督学习 (supervised learning) 和无监督学习 (unsupervised learning) 的最大区别为数据样本是否含有标签。监督学习是指在数据样本有标签的情况，让机器去学习数据特征与标签之间的关系，最简单的监督学习例如线性回归与逻辑回归，无监督学习则是指在数据无标签的情况下，让机器去学习数据本身之间的关系，例如 K-means 和 PCA，并最终都以参数的形式表现出来。除此之外还有工程上常用的半监督学习 (semi-supervised learning) 和弱监督学习 (weakly supervised learning)。

强化学习

强化学习则本质上是在执行一种搜索的任务，通过机器去搜索某种特定环境中的最优策略，就是强化学习 (自己总结，不来自百度)。强化学习和标签学习的本质任务不同，强化学习拓展了机器学习的边界，让机器学习从预测任务转变了决策任务。

强化学习的特点 (Different from online learning)

- 无监督者，只有奖励机制 (Reward)
- 收益回报等反馈是延迟的，不是即时的
- 智能体 (Agent) 的动作决策影响着环境，如果环境因为它的决策变差了，它会因此背锅
- 目前强化学习的应用可以主要分为游戏 (Game theory)、机器 (robot, helicopter, automonous driving)、经济 (博弈论) 等等，这节课我们将演示通过深度强化学习的方法去训练一只能飞很远的 Flappy Bird

Agent & Environment

- Agent & Enviroment
- Action A_t & Reward R_t & State S_t
- Policy $\pi(a|s)$ & Value Function $v_\pi(s)$ &Model $\mathcal{P}$ $\mathcal{R}$ inside a RL agent

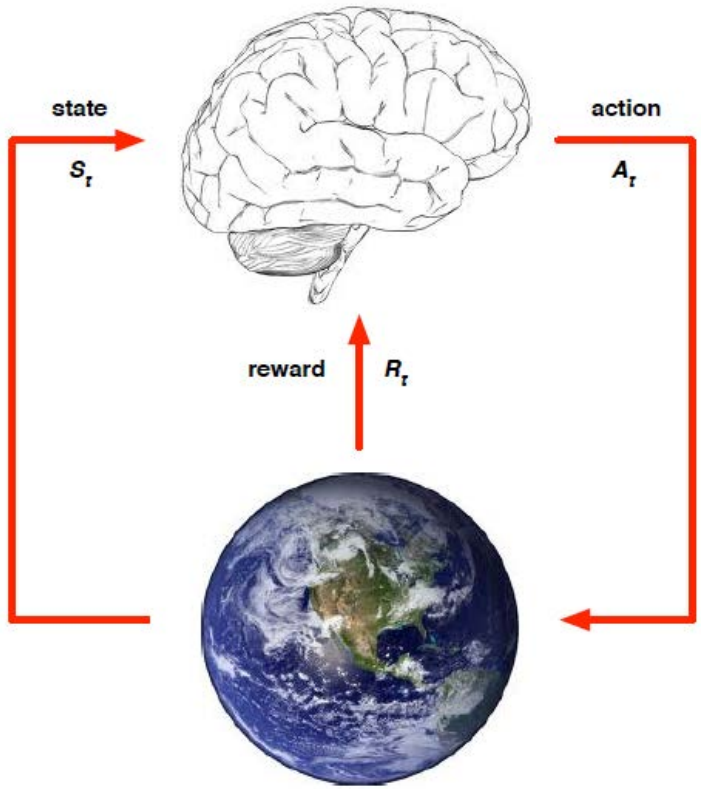


图: Fully Observable Environments

强化学习分类 (based on RL Agent)

- Value based, Policy based, Actor Critic
- Model free, Model based

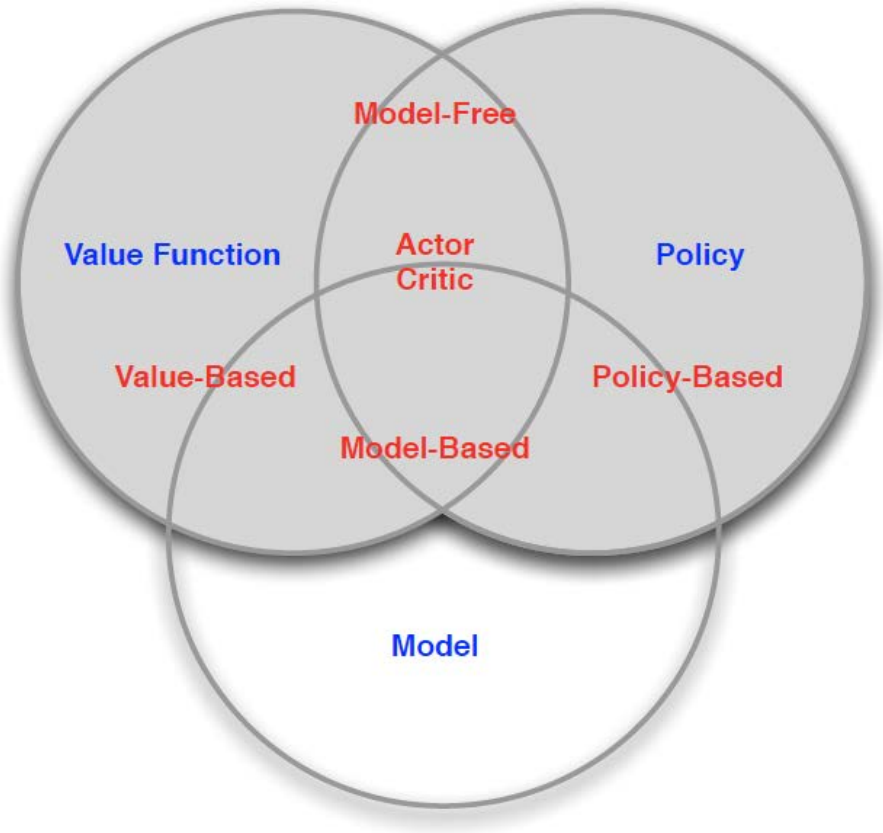


图: RL Agent

Intro to MDPs

- 下面我会开始讲解强化学习最重要的模型——马尔可夫决策过程
- 马尔科夫决策过程 (Markov decision process, MDP) 是对完全可观测的环境进行描述的模型，也就是说观测到的状态内容完整地决定了决策的需要的特征。几乎所有的强化学习问题都可以转化为 MDP。
- 该部分内容涉及的数学理论知识较多，在这次报告中，我已经尽力避免了堆积的公式。

Markov Property

马尔可夫性

马尔可夫性是一种假设，“未来的一切仅与现在有关，独立于过去的状态”。当且仅当 S_t 满足以下性质时，我们说 S_t 符合马尔可夫性。

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

从上述的式子可以看出，t+1 时刻的状态包含了 1,..,t 时刻状态的全部历史信息，并且当我们知道 t 时刻的状态后，我们只关注于环境的信息，而不用管之前所有状态的信息，这就是马尔可夫性，当论文中说某一状态或其他信息符合马尔可夫性时，我们也应当联想到这个性质。

State Transition Matrix

状态传输矩阵

对于当前的马尔可夫状态 s 和其他的状态 s'，状态传输矩阵的定义：

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix}$$

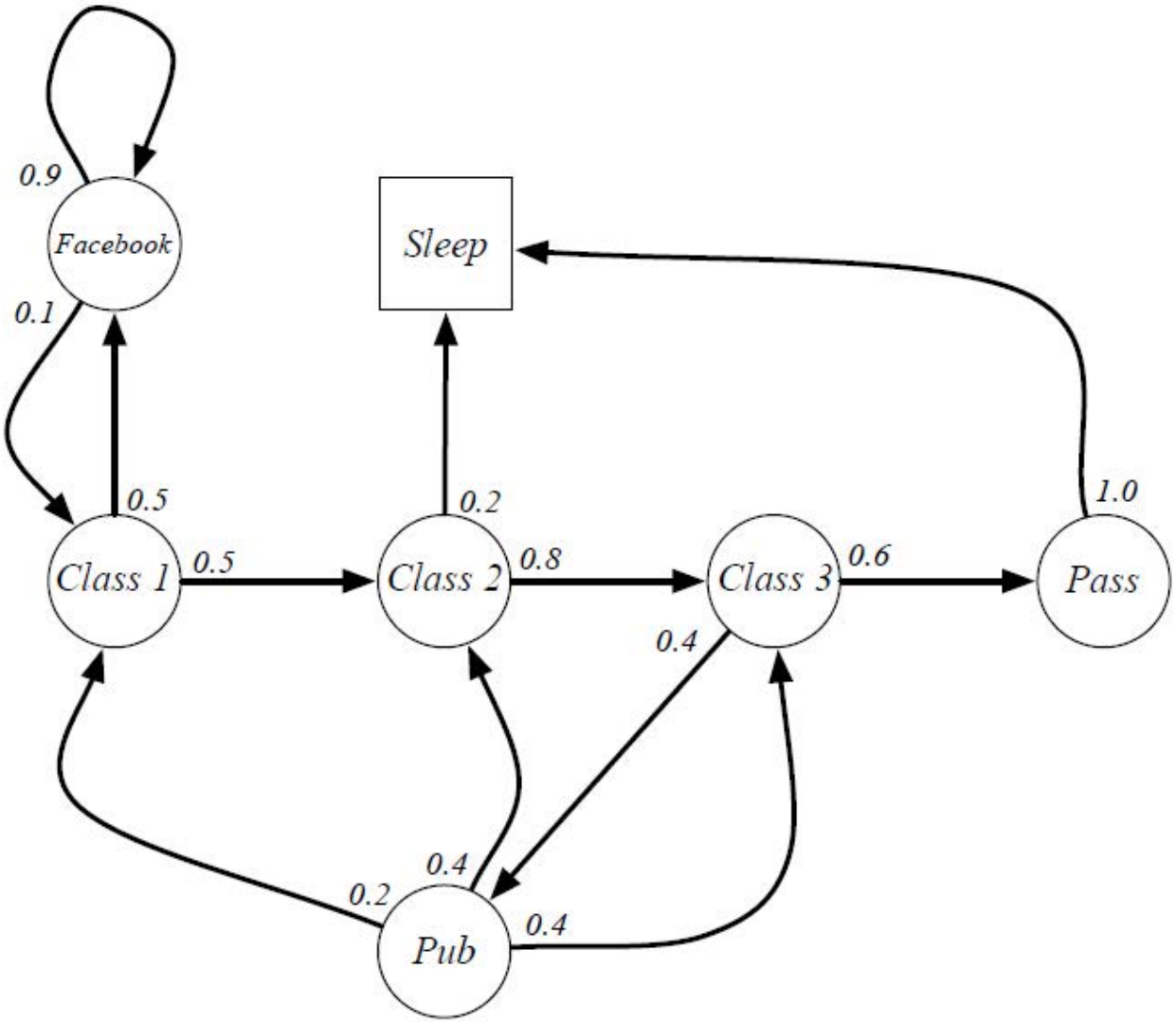
以上就是状态传输矩阵的定义，大部分的模型建立都是利用矩阵的运算的，所以这部分很重要，当然 $\sum \mathcal{P}_{ss'} = 1$，这相信比较好理解。

Markov Chain

马尔可夫过程/链

马尔可夫链 (Markov Chain) 又称马尔可夫过程 (Markov Process)，是一种无记忆的随机过程 (memoryless random process)，我们给出如下 Definition，马尔可夫链是状态与转移概率的组合 $\langle \mathcal{S}, \mathcal{P} \rangle$，其中状态 $\mathcal{S}$ 是状态的集合，概率 $\mathcal{P}$ 是概率的矩阵。

马尔可夫链举例


0.9
Facebook
0.1
Sleep
0.5
0.2
1.0
Class 1
0.5
Class 2
0.8
Class 3
0.6
Pass
0.4
0.4
0.2
Pub
0.4

Markov Reward Process

马尔可夫奖励过程

简单来说，马尔可夫奖励过程就是含有奖励的马尔可夫链，要想理解 MRP 方程的含义，我们就得弄清楚奖励函数的由来，我们可以把奖励表述为进入某一状态后收获的奖励。奖励函数如下所示：

$$\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$$

所以 MRP 的定义为 $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
其中，S 还是状态合集，P 是概率传输矩阵，R 是奖励函数如上所示，γ 是衰减因子。衰减因子是金融学上的概念，代表了对于远期利益的不确定性，其中 $\gamma \in [0, 1]$，下面讲到回报时还会提到。

Return & Value Function

收益与回报

回报的定义是从当前时刻开始的回报与衰减因子的乘积之和：

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

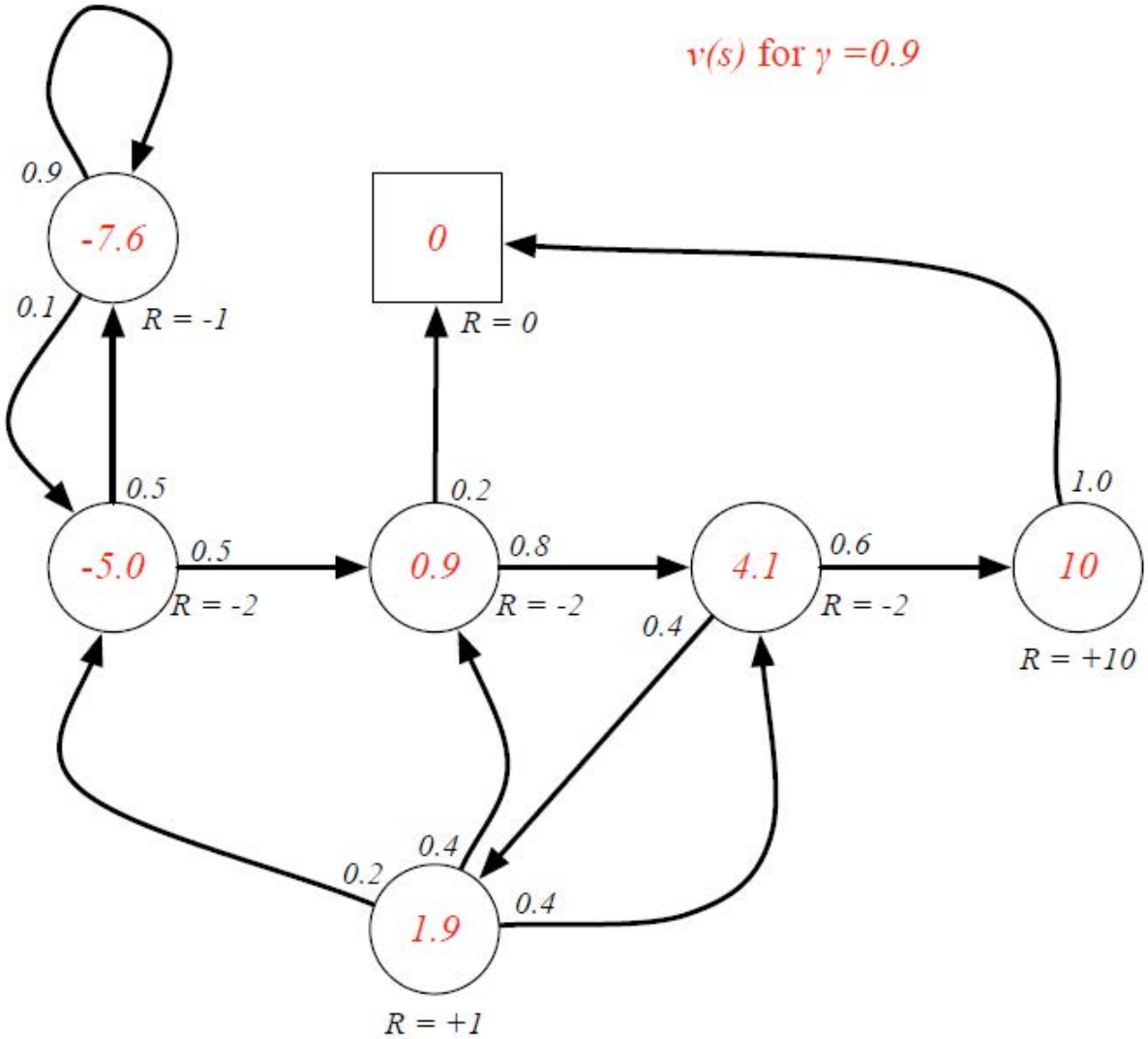
衰减因子代表人们对于未来奖励的期望，如果 γ 趋近 0，则更重视眼前的利益，如果 γ 趋近 1，则更重视未来的利益。

价值函数

价值函数的定义是当处于现在状态 s 时，MRP 未来回报的期望值，价值函数给出了当前状态的长期价值。

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

马尔可夫奖励过程举例


v(s) for γ =0.9
0.9
-7.6
0.1
R = -1
0
R = 0
0.5
-5.0
0.5
R = -2
0.2
0.9
0.8
R = -2
4.1
0.6
R = -2
1.0
10
R = +10
0.4
0.4
0.2
1.9
0.4
R = +1

Bellman equation

贝尔曼方程

要想求解马尔可夫奖励过程的价值函数，我们在这里引入了贝尔曼方程，首先让我们看看贝尔曼方程：

$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]
\end{aligned}
$$

这就是贝尔曼方程简单的推导过程，这样我们就可以把价值函数分为两部分，一部分是即时奖励 R，一部分是计算损失的下一状态的价值函数。

Bellman Equation

基于状态的贝尔曼方程

我们抛开时间的关系，仅由状态来列写上述方程：

$$v(s) = R_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

以上两式就是贝尔曼方程的两种形式了，下面我们来进行求解，将贝尔曼方程简化为矩阵形式：

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{R}_n & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

直接求解贝尔曼方程

下面进行线性方程的矩阵直接求解：

$$
\begin{aligned}
v &= \mathcal{R} + \gamma \mathcal{P} v \\
(1 - \gamma \mathcal{P}) &= \mathcal{R} \\
v &= (1 - \gamma \mathcal{P}) \mathcal{R}
\end{aligned}
$$

当然这种直接解法只能适用于小型的 MRP 模型，大型的 MRP 模型通常采用迭代的方法，比如动态规划，蒙特卡洛评估，时序差分学习等等。

Markov Decision Process

马尔可夫决策过程

MDP 就是具有决策状态的马尔可夫奖励过程，这里我们直接给出了马尔可夫决策过程的定义 $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ 显然比起马尔可夫奖励过程，我们多了一个 $\mathcal{A}$ 集合代表决策过程中所有 action 的集合。而相应的，我们也需要改动传输概率矩阵和奖励函数的定义式了，因为他们都需要与 action 有关了。

$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$\mathcal{R}_{s}^{a} = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Policies

策略 (Policies) 是 Agent 对于环境所表达的行为，这里我们给出它的定义和解释：

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

在上述定义中，MDP 模型依旧是所有状态保持马尔可夫性，则我们可以得出 MDP 的策略也只与当前状态有关，与历史无关。另外策略也是与时间无关的，仅与当前状态有关，即 $A_t \sim \pi(\cdot|S_t), \forall t > 0$. 这里我们还给出基于策略 π 的传输概率矩阵与奖励，即新的定义：

$$\mathcal{P}^{\pi}_{ss'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^{a}_{ss'}$$

$$\mathcal{R}^{\pi}_{s} = \sum_{a \in \mathcal{A}} \pi(a|s) R^{a}_{s}$$

Policy based Value Function

基于策略的价值函数

MDP 模型中有两种基于策略的价值函数：(1) 在状态 s 时收益的期望，代表的是状态带来的价值 (2) 在状态 s 时，采取动作 a 后收益的期望，代表的是动作带来的价值。两者共同构成了 MDP 的价值函数。

State-value function:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$
$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

Action-value function:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$
$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

Optimal Value Function

Theorem of MDP

- 对于 MDP 模型，我们承认存在一个最优策略 π_*，比其他任何策略都好，即 $\pi_* \geq \pi, \forall \pi$，这里我们还需要再定义策略比较的规则，即 $\pi \geq \pi'$ *if* $v_\pi(s) \geq v_{\pi'}(s), \forall s$
- 所有的最优策略都有相同的最优价值函数，即 $v_{\pi_*}(s) = v_*(s)$
- 所有的最优策略都有相同的最优动作价值函数，即 $q_{\pi_*}(s,a) = q_*(s,a)$

最优价值函数

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

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

所有的 MDP 模型的最终任务就是为了确定最优价值函数相应的策略。

Bellman Optimality Equation

贝尔曼最优方程

$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*\left(s'\right)$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*\left(s', a'\right)$$

Solving

求解贝尔曼最优方程

贝尔曼最优方程是非线性的，所以我们需要用迭代的方法去求解

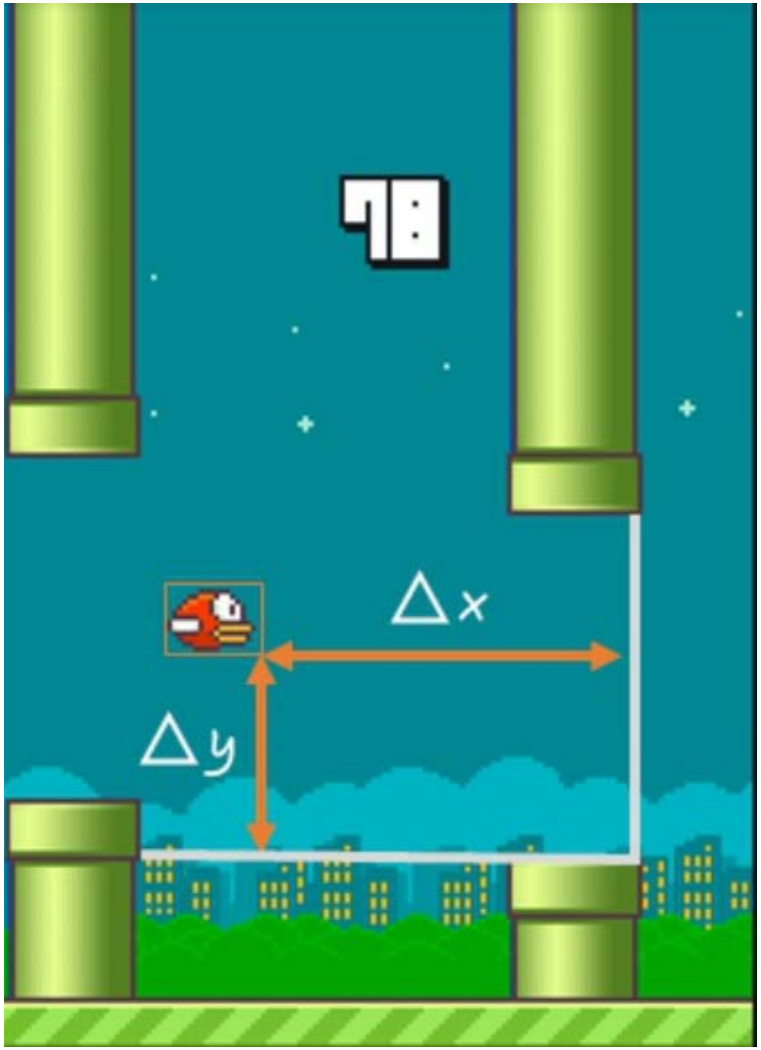
- Value Iteration 价值迭代
- Policy Iteration 策略迭代
- Q-learning
- Sara

RL Applicated in Flappy Bird

为了能实现“永远不会碰壁的 Bird”，在这个问题中我们需要确定状态 (state)、动作 (action)、奖励 (reward) 这三个要素，以便小鸟 (即智能体 Agent) 会根据当前状态来采取动作，并记录被反馈的奖赏，以便下次再到相同状态时能采取更优的动作。

状态的选择

定义状态 $(\Delta x, \Delta y)$ 如下图所示



Action & Reward

动作的选择

每一帧，小鸟只有两种动作可选：1. 向上飞一下。2. 什么都不做。

奖励的选择

小鸟活着时，每一帧给予 1 的奖赏；若死亡，则给予-1000 的奖赏。

Q-learning

What is Q ?

Q 为动作效用函数 (action-utility function)，用于评价在特定状态下采取某个动作的优劣，它是智能体的记忆。在这个问题中，状态和动作的组合是有限的。所以我们可以把 Q 当做是一张表格。表中的每一行记录了状态 $(\Delta x, \Delta y)$，选择不同动作（飞或不飞）时的奖赏。

状态	飞	不飞
$(\Delta x_1, \Delta y_1)$	1	20
$(\Delta x_1, \Delta y_2)$	20	-100
...	...	...
$(\Delta x_m, \Delta y_{n-1})$	-100	2
$(\Delta x_m, \Delta y_n)$	50	-200

update Q-table

Q 表格将根据以下公式进行更新：

$$Q(S,A) \leftarrow (1-\alpha)Q(S,A) + \alpha\left[R(S,A) + \gamma \max_{a} Q\left(S',A\right)\right]$$

其中 α 为 learning rate 学习率，γ 为 dicount factor 衰减因子。

Training Code

Algorithm 1: Q-learning

Data: state S, action A, policy $\pi(S)$, reward $R(S, A)$
Result: Optimal Q table

Initialize $Q = \{\}$;
while Q *is not Converage* **do**
    Initialize bird's State S and begin a new game ;
    while S *is not dead* **do**
        Take Action $a = \pi(S, A)$;
        $S' \leftarrow S(A)$;
        $R(S, A) \leftarrow R$;
        $Q(S, A) \leftarrow (1 - \alpha)Q(S, A) + \alpha\left[R(S, A) + \gamma \max_a Q\left(S', A\right)\right]$;
        $S \leftarrow S'$;
    end
end

神经网路

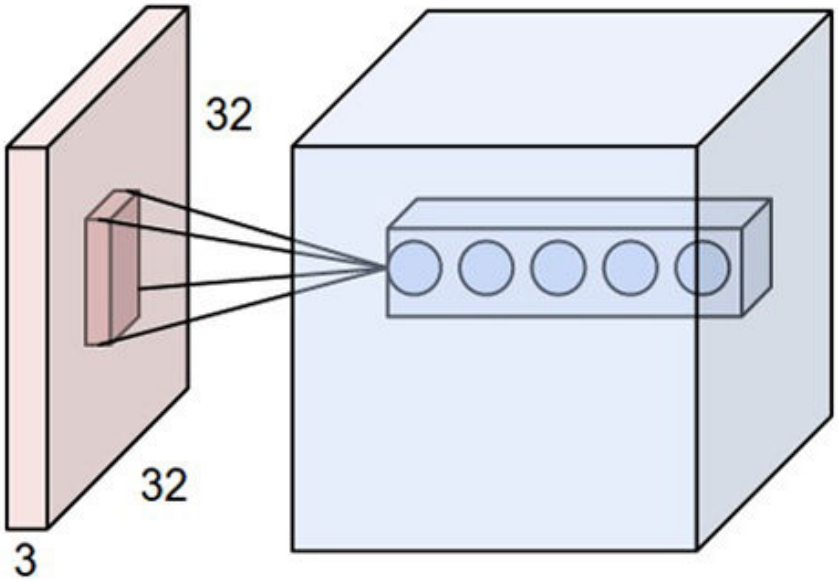
FC-NN

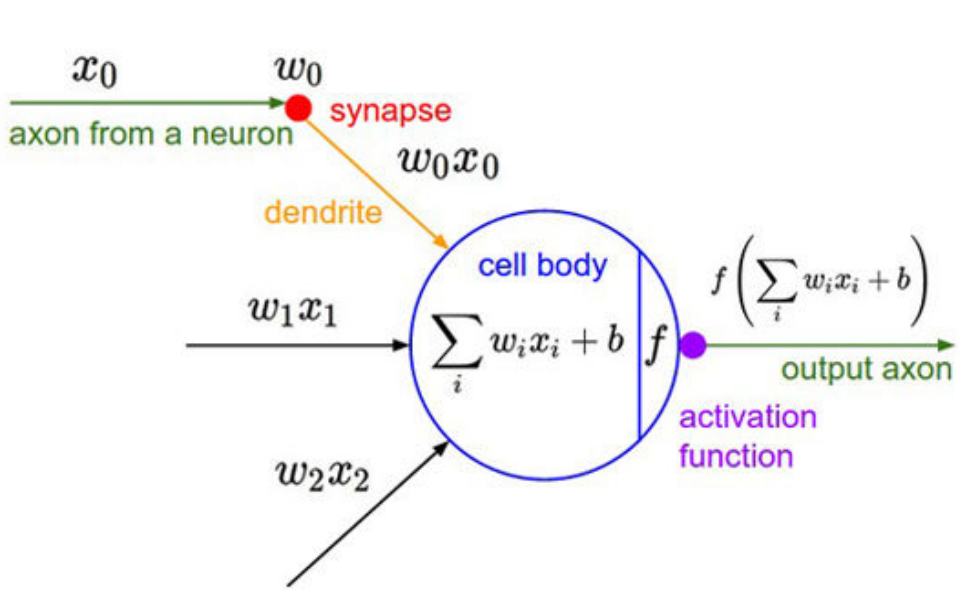
全连接神经网络 (Fully Connected Nerual Network) 属于传统神经网络，解决了很多早期的机器学习问题。

CNN & RNN

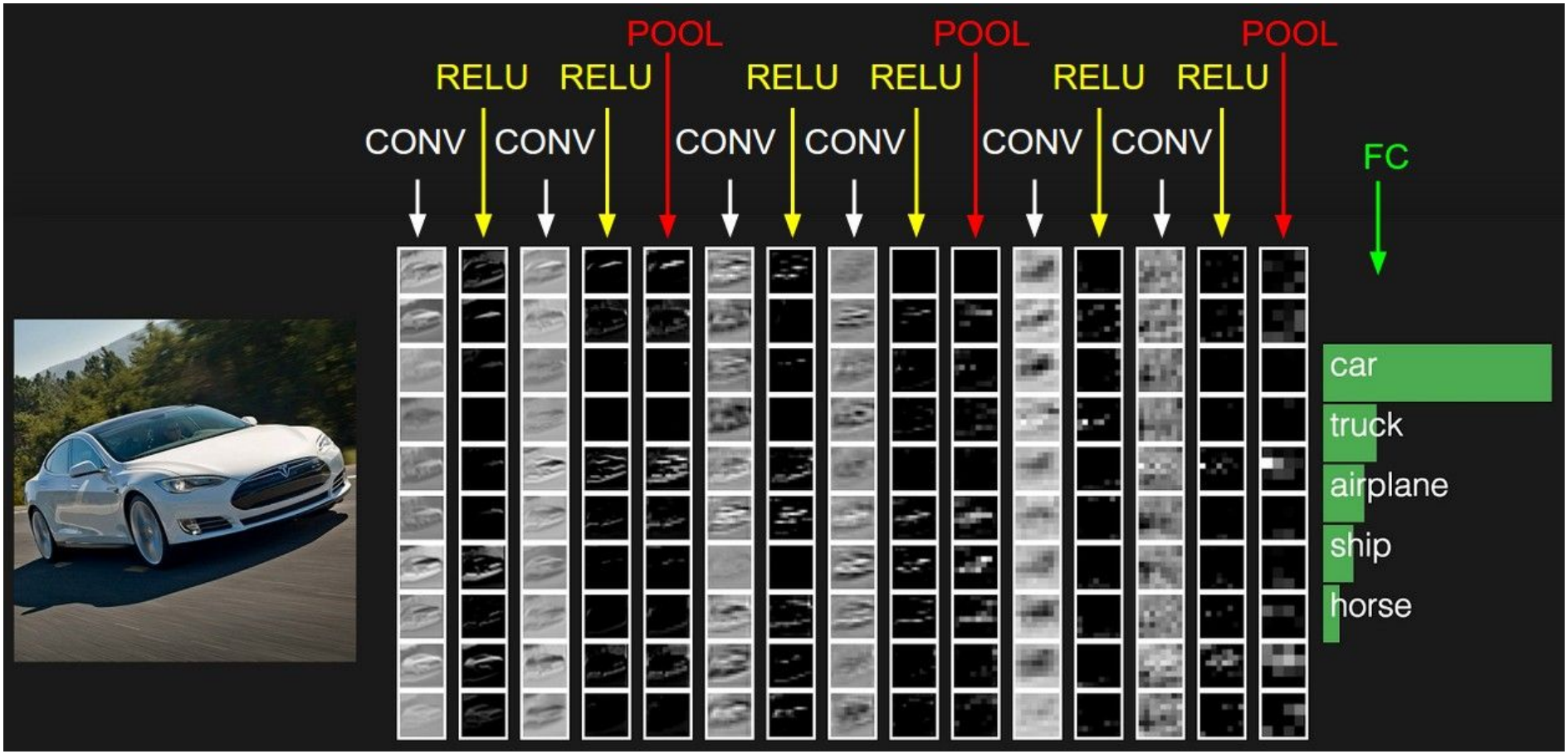
卷积神经网络 (Convolutional Nerual Network) 与循环神经网络 (Recurrent Nerual Network) 的算法均是上世纪发明的，但是由于深度学习算力的提高，这两个神经网络展示了一些恐怖的性质。

What is CNN ?


32
32
3


x_0
w_0
axon from a neuron
synapse
w_0x_0
dendrite
cell body
w_1x_1
$\sum_i w_ix_i + b$
f
$f\left(\sum_i w_ix_i + b\right)$
output axon
activation function
w_2x_2

CNN Aplicated in Image Classification



深度强化学习

DQN(Deep Q-Learning) 可谓是深度强化学习 (Deep Reinforcement Learning，DRL) 的开山之作，是将深度学习与强化学习结合起来从而实现从感知 (Perception) 到动作 (Action) 的端对端 (End-to-end) 学习的一种全新的算法。由 DeepMind 在 NIPS 2013 上发表，后又在 Nature 2015 上提出改进版本。

Why we need this ?

将深度学习与强化学习结合，直接从高维原始数据学习控制策略。而 DQN 是 DRL 的其中一种算法，它要做的就是将卷积神经网络和 Q-Learning 结合起来，CNN 的输入是原始图像数据 (作为状态 State)，输出则是每个动作 Action 对应的价值评估 Value Function (Q 值)。

DQN Solving problem

- 通过 Q-Learning 使用 reward 来构造标签
- 通过 experience replay(经验池) 的方法来解决相关性及非静态分布问题
- 使用一个 CNN(MainNet) 产生当前 Q 值，使用另外一个 CNN(Target) 产生 Target Q 值

构造标签

前面提到 DQN 中的 CNN 作用是对在高维且连续状态下的 Q-Table 做函数拟合，而对于函数优化问题，监督学习的一般方法是先确定 Loss Function，然后求梯度，使用随机梯度下降等方法更新参数。DQN 则基于 Q-Learning 来确定 Loss Function。

Q-learning 模型

θ 为网络参数，通过梯度下降或 SGD 等算法更新。

$$L(\theta) = E\left[(\text{Target } Q - Q(s, a; \theta))^2\right]$$

$$TargetQ = r + \gamma \max_{a'} Q\left(s', a'; \theta\right)$$

Experience Replay & Target Network

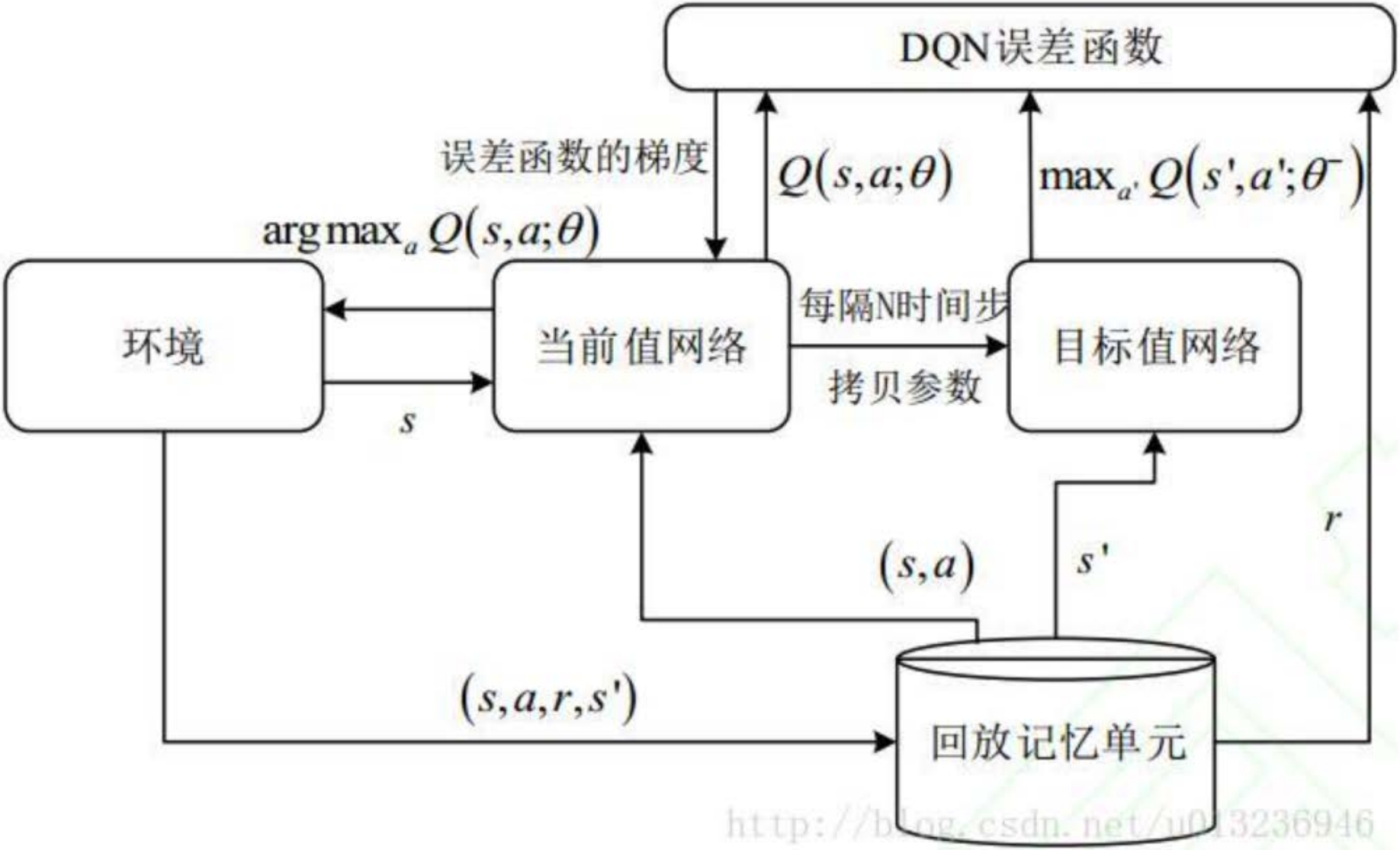
经验池

经验池的功能主要是解决相关性及非静态分布问题。具体做法是把每个时间步 agent 与环境交互得到的转移样本 (s_t, a_t, r_t, s_{t+1}) 储存到回放记忆单元，要训练时就随机用 minibatch 来训练。

目标网络

$Q(s, a; \theta_i)$ 表示当前网络 MainNet 的输出，用来评估当前状态动作对的值函数；$Q(s, a; \theta_i^-)$ 表示 TargetNet 的输出，代入上面求 TargetQ 值的公式中得到目标 Q 值。根据上面的 Loss Function 更新 MainNet 的参数，每经过 N 轮迭代，将 MainNet 的参数复制给 TargetNet。

Nature 2015


DQN误差函数
误差函数的梯度
$Q(s,a;\theta)$
$\max_{a'} Q(s',a';\theta^-)$
$\arg\max_a Q(s,a;\theta)$
环境
当前值网络
每隔N时间步
拷贝参数
目标值网络
s
(s,a)
s'
r
(s,a,r,s')
回放记忆单元

Training Code

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

- 周志华. 机器学习 [M] 北京：清华大学出版社,2016:197-204.
- 李航. 统计学习方法（第 2 版）[M] 北京：清华大学出版社, 2019.
- 阿尔法杨 XDU. 【决策模型】马尔可夫决策过程 https://zhuanlan.zhihu.com/p/271221558.
- 牛阿. 如何用简单例子讲解 Q - learning 的具体过程? https://www.zhihu.com/question/26408259/answer/123230350.
- 草帽 B-O-Y. 深度强化学习——DQN https://blog.csdn.net/u013236946/article/details/72871858.
- Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.
- Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

Q&A
知乎：阿尔法杨 XDU