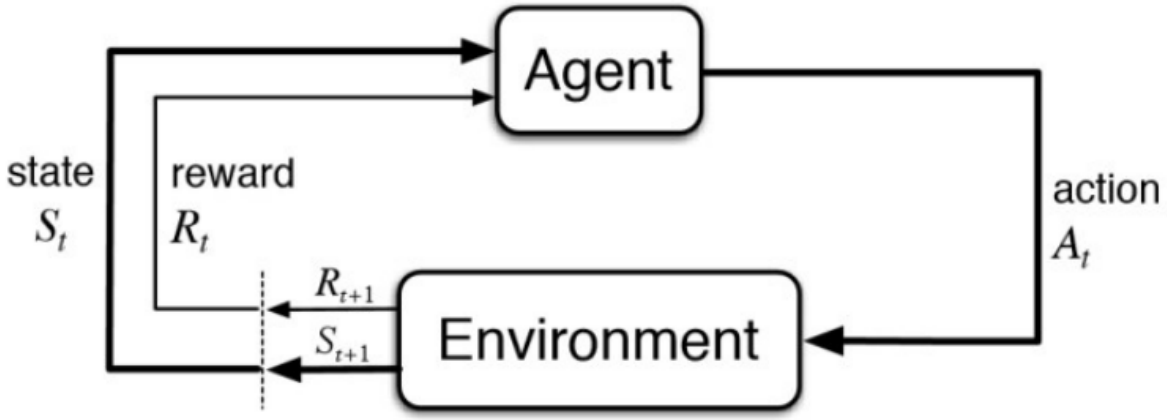
Reinforcement Knowledge Graph Reasoning for Explainable Recommendation

Terminology

Reinforcement Learning

- Reinforcement learning is a machine learning training method based on rewarding desired behaviors and/or punishing undesired ones. In general, a reinforcement learning agent is able to perceive and interpret its environment, take actions and learn through trial and error.
- 强化学习主要由智能体（Agent）、环境（Environment）、状态（State）、动作（Action）、奖励（Reward）组成



-
- Bellman

1. Bellman方程

在介绍强化学习算法之前先介绍一个比较重要的概念，就是**Bellman方程**，该方程表示动作价值函数，即在某一个状态下，计算出每种动作所对应的value（或者说预期的reward）。

$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t | S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \lambda R_{t+2} + \lambda^2 R_{t+3} + \ldots | S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \lambda\left(R_{t+2} + \lambda R_{t+3} + \ldots\right) | S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \lambda G_{t+1} | S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \lambda v\left(S_{t+1}\right) | S_t = s\right]
\end{aligned}
$$

上面公式中：

1. s表示一个具体的状态值,很自然 $S_t, S_{t+1}, \ldots$ 就是表示当前时刻，下一时刻和下下一时刻,...,的状态
2. R_{t+1} 表示在t+1时刻所获得的奖励，其他同理
3. G_t 表示t时刻总的回报奖励，因为当前时刻做的某一个决定，未来不同时刻都会有不同形式的奖励。（或者也可以这么理解：前面 G_t 代表的是当前时刻某一个动作所带来的的奖励，而 $v(s)$ 就表示在当前时刻的一个奖励期望，即综合考虑所能采取的所有动作之后我们所能获得的奖励，我们把 $v(s)$ 称为**value function**

上面这个公式就是Bellman方程的基本形态。从公式上看，当前状态的价值和下一步的价值以及当前的反馈Reward有关。它表明**价值函数（Value Function）**是可以通过迭代来进行计算的!!!

  -

- Action-value function

2. 动作价值函数

前面介绍的Bellman方程是价值函数，它直接估计的是某个状态下所有动作的价值期望，但是如果我们能够知道某个状态下每个动作的价值岂不是更好？这样我们可以选择价值最大的那个动作去执行，所以就有了**动作价值函数（action-value function）**，它的表达形式其实是类似的：

$$
\begin{aligned}
Q^{\pi}(s,a) &= \mathbb{E}\left[r_{t+1} + \lambda r_{t+2} + \lambda^2 r_{t+3} + \ldots | s,a\right] \\
&= \mathbb{E}_{s'}\left[r + \lambda Q^{\pi}\left(s',a'\right)|s,a\right]
\end{aligned}
$$

上面公式中的 π 表示动作选择策略(例如以0.1的概率随机选择action，以0.9的概率按照Q网络选择value值最大的action)

其实我们最初的目的是找到当前状态下应该执行哪个action，但是如果我们求解出最优的 $Q^{\pi}(s,a)$,其实也就等价于找到了这个action，这种求解方法也叫**value-based方法。**

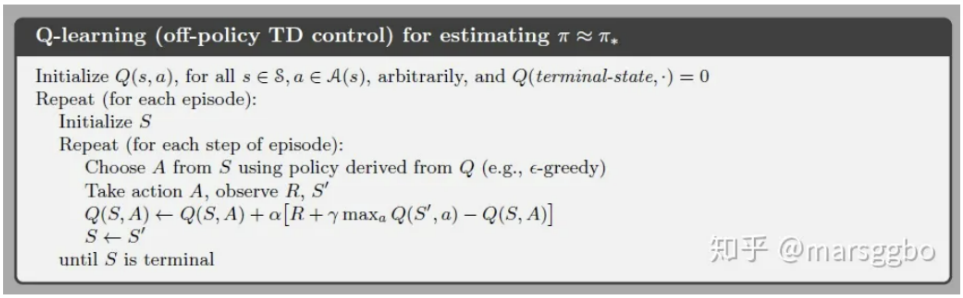
> 其实还有**policy-based(直接计算策略函数)** 和 **model-based(估计模型，即计算出状态转移函数，进而求解出整个MDP(马尔科夫过程)过程)** 方法,下面主要以介绍value-based为主。

最优的动作价值函数为：

$$
Q(s,a) = \max \pi Q^{\pi}(s,a) = \mathbb{E}s'\left[r + \lambda \max_{a'} Q\left(s',a'\right)|s,a\right]
$$

有一点要注意的是 $Q^{*}(s,a)$ 表示的是在t时刻的动作价值最优值，而仔细看看上面的等式可以发现，我们还需要求解出下一个状态S'所对应的动作价值最优解。我们还在计算当前的Q值，怎么能有下个状态的Q值呢？所以，在实际运用时，我们会使用之前的Q值，也就是说每次我们会根据新得到的reward和原来的Q值来更新现在的Q值，具体的可以看看下面的算法介绍。

-
- Q- Learning
  - https://zhuanlan.zhihu.com/p/98962807
-

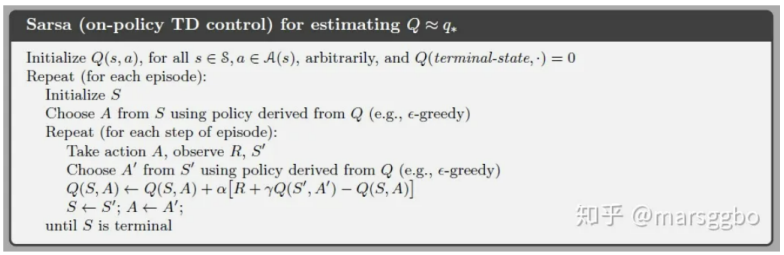


2. 算法流程图

- 初始化环境状态**S**
- 将当前环境状态**S**输入到Q网络(即策略网络，保存了action和value对应关系的table)，然后输出当前状态的动作**A**
- 更新Q网络
  - $Q_{target} = R + \gamma \max_a Q(S', a)$ 表示Q真实值，简单理解就是我在S状态下采取了action，从环境中获得了R的奖励，然后对下一时刻的Q值应该也是有影响的，这个影响因子就是γ。另外这次是是一个递归的表达式，所以也可以看出离当前时刻越远，我所采取动作的影响力就越低。
  - $Q_{target} - Q(S, A)$ 就是常说的TD(temporal difference) error，这个error在后面的DQN中会作为损失函数。
- 更新当前状态为**S'**
- 返回第二步重复执行，直到满足限定条件

- Sarsa

  #### 1. 算法总结



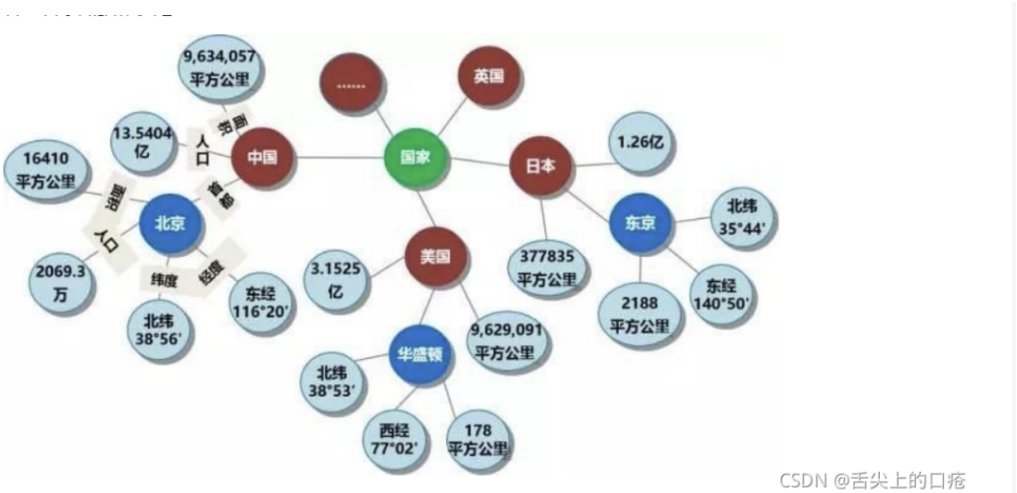
3. 和Q-learning的区别

其实可以看到Q-learning和Sarsa的最大区别就是对Q网络的更新策略，Sarsa使用的是使用**下次状态所采取的的动作所对应的Q值**来更新Q值，而Q-learning使用**下次状态S2的最大Q值**用于更新。

感性的理解就是Sarsa会探索更多的可能性，而Q-learning会铁定心地选择最大可能性的选择。因此，Q-learning虽然具有学习到全局最优的能力，但是其收敛慢；而Sarsa虽然学习效果不如Q-learning，但是其收敛快，直观简单。因此，对于不同的问题，我们需要有所斟酌。

knowledge graphs (KG)

- In knowledge representation and reasoning, knowledge graph is a knowledge base that uses a graph-structured data model or topology to integrate data.

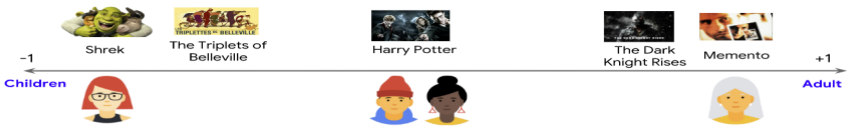


如图所示，你可以看到，如果两个节点之间存在关系，他们就会被一条无向边连接在一起，那么这个节点，我们就称为**实体（Entity）**，它们之间的这条边，我们就称为**关系（Relationship）**。

-

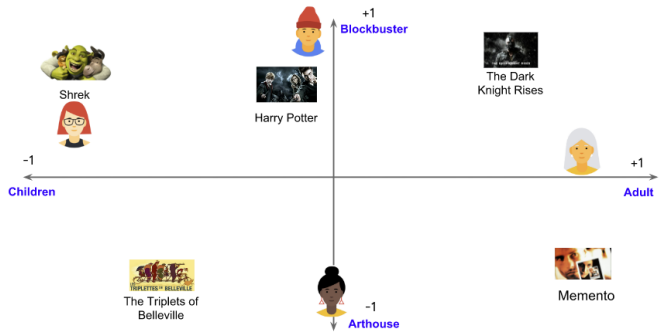
Collaborative filtering (CF) 协同过滤

- To address some of the limitations of content-based filtering, collaborative filtering uses similarities between users and items simultaneously to provide recommendations.
- 1D Embedding

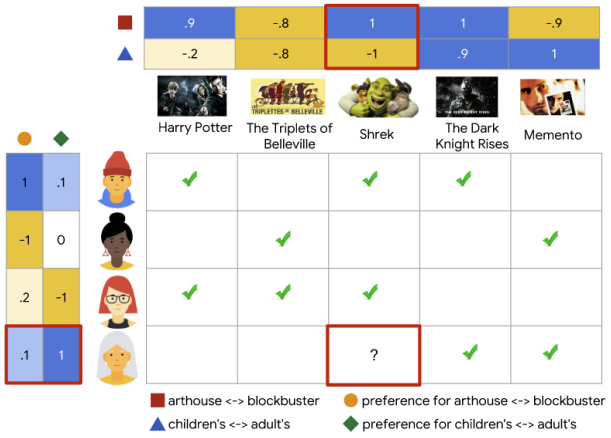


In the diagram below, each checkmark identifies a movie that a particular user watched. The third and fourth users have preferences that are well explained by this feature—the third user prefers movies for children and the fourth user prefers movies for adults. However, the first and second users' preferences are not well explained by this single feature.

-
- 2D Embedding



We again place our users in the same embedding space to best explain the feedback matrix: for each (user, item) pair, we would like the dot product of the user embedding and the item embedding to be close to 1 when the user watched the movie, and to 0 otherwise.



1.基于用户的协同过滤

思想：找到和目标用户相似的用户,推荐该相似用户使用过但该用户没见过的物品。

主要有两个步骤：

1.计算每个用户和目标用户的相似度，并选出n个最相似的(n为一个超参数)

2.根据相似用户对某一物品的评价计算出目标用户对这个物品的评价，设定一个阈值判断是否推荐。

示例：

	物品1	物品2	物品3	物品4	物品5
Alice	5	3	4	4	?
用户1	3	1	2	3	3
用户2	4	3	4	3	5
用户3	3	3	1	5	4
用户4	1	5	5	2	1

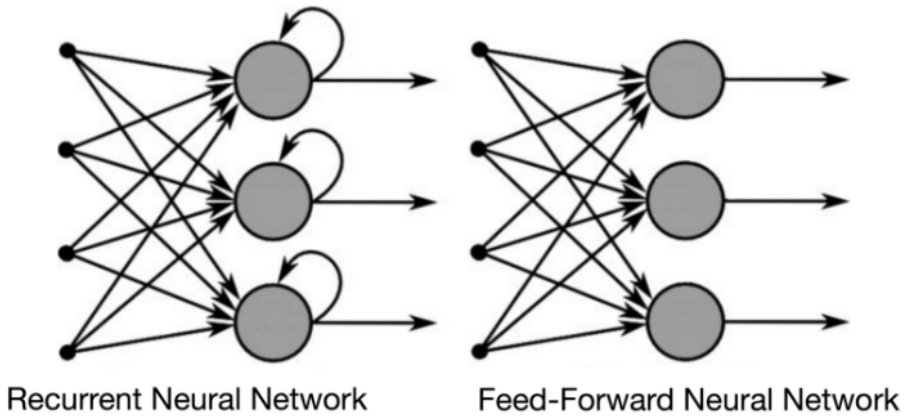
图1.评分矩阵

这是一个不同用户对于不同物品的评分矩阵，现在要决定是否对Alice推荐物品5，也就是要计算出Alice对物品5的评分。

-

Recurrent Neural Networks (RNN)

- A recurrent neural network is a class of artificial neural networks where connections between nodes can create a cycle, allowing output from some nodes to affect subsequent input to the same nodes. This allows it to exhibit temporal dynamic behavior.



-

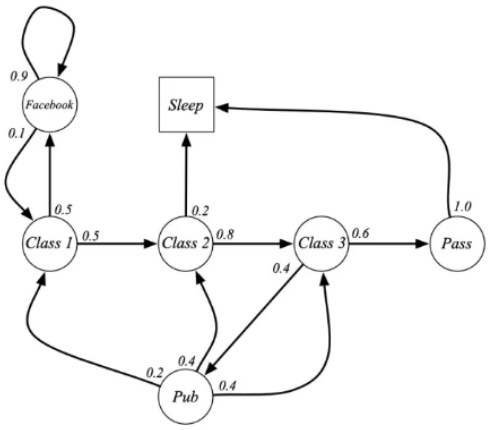
beam search-based algorithm

- 在当前级别的状态下计算所有可能性，并按照递增顺序对他们进行排序，但只保留一定数量的可能结果（依据Beam Width决定数量），接着根据这些可能结果进行扩展，迭代以上的动作直到搜索结束并返回最佳解（具有最高概率的那个）。
- beam size > 1的greedy search

Markov Decision Process (MDP)

- A Markov decision process problem is a tuple , where S is the underlying state space, A is the set of actions, is the cost or immediate reward function, and is the probability that action a in state u will lead to state v.
- MRP是带有价值（Value）的马尔可夫链，以元组<S,P,R,γ>表示
- S 表示有限数量的状态集
- P 是各状态间的转移矩阵（Transition Matrix）
- 回报函数 R, 返回的是一个标量（实数），表示：假设目前状态是S，仅考虑下一状态 St+1 能获得多少回报 Rs=E[Rt+1|St=s]，也称立即回报（Immediate Reward）
- 折扣系数 γ∈[0,1]，γ等于0意味着只看眼前，γ 等于1意味着长远和眼前一样重要，重视长远，通过调节γ 的数值，来控制重视长远的程度。
-

  下图是David Silver课程中的例子，学生从状态Class 1到状态Class 2的概率是0.5，对应右侧矩阵第1行第2列（C1,C2）的值0.5，学生在Facebook不能自拔，状态转移概率0.9，对应矩阵（FB，FB）位置为0.9。注意到：每一行所有数值相加等于1。



$$\mathcal{P} = \begin{array}{c} \\ C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{array} \begin{array}{c} \begin{array}{ccccccc} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{array} \\ \begin{bmatrix} & 0.5 & & & & 0.5 & \\ & & 0.8 & & & & 0.2 \\ & & & 0.6 & 0.4 & & \\ & & & & & & 1.0 \\ 0.2 & 0.4 & 0.4 & & & & \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1 \end{bmatrix} \end{array}$$

-

Monte Carlo Method

- Monte Carlo involves learning through sampling rewards from the environment and averaging over those obtained rewards. For every episode, our agent takes action and obtains rewards. When each episode(experience) terminates then we take the average.
- 蒙特卡洛又称统计试验法，是基于概率论的算法。其实质就是将问题转化为一个概率问题，并用计算机模拟产生一堆随机数，再对随机数进行统计工作。
- 蒙特卡洛模拟方法=建立概率模型+计算机模拟+数理统计。
- Incremental means:
    - Also known as moving averages, it maintains the mean incrementally i.e. the mean is calculated as soon as new reward is encountered. In order to calculate inremental mean, we don't wait till every reward is encountered and then divide it by the total number of times that state was visited. We calculate it in an online fashion.

‣ The total mean for n time steps is calculated as :

$$U(n) = \frac{1}{N} \sum_{i=1}^{N} X(i)$$

Total Mean/average

‣ Let's say, U(n) is the new mean we want to calculate. Then this U(N) can be split into the sum of the previous mean till **N-1** and the value obtained at **N** timestep. Can be written as:

$$U(n) = \frac{1}{N} \sum_{i=1}^{N-1} X(i) + X(N)$$

New Mean = Total Mean (N-1) + Value at N timestep

‣ Now looking at the summation term, it can be defined as (**N-1**) times the mean obtained till N-1 i.e. U(N-1). Substituting this in the above equation:

$$U(N) = \frac{1}{N} ((N-1) U_{N-1} + X(N))$$

Substituting

‣ Rearranging these terms :

$$U(N) = \frac{1}{N} (N. U_{N-1} - U_{N-1} + X(N))$$

Opening the bracket

‣ Taking N common and Solving further:

$$U(N) = N.\frac{1}{N} U_{N-1} + \frac{1}{N} (X_N - U_{N-1})$$
$$U(N) = U_{N-1} + \frac{1}{N} (X_N - U_{N-1})$$

The final Equation for Incremental Means

    - It tells that the new mean (U(N)) can be defined as the sum of the previous estimate of the mean (U(N-1)) and the difference between the current obtained

return (X(N)) and the previous estimate of the mean (U(N-1)) weighted by some step size i.e. the total number of times a state was encountered over different episodes. The difference between the current obtained return (X(N)) and the previous estimate of the mean (U(N-1)) is called the error term and it means that we are just estimating our new mean in direction of this error term. It is important to note that we are not accurately calculating the new mean but just pushing the new mean in the direction of the real mean.

REINFORCE with baseline

- the learned state-value function estimates the value of the only the first state of each state transition
- http://www.yaotu.net/biancheng/32977.html
- baseline一词应该指的是对照组，基准线，就是你这个实验有提升，那么你的提升是对比于什么的提升，被对比的就是baseline

Policy network

- https://blog.csdn.net/Wwm52/article/details/125575253

  > 一个神经网络，输入是状态，输出直接就是动作（不是Q值）。

  前面三种算法都是基于价值(value)的方法，即输入当前状态，然后计算出每个action的价值，最后输出价值最大的action。而policy network则是根据某种策略直接输出action,即 $A = \pi(S, \theta)$ 或者表示为

  $$A^* = \underset{A}{\operatorname{argmax}}\ \pi(A|S, \theta)$$

-
- 策略网络即一个神经网络模型，可以通过观察当前的环境状态，来直接预测出一个最佳的行动策略，使这个策略可以获得最大的期望收益。得到每个行动方案所对应的概率。
-
  - Loss 函数

1. loss函数构造

和前面算法类似，一个比较直观的损失函数构造方式如下

$$L(\theta) = \mathbb{E}\left(r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots | \pi(,\theta)\right)$$

但是上面的loss函数有个问题是式子中的 $r_1, r_2, \ldots$ 都是从环境中获取的，那么如何对策略 $\pi(\theta)$ 做参数更新呢?

我们换个角度想，如果一个action得到的reward多，我们就应该加大这个action的概率，反之就减少。所以目标函数可以写成如下形式:

$$E_x[f(x)] = \sum_x p(x) f(x)$$

其中x表示某个action，p(x)和f(x)分别表示该action的概率和对应的reward。

> 更一般地说，f(x)应该是对action的评价指标，我们可以用reward，当然也可以用其他的指标，如Q值等等。换句话说Policy Network的核心就是这个评价指标的选取。

我们继续分析上面的目标函数，将目标函数对策略网络的参数 θ 做求导:

$$\begin{aligned}
\nabla_\theta E_x[f(x)] &= \nabla_\theta \sum_x p(x) f(x) && \text{definition of expectation} \\
&= \sum_x \nabla_\theta p(x) f(x) && \text{swap sum and gradient} \\
&= \sum_x p(x) \frac{\nabla_\theta p(x)}{p(x)} f(x) && \text{both multiply and divide by } p(x) \\
&= \sum_x p(x) \nabla_\theta \log p(x) f(x) && \text{use the fact that } \nabla_\theta \log(z) = \frac{1}{z} \nabla_\theta z \\
&= E_x\left[f(x) \nabla_\theta \log p(x)\right] && \text{definition of expectation}
\end{aligned}$$

由上面的求导可知，其实目标函数也可以写成

$$\begin{aligned}
L(\theta) &= -\sum \log p(x) f(x) \\
&= -\sum \log \pi(A|S,\theta) f(A|S)
\end{aligned}$$

下图是文献中的截图，总结了多种评价指标，如Q,reward,TD等。

-

Value network

- 与策略网络不同的是，估值网络则是学习action对应的期望价值，成为Q-learning，期望价值指的是从当前的这一步到后续的所有步骤总共可以获得的期望的最大值，用Q表示。
- Q-learning
  - https://www.jianshu.com/p/1c0d5e83b066
- DQN(Deep Q-learning Network)

DQN(Deep Q-learning Network)

> 通过计算每一个状态动作的价值，然后选择价值最大的动作执行。

1. 深度学习如何和强化学习结合?

前面介绍的Q-learning和Sarsa的action和state都是在离散空间中，但是有的情境下无法用离散空间表达，而且如果真的用离散空间表达，那么空间会非常巨大，这对计算机来说会很难处理。例如自动驾驶车的state和action，我们不可能用一个表格来记录每个state和对应action的value值，因为几乎有无限种可能。那么如何解决这种问题呢?

我们以自动驾驶为例，仔细想想可以知道，车的状态是高维表示的（例如，当前的位置，车的油耗，路况等等数据来表示当前状态），而动作相对来说可能是低维的（为方便说明，假设速度恒定，最后的动作只有方向盘旋转角度）。

因为我们要做的是针对某一时刻的状态选择最合适的动作，所以我们可以把车状态当做高维输入数据，车的当前时刻的动作当做是低维输出，我们可以对二者构建一个映射关系。

$$x \rightarrow f(x|W) \rightarrow y$$
$$S \rightarrow f(S|W) \rightarrow A$$

上面等式的含义就是对把状态S作为输入数据，然后经过映射函数 $f(S|W)$ 后得到一个向量 $[Q(s,a_1),Q(s,a_2),Q(s,a_3),\ldots,Q(s,a_n)]$ ，这个卷积神经网络做图像分类任务时的输出值的含义类似，每个位置代表不同action的value，我们可以选择value值最大的作为S状态的action。

那么这个映射函数就可以用 **"万能"** 的神经网络代替，也就是后面要介绍的**DQN**了。

2. 如何训练DQN?

1） loss函数构造

我们知道，要训练一个神经网络，那么我们就需要构建loss函数，而这个loss函数的构建又需要真实的label和预测的label。

预测的label很好理解，其实就是最终得到的输出向量嘛，那么真实的label是什么呢？其实就是前面Q-learning算法中介绍到的 Q_{target} ,所以**TD error**表达式如下:

$$L(w)=\mathbb{E}[\underbrace{r+\gamma \max a' Q(s',a',w)}_{\text{Target}}-Q(s,a,w))^2]$$

2）训练算法

Playing Atari with Deep Reinforcement Learning

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

具体的算法还涉及到很多细节，例如Experience Replay，也就是经验池的技巧，就是如何存储样本及采样问题。按照脑科学的观点，人的大脑也具有这样的机制，就是在回忆中学习。那么上面的算法看起来那么长，其实就是反复试验，然后存储数据。接下来数据存到一定程度，就每次随机采用数据，进行梯度下降！也就是在DQN中增强学习Q-Learning算法和深度学习的SGD训练是同步进行的！通过Q-Learning获取无限量的训练样本，然后对神经网络进行训练。样本的获取关键是计算y，也就是标签。

Exponential Linear Unit (ELU)

- The Exponential Linear Unit (ELU) is an activation function for neural networks. In contrast to ReLUs, ELUs have negative values which allows them to push mean unit activations closer to zero like batch normalization but with lower computational complexity.

Term frequency-inverse document frequency (TF_IDF)

- TF-IDF stands for term frequency-inverse document frequency and it is a measure, used in the fields of information retrieval (IR) and machine learning, that can quantify the importance or relevance of string representations (words, phrases, lemmas, etc) in a document amongst a collection of documents (also known as a corpus).
- F-IDF模型（term frequency–inverse document frequency，词频与逆向文件频率）。TF-IDF是一种统计方法，用以评估某一字词对于一个文件集或一个语料库的重要程度。TF-IDF的主要思想是，如果某个词或短语在一篇文章中出现的词频高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。
- TF-IDF有两个值，一个是词频率，另一个是IDF（inverse document frequency，逆向文件频率）。如图中的计算方式。

> **词权重： 基于Tf-idf模型**
>
> 主要思想：如果某个词或短语在一篇文章中出现的词频高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力
>
> $$tfidf = tf * idf$$
> $$tf = \frac{\text{词的频数}}{\text{句子总词数}}$$
> $$idf = \log \frac{\text{总文档数}}{\text{包含该词的文档数}}$$
>
> 例子：
> 库中10000篇文档，10000篇提到母牛，10篇提到产奶量
>
> 某篇文章：。。。**母牛**的**产奶量**。。。(100个词，母牛出现五次，产奶量出现两次)
>
> **母牛：** tf = 5/100, idf = log 10000/10000=0,
> Tfidf: 0.05*0=0
> **产奶量：** tf = 2/100, idf = log 10000/10=3 ,
> Tfidf: 0.02*3 = 0.06

-