参考知乎<https://zhuanlan.zhihu.com/c_1139186891151724544>

**基于值（value-based）的算法，需要先构建出一个Q表，之后根据Q表，使用贪婪策略来选择动作，然后获得下一状态和下一状态下的动作，再更新Q值。**

# 1、Q-Learning

指在某一时刻的状态state下，采取动作action能够获取的收益的期望。Q-Learning的本质也即是学习Q值，生成状态-动作Q表格。一般应用于有限的状态和动作空间。超参数控制学习的速度，表示未来的期望占有的权重，如果比较小，那么会更加看重当前选择的回报。多次循环之后，Q表格会逐渐收敛。也即是长远来看，当前的状态-动作Q值已经达到了稳定，不会再更新。此时对于每一种状态其值最大的动作即是最佳的选择。

Initialize arbitrarily

For each episode:

Initialize <s>

For each step of episode:

Choose <a> for <s> using policy derived from <action space> (e.g., -greedy)

Act <a>, observe <r>, <>

Until <s> is terminal or step over

# 2、SARSA（state, action, reward, state, action）

SARSA与Q-Learning非常相似，其区别在于SARSA同时为当前状态和下一状态采用策略选择动作，并且在更新当前值时采用策略选择好的动作的值作为未来预期的收益期望(而Q-Learning是使用下一状态的最大Q值作为预期收益期望)。每当一次值更新完成后，立刻将下一状态和通过策略选择的下一状态的动作当作要处理的状态-动作对。也即是因为同时获得用于下一次处理的状态-动作对，故称为SARSA。

Initialize arbitrarily

For each episode:

Initialize <s>

Choose <a> for <s> using policy derived from <action space> (e.g., -greedy)

For each step of episode:

Act <a>, observe <r>, <>

Choose <> for <> using policy derived from <action space> (e.g., -greedy)

Until <s> is terminal or step over

# 3、SARSA-

在SARSA的基础上进行了改进，主要在于增加了状态-动作对的E值，用于表示状态下每一个选择的重要程度。对于某一状态来说，被选择的动作所对应的E值会增大，而长时间没有被选择的动作所对应的E值会衰减，其衰减系数即由控制。同时，E值也参与状态-动作Q值的更新，且每一步都会更新全部的状态-动作Q值（其E值衰减的很小，更新幅度不大）。这种SARSA-的做法，考虑到了所有的Q值的更新，即是是那些很少被选择的动作。每个episode结束后，如果达到最终状态，那么更新完的E表格代表衡量状态-动作对实现最终的状态的贡献大小。

Initialize arbitrarily

For each episode:

, for all <S>, <A>

Initialize <s>, <a>

For each step of episode:

Act <a>, observe <r>, <>

Choose <> for <> using policy derived from <action space> (e.g., -greedy)

or

For all <S>, <A>:

Until <s> is terminal or step over

# 4、Deep-Q-Network

以上三种强化学习的方式，基本都针对有限状态和动作空间为离散空间，通过不断迭代更新状态-动作Q表格，获得最终收敛后的每种状态下的最优动作。而Deep-Q-Network（DQN）和上面三种方法一样，都是value-based（在这里指基于Q值）方法，不过DQN结合深度网络，通过神经网络来确定动作的选择。同时对每个batch采用梯度下降法更新神经网络的参数。DQN是一种状态转换和Q值更新分开的方法。在处理大型策略问题上更为有效。

一般而言，DQN分为两个步骤：经验存储和经验回放。DQN用到的神经网络包括：预测网络和目标网络。经验存储用于收集足够多的，此时采用预测网络来获得动作action，并从环境中获得奖励和下一次的状态；经验回放是指每次从经验池中取出一个batch的，通过预测网络获得原始值，通过目标网络结合奖励获得目标值，最后通过梯度下降法求损失函数，更新预测网络的参数。目标网络也即是从长远来看的长期收益期望计算函数，在每过一段时间后更新目标网络的参数。

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights

Initialize target action-value function with weights

For each episode:

Initialize <s>

For each step of episode:

With probability select a random action <a>

otherwise select <a>

Execute action <a> in emulator and observe reward <r> and next state <>

Store transition in D

When the memory D has enough data, do this:

Sample random minibatch of transitions from D

Set

Perform a gradient descent step on with respect to the network parameters

Every C step reset

**基于策略（policy-based）的方法直接学习策略，更新策略，而不用再构造状态-动作Q表格，显得更为直接。**

# 5、Policy Gradients

策略梯度（Policy Gradients）是一种最基本policy-based的方法。最初提出的Policy Gradients叫做reinforce。这其中的思想就是，用一个参数为的神经网络作为策略网络，去拟合策略。先用初始化的策略，在环境中运行，之后获得大量的数据，之后利用这些数据，使用公式去更新参数，之后就获得了一个更加好的策略，之后再使用更新过的策略去和环境互动，获得数据，再更新参数，不断循环，直到变得收敛稳定，就不再更新参数了。

Policy Gradients方法采用策略网络给出每个状态可能采取的动作的概率，然后用一个episode内的所有来更新策略网络的参数,其中最终的奖励要通过计算当前状态的即时收益加上之后的衰减系数乘以之后的收益获得。经过几个episode后，策略网络逐渐收敛稳定。需要注意的一点是，Policy Gradients只能采用回合制更新，而不能单步更新。

Initialize arbitrarily

For each episode do

For do

Get initial state <>

Choose <> by probability from ,

Execute action <> in emulator and observe reward <> and next state <>

Store in memory pool

Get all , , from the memory pool, and clean memory pool

for :

Calculate discount reward by

Normalize for

Update by use gradient descent.

# 6、Actor-Critic

在这个算法中，我们需要两个网络，其中一个是actor网络进行PG算法去选择动作，一个是critic网络进行DQN算法去评判动作。很明显，与PG算法的不同在于多了一个动作评判网络。Critic network接收当前状态、奖励与下一个状态，即，该网络通过学习获得当前状态的价值和下一状态的价值，然后把评判的结果传递给actor网络并更新自身网络参数。Actor网络仍然执行PG算法更新网络参数。这样， 可以做到单步更新或batch更新。也即是使用critic network的评判结果代替PG算法中的奖励。这样相互执行动作和评判动作，最后两个网络都趋于收敛稳定。

Initialize ， arbitrarily for action network and critic network

For each episode:

Initialize one state or a batch state <s>

For each step of episode:

Choose <a> by probability from ,

Execute action <a> in emulator and observe reward <r> and next state <>

Passing to critic network

Get critic result by and pass to action network

Update by use gradient descent

Update by use gradient descent.

# 7、DDPG(Deep Deterministic Policy Gradient)

DDPG是一个基于Actor Critic结构的算法，所以DDPG也具有Actor网络和Critic网络。DDPG相比较于普通AC算法的优点在于DDPG算法是一个确定性策略的算法，而AC是一个不确定策略的算法。就是说AC仅仅是指出Actor所做的动作是好还是不好，但是 DDPG算法不仅仅是指出这个动作好还是不好，而提出哪个动作才是最好的。该算法一共有四个网络，分别是actor network，用于生成当前状态对应的动作，target-actor network，用于生成下一状态对应的动作，critic network，用于对当前状态-动作进行评判，target-critic network，用于对下一状态-动作进行评判。其中两个target network的更新方式为软更新。

Randomly initialize critic network and actor with weights and

Initialize target network and with weights ,

Initialize replay buffer

For each episode:

Initialize a random process noise for action exploration

For do:

Get <> from environment

Select action <> according to the current policy and exploration noise

Execute action <> in emulator and observe reward <> and next state <>

Store transition in

Sample a random minibatch of transitions from

Set

Update critic by minimizing the loss:

Update the actor policy using the sampled gradient:

Update the target networks: