Deep Q Network

2019年7月25日 9:25

blog：https://zhuanlan.zhihu.com/p/21421729

一：Q-learning +Deep Learning

（来自 <https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/4-2-DQN2/>）

算法思想：来源NIPS2013

Algorithm 1: deep Q-learning with experience replay.

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 $a_t=\text{argmax}_a Q(\phi(s_t),a;$ DQN 算法更新 (Tensorflow)

    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 $(y_j-Q(\phi_j,a_j;\theta))^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q}=Q$

  End For

End For

损失函数：

$$L(\theta)=E_{s,a,r,s'}[(Q(s,a|\theta)-y)^2], \qquad \text{where } y=r+\gamma \max_{a'} \overline{Q}(s',a'|\overline{\theta})$$

其中 $\overline{Q}(s',a'|\overline{\theta})$ 表示目标网络，其参数更新与 θ 不同步（滞后）

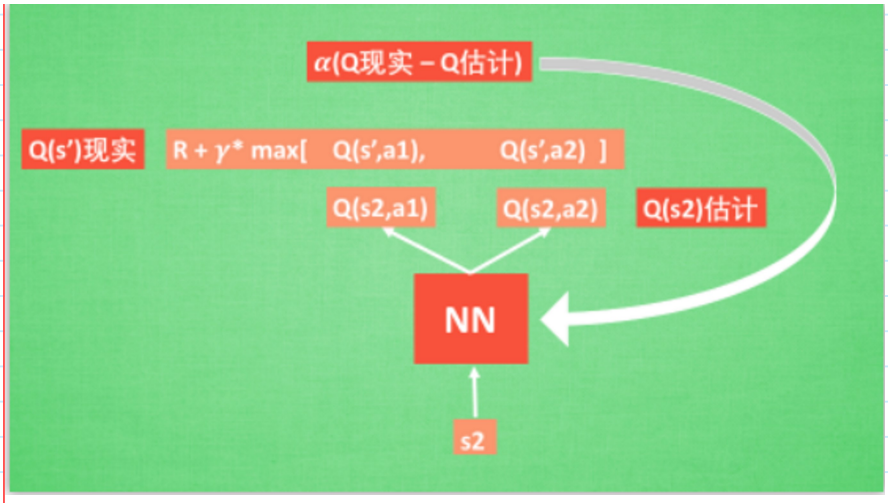
方式1：input state and action to NN

outout Q and a.value

方式2：input state to NN

output a1.value and a2.value

神经网络更新的方式



核心技巧：

1)Experience replay 经验回放

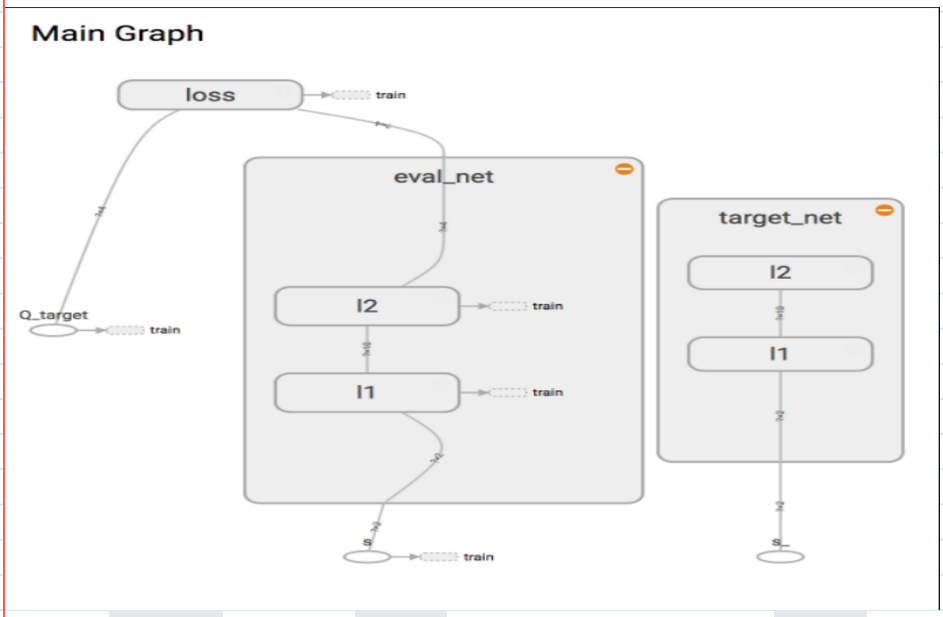
2)Fixed Q-targets 目标网络

二：算法更新

记忆库（用于重复学习）、神经网络计算Q值（action value）、暂时冻结q_target（Q现实）（切断相关性） off-policy

三：神经网络

建立两个神经网络



1）其中target_net 用于预测 q_target 值, 他不会及时更新参数. eval_net 用于预测 q_eval, 这个神经网络拥有最新的神经网络参数. 不过这两个神经网络结构是完全一样的, 只是里面的参数不一样。

2）两个神经网络是为了固定住一个神经网络 (target_net) 的参数, target_net 是 eval_net 的一个历史版本, 拥有 eval_net 很久之前的一组参数, 而且这组参数被固定一段时间, 然后再被 eval_net 的新参数所替换.

而 eval_net 是不断在被提升的。所以eval_net的参数是要通过训练得到的，target_net的参数不用训练，只需要建立相同的网络结构，所有参数由eval_net赋值即可。

3）一个神经网络得到的是Q估计，一个神经网络得到的是Q现实。

四、Q表的更新

存储Q值，最简单的想法就是用矩阵，一个s一个a对应一个Q值，所以可以把Q值想象为一个很大的表格，横列代表s，纵列代表a，里面的数字代表Q值：

	a1	a2	a3	a4
s1	Q(1,1)	Q(1,2)	Q(1,3)	Q(1,4)
s2	Q(2,1)	Q(2,2)	Q(2,3)	Q(2,4)
s3	Q(3,1)	Q(3,2)	Q(3,3)	Q(3,4)
s4	Q(4,1)	Q(4,2)	Q(4,3)	Q(4,4)

1）初始化Q表

	a1	a2	a3	a4
s1	0	0	0	0
s2	0	0	0	0
s3	0	0	0	0
s4	0	0	0	0

2）假设我们选择a2动作，然后得到的reward是1，并且进入到s3状态，接下来我们要根据这个公式来更新Q表的相应项

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \lambda \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

3）用神经网络来表示Q值，实现Q-Network

把目标Q值当作带标签的数据：

$$R_{t+1} + \lambda \max_a Q(S_{t+1}, a)$$

那么损失函数就变成了：

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

这里采用了均方误差，也可以采用其他方式度量。

一个是训练神经网络（eval_net)的输出,一个是Q-learning生成的样本

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。

五、经验回放

最初通过随机策略产生样本sample{s,a,r,s_}，然后把样本通过经验池的方式存储起来，当数据量积累到一定程度，随机抽出样本，这样可以打破样本之间的关联性。类似于在回忆中学习。

六、目标网络

前期通过Q-learning不断去生成样本数据，当样本容量达到一定程度后把样本拿去训练一个神经网络（eval-network)，使得训练出来的神经网络(实际上就是Q（s,a)=f(s,a,w)）(神经网络的输入就是state)能够

拟合Q。然后把这个神经网络复制给一个跟他一模一样的神经网络(target_net)，然后经验池清空、重填，并不断训练神经网络（evel_net)。最终使得能够输入一个state,经过一个神经网络，输出一个action value function。