Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see [The Python Standard Library](https://docs.python.org/3/library/index.html#library-index). [The Python Language Reference](https://docs.python.org/3/reference/index.html#reference-index) gives a more formal definition of the language. To write extensions in C or C++, read [Extending and Embedding the Python Interpreter](https://docs.python.org/3/extending/index.html#extending-index) and [Python/C API Reference Manual](https://docs.python.org/3/c-api/index.html#c-api-index). There are also several books covering Python in depth.

This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python’s most noteworthy features, and will give you a good idea of the language’s flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in [The Python Standard Library](https://docs.python.org/3/library/index.html#library-index).

**Python 简介**

Python 是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。

Python 的设计具有很强的可读性，相比其他语言经常使用英文关键字，其他语言的一些标点符号，它具有比其他语言更有特色语法结构。

* **Python 是一种解释型语言：** 这意味着开发过程中没有了编译这个环节。类似于PHP和Perl语言。
* **Python 是交互式语言：** 这意味着，您可以在一个 Python 提示符 **>>>** 后直接执行代码。
* **Python 是面向对象语言:** 这意味着Python支持面向对象的风格或代码封装在对象的编程技术。
* **Python 是初学者的语言：**Python 对初级程序员而言，是一种伟大的语言，它支持广泛的应用程序开发，从简单的文字处理到 WWW 浏览器再到游戏。

**Python 发展历史**

Python 是由 Guido van Rossum 在八十年代末和九十年代初，在荷兰国家数学和计算机科学研究所设计出来的。

Python 本身也是由诸多其他语言发展而来的,这包括 ABC、Modula-3、C、C++、Algol-68、SmallTalk、Unix shell 和其他的脚本语言等等。

像 Perl 语言一样，Python 源代码同样遵循 GPL(GNU General Public License)协议。

现在 Python 是由一个核心开发团队在维护，Guido van Rossum 仍然占据着至关重要的作用，指导其进展。

Python 2.7 被确定为最后一个 Python 2.x 版本，它除了支持 Python 2.x 语法外，还支持部分 Python 3.1 语法。

**Python 特点**

* **1.易于学习：**Python有相对较少的关键字，结构简单，和一个明确定义的语法，学习起来更加简单。
* **2.易于阅读：**Python代码定义的更清晰。
* **3.易于维护：**Python的成功在于它的源代码是相当容易维护的。
* **4.一个广泛的标准库：**Python的最大的优势之一是丰富的库，跨平台的，在UNIX，Windows和Macintosh兼容很好。
* **5.互动模式：**互动模式的支持，您可以从终端输入执行代码并获得结果的语言，互动的测试和调试代码片断。
* **6.可移植：**基于其开放源代码的特性，Python已经被移植（也就是使其工作）到许多平台。
* **7.可扩展：**如果你需要一段运行很快的关键代码，或者是想要编写一些不愿开放的算法，你可以使用C或C++完成那部分程序，然后从你的Python程序中调用。
* **8.数据库：**Python提供所有主要的商业数据库的接口。
* **9.GUI编程：**Python支持GUI可以创建和移植到许多系统调用。
* **10.可嵌入:**你可以将Python嵌入到C/C++程序，让你的程序的用户获得"脚本化"的能力。

# 强化学习通俗导论（一）：什么是强化学习

2018-06-16 22:12:21 [刺客伍六七](https://me.csdn.net/qq_39521554) 阅读数 23771  收藏 更多

分类专栏： [Data Science：统计学习](https://blog.csdn.net/qq_39521554/category_7361350.html) [Data Science：机器学习](https://blog.csdn.net/qq_39521554/category_7361355.html)

#### 1. 定义

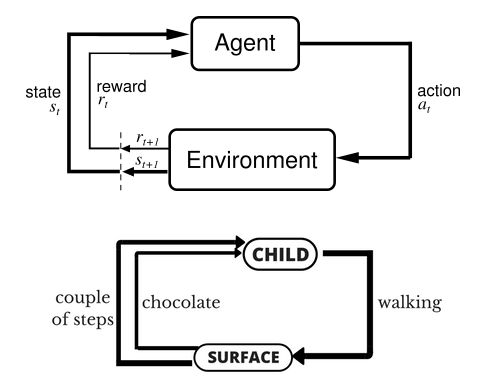
强化学习是机器学习的一个重要分支，是多学科多领域交叉的一个产物，它的本质是解决 **decision making 问题**，即自动进行决策，并且可以做连续决策。

它主要包含四个元素，**agent，环境状态，行动，奖励**, 强化学习的目标就是获得最多的累计奖励。

让我们以小孩学习走路来做个形象的例子：

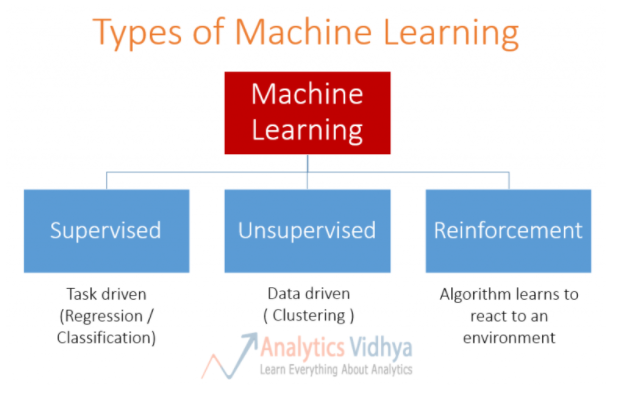
小孩想要走路，但在这之前，他需要先站起来，站起来之后还要保持平衡，接下来还要先迈出一条腿，是左腿还是右腿，迈出一步后还要迈出下一步。

小孩就是 agent，他试图通过采取行动（即行走）来操纵环境（行走的表面），并且从一个状态转变到另一个状态（即他走的每一步），当他完成任务的子任务（即走了几步）时，孩子得到奖励（给巧克力吃），并且当他不能走路时，就不会给巧克力。



#### 2. 和监督式学习, 非监督式学习的区别

在机器学习中，我们比较熟知的是监督式学习，非监督学习，此外还有一个大类就是强化学习：



**强化学习和监督式学习的区别：**

监督式学习就好比你在学习的时候，**有一个导师在旁边指点**，他知道怎么是对的怎么是错的，但在很多实际问题中，例如 chess，go，这种有成千上万种组合方式的情况，不可能有一个导师知道所有可能的结果。

而这时，强化学习会在没有任何标签的情况下，通过先尝试做出一些行为得到一个结果，通过这个结果是对还是错的反馈，调整之前的行为，就这样不断的调整，算法能够学习到在什么样的情况下选择什么样的行为可以得到最好的结果。

**就好比你有一只还没有训练好的小狗**，每当它把屋子弄乱后，就减少美味食物的数量（惩罚），每次表现不错时，就加倍美味食物的数量（奖励），那么小狗最终会学到一个知识，就是把客厅弄乱是不好的行为。

两种学习方式都会学习出输入到输出的一个映射，监督式学习出的是之间的关系，可以告诉算法什么样的输入对应着什么样的输出，强化学习出的**是给机器的反馈 reward function**，即用来判断这个行为是好是坏。

另外强化学习的结果反馈有延时，有时候可能需要走了很多步以后才知道以前的某一步的选择是好还是坏，而监督学习做了比较坏的选择会立刻反馈给算法。

而且强化学习面对的输入总是在变化，每当算法做出一个行为，它影响下一次决策的输入，而监督学习的输入是独立同分布的。

通过强化学习，一个 agent 可以在探索和开发（exploration and exploitation）之间做权衡，并且选择一个最大的回报。   
exploration 会尝试很多不同的事情，看它们是否比以前尝试过的更好。   
exploitation 会尝试过去经验中最有效的行为。

一般的监督学习算法不考虑这种平衡，就只是是 exploitative。

**强化学习和非监督式学习的区别：**

非监督式不是学习输入到输出的映射，而是模式。例如在向用户推荐新闻文章的任务中，非监督式会找到用户先前已经阅读过类似的文章并向他们推荐其一，而强化学习将通过向用户先推荐少量的新闻，并不断获得来自用户的反馈，最后构建用户可能会喜欢的文章的“知识图”。

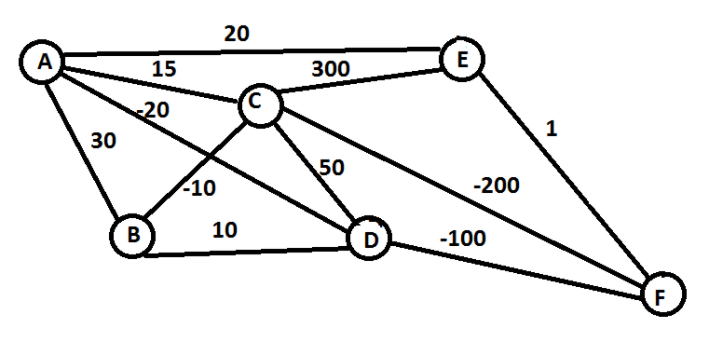
#### 3. 主要算法和分类

从强化学习的几个元素的角度划分的话，方法主要有下面几类：

* Policy based, 关注点是找到最优策略。
* Value based, 关注点是找到最优奖励总和。
* Action based, 关注点是每一步的最优行动。

我们可以用一个最熟知的旅行商例子来看，

我们要从 A 走到 F，每两点之间表示这条路的成本，我们要选择路径让成本越低越好：



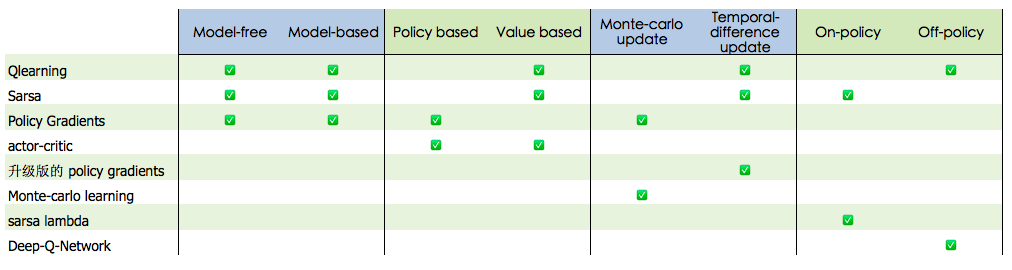
那么几大元素分别是：

* states ，就是节点 {A, B, C, D, E, F}
* action ，就是从一点走到下一点 {A -> B, C -> D, etc}
* reward function ，就是边上的 cost
* policy，就是完成任务的整条路径 {A -> C -> F}

有一种走法是这样的，在 A 时，可以选的 (B, C, D, E)，发现 D 最优，就走到 D，此时，可以选的 (B, C, F)，发现 F 最优，就走到 F，此时完成任务。   
这个算法就是强化学习的一种，叫做 **epsilon greedy**，是一种 **Policy based** 的方法，当然了这个路径并不是最优的走法。

#### 此外还可以从不同角度使分类更细一些：

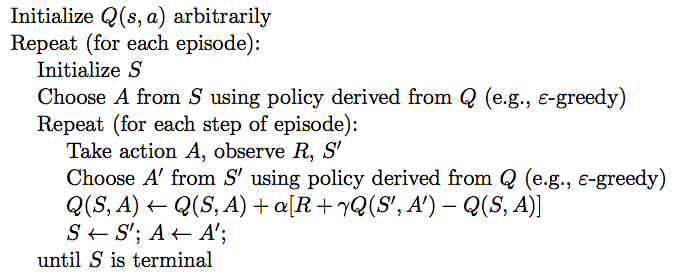
如下图所示的四种分类方式，分别对应着相应的主要算法：



* **Model-free**：不尝试去理解环境, 环境给什么就是什么，一步一步等待真实世界的反馈, 再根据反馈采取下一步行动。
* **Model-based**：先理解真实世界是怎样的, 并建立一个模型来模拟现实世界的反馈，通过想象来预判断接下来将要发生的所有情况，然后选择这些想象情况中最好的那种，并依据这种情况来采取下一步的策略。它比 Model-free 多出了一个虚拟环境，还有想象力。
* **Policy based**：通过感官分析所处的环境, 直接输出下一步要采取的各种动作的概率, 然后根据概率采取行动。
* **Value based**：输出的是所有动作的价值, 根据最高价值来选动作，这类方法不能选取连续的动作。
* **Monte-carlo update**：游戏开始后, 要等待游戏结束, 然后再总结这一回合中的所有转折点, 再更新行为准则。
* **Temporal-difference update**：在游戏进行中每一步都在更新, 不用等待游戏的结束, 这样就能边玩边学习了。
* **On-policy**：必须本人在场, 并且一定是本人边玩边学习。
* **Off-policy**：可以选择自己玩, 也可以选择看着别人玩, 通过看别人玩来学习别人的行为准则。

#### 主要算法有下面几种，今天先只是简述：

**1. Sarsa**



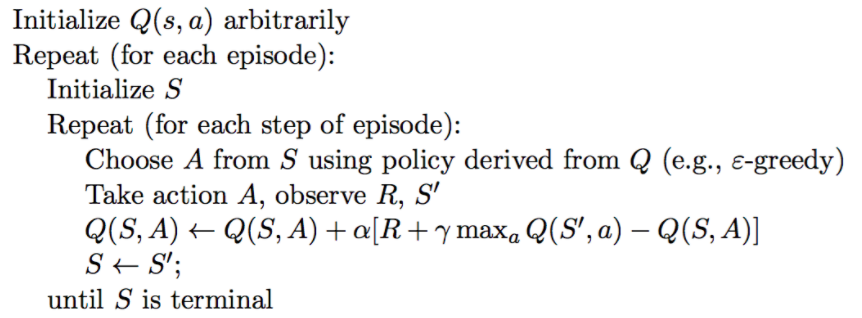
Q 为动作效用函数（action-utility function），用于评价在特定状态下采取某个动作的优劣，可以将之理解为智能体（Agent）的大脑。

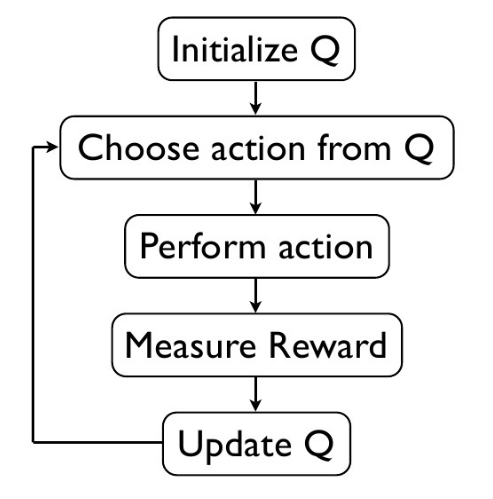
SARSA 利用马尔科夫性质，只利用了下一步信息, 让系统按照策略指引进行探索，在探索每一步都进行状态价值的更新，更新公式如下所示：

http://upload-images.jianshu.io/upload_images/1667471-e1a06de23d965061.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240

s 为当前状态，a 是当前采取的动作，s’ 为下一步状态，a’ 是下一个状态采取的动作，r 是系统获得的奖励， α 是学习率， γ 是衰减因子。

**2. Q learning**





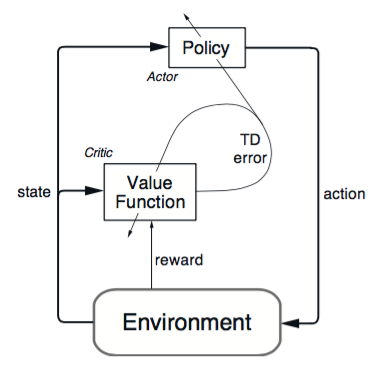
Q Learning 的算法框架和 SARSA 类似, 也是让系统按照策略指引进行探索，在探索每一步都进行状态价值的更新。关键在于 Q Learning 和 SARSA 的更新公式不一样，Q Learning 的更新公式如下：

http://upload-images.jianshu.io/upload_images/1667471-810b358d941dc723.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240

**3. Policy Gradients**

系统会从一个固定或者随机起始状态出发，策略梯度让系统探索环境，生成一个从起始状态到终止状态的状态-动作-奖励序列，s1,a1,r1,.....,sT,aT,rT，在第 t 时刻，我们让 gt=rt+γrt+1+... 等于 q(st,a) ，从而求解策略梯度优化问题。

**4. Actor-Critic**



算法分为两个部分：Actor 和 Critic。Actor 更新策略， Critic 更新价值。Critic 就可以用之前介绍的 SARSA 或者 Q Learning 算法。

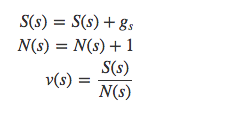
**5. Monte-carlo learning**

用当前策略探索产生一个完整的状态-动作-奖励序列:   
s1,a1,r1,....,sk,ak,rk∼π

在序列第一次碰到或者每次碰到一个状态 s 时，计算其衰减奖励:

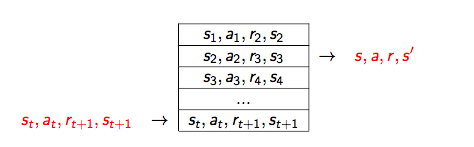
http://upload-images.jianshu.io/upload_images/1667471-7dc4922342e42a45.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240

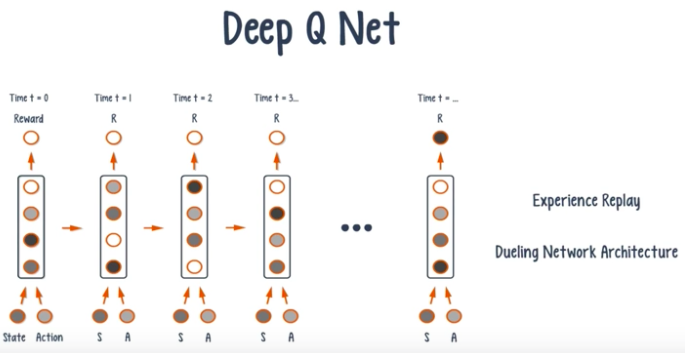
最后更新状态价值:



**6. Deep-Q-Network**

DQN 算法的主要做法是 Experience Replay，将系统探索环境得到的数据储存起来，然后随机采样样本更新深度神经网络的参数。它也是在每个 action 和 environment state 下达到最大回报，不同的是加了一些改进，加入了经验回放和决斗网络架构。



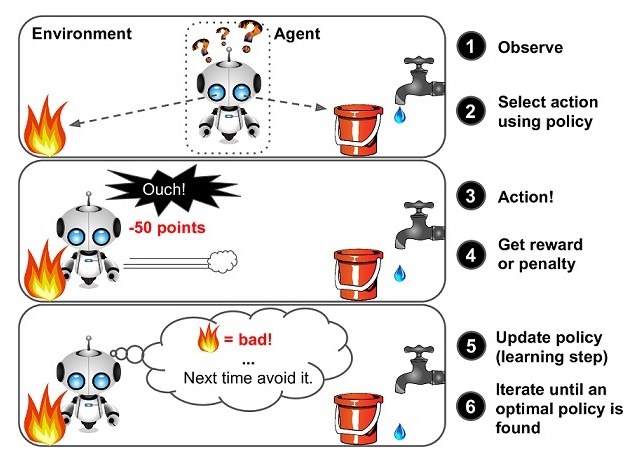


#### 4. 应用举例

强化学习有很多应用，除了无人驾驶，AlphaGo，玩游戏之外，还有下面这些工程中实用的例子：

**1. Manufacturing**

例如一家日本公司 Fanuc，工厂机器人在拿起一个物体时，会捕捉这个过程的视频，记住它每次操作的行动，操作成功还是失败了，积累经验，下一次可以更快更准地采取行动。



**2. Inventory Management**

在库存管理中，因为库存量大，库存需求波动较大，库存补货速度缓慢等阻碍使得管理是个比较难的问题，可以通过建立强化学习算法来减少库存周转时间，提高空间利用率。

**3. Dynamic pricing**

强化学习中的 Q-learning 可以用来处理动态定价问题。

**4. Customer Delivery**

制造商在向各个客户运输时，想要在满足客户的所有需求的同时降低车队总成本。通过 multi-agents 系统和 Q-learning，可以降低时间，减少车辆数量。

**5. ECommerce Personalization**

在电商中，也可以用强化学习算法来学习和分析顾客行为，定制产品和服务以满足客户的个性化需求。

**6. Ad Serving**

例如算法 LinUCB （属于强化学习算法 bandit 的一种算法），会尝试投放更广范围的广告，尽管过去还没有被浏览很多，能够更好地估计真实的点击率。

再如双 11 推荐场景中，阿里巴巴使用了深度强化学习与自适应在线学习，通过持续机器学习和模型优化建立决策引擎，对海量用户行为以及百亿级商品特征进行实时分析，帮助每一个用户迅速发现宝贝，提高人和商品的配对效率。还有，利用强化学习将手机用户点击率提升了 10-20%。

**7. Financial Investment Decisions**

例如这家公司 Pit.ai，应用强化学习来评价交易策略，可以帮助用户建立交易策略，并帮助他们实现其投资目标。

**8. Medical Industry**

动态治疗方案（DTR）是医学研究的一个主题，是为了给患者找到有效的治疗方法。 例如癌症这种需要长期施药的治疗，强化学习算法可以将患者的各种临床指标作为输入 来制定治疗策略。

<https://www.mql5.com/zh/articles/3856>

<http://cdmd.cnki.com.cn/Article/CDMD-10701-1015437700.htm>

<https://www.mql5.com/zh/articles/4777>

<https://blog.csdn.net/u013390476/article/details/50925347>