#RL

Deep Q Network

使用神经网络计算 Q 值

以往，我们通过 $Q(s,a)$ 对指定状态 s 下的行动 a 做出评价（输出 Q 值），现在，我们的状态空间变得极大，因此，我们不能再使用有限维的表格了。我们制造一个神经网络，使之输入 s,a，就输出一个 $Q(s,a;\theta)$，其中，θ 是神经网络的参数网络的输出表征了 Future reward.

使用经验池存储过去的经验

E.g.初始化一个容量 $N=5$ 的经验池，设置回放阈值 $n=3$，再设置 mini-batch 的容量是 $b=2$。

- 每次向经验池中存储新的经验
- 如果 $t \geq n$,那么将从经验池中随机选取一个 minibatch 更新网络
- 如果经验池填满，则推出旧的经验

DQN 总算法

初始化容量为 N 的 Buffer, 初始化一个神经网络 Q,
一共游玩 M 个回合：
表征目前的图像，送入神经网络中

- 对于每个回合里的每一个 timestep:
  - 以 ϵ 的概率随机选择动作 a_t，否则，以贪心策略选择动作 $\max_a Q(\phi(s),a;\theta)$
  - 执行选择的动作 a_t，得到奖励 r_t
  - 表征下一帧图像 $\phi(s_{t+1})$
  - 获得第一个经验 (s_t,a_t,r_t,s_{t+1}) →向 Buffer 中插入经验
  - 从 Buffer 中抽取 minibatch (s_j,a_j,r_j,s_{j+1})
  - 计算 y_j，若 y_j 为结束状态，则 $y_j=r_j$，否则，利用神经网络预测未来的收益，取 $y_j=r_j+\gamma\max_{a'}Q(s_j,a';\theta)$ 。
  现在，你已经知道了对于一个历史情况，它的真实收益可能是多少。
  - 计算误差 $||y_j-Q(s_j,a_j;\theta)||$，使之最小化（梯度下降）
  循环游玩多个回合，直至结束

DQN 的特性是——利用经验池存储过往经验，利用历史数据训练今天的网络。在实际中，buffer 不一定是空的，可能是先塞入 $1/2$ 到 $1/3$ 来预训练。
这样的方式可能导致神经网络缺乏与环境的交互。还有一个问题：由于神经网络参数的不断更新，而经验池中的数据都是在旧有的参数下得到的，这会导致经验池中的参数越来越不能用了！

优点：利用了连续的状态空间，收敛更快，更稳定
缺点：**动作空间仍然是离散的！**，只有一个 Q Network，同时用于采样和计算目标值 y_i，可能导致一些不稳定，尤其是可能导致对于 Q 值的过大估计。这主要是有两个因素导致的：

- DQN 更新过程中的两次最大值操作将导致高估，尤其是当噪声出现的时候
- 用自身的估计更新自身，从而导致自举 (Bootstrap，左脚踩右脚上天)的问题
  当然，要是都以相同的幅度偏高，倒也没问题，然而事实上，这种“都偏高”的情况很少

Double DQN

现在，使用两个完全相同的网络解决对于 Q 的过高估计问题，仅仅使用 $Q(s,a,\theta)$ 控制 Agent，而使用 $Q(s,a,;\theta^-)$ 更新 y
而 Q^- 始终比 Q “慢半拍”：

- Hard Update: 定期把 Q 的参数传给 Q^-
- Soft Update : $\theta^- = \tau\theta + (1-\tau)\theta^-$
  Q^- 网络被称为“目标网络”，这**大大延缓**了过高估计的到来，此外，Double DQN 还做了一件微不足道的工作：

$$Y_t = R_{t+1} + \gamma Q'(S_{t+1}, \arg\max_a Q; \theta)$$

  正是因为这个工作，动作估计和动作选择是完全分离的！

Dueling DQN

定义一个最优优势函数：表征了在最优策略下选择 a 相对于 Baseline 的差距

$$A^\star(s,a) = Q^\star(s,a) - V^\star(s)$$

于是，

$$Q(s,a) = V(s) - A(s,a)$$

因此，Q 的估计被拆成了对于 V 的估计和对于优势值的估计
这样有诸多问题，例如，训练过程中，V 和 A 可能相抵（一个增加，一个减少，结果没变），等等，因此，我们将 Q 减去一项：

$$Q(s,a) = V(s) - A(s,a) - \max_a A(s,a)$$

这样，只要 V 和 A 变一个，Q 就会变

Distributional Q -function

上面的所有做法仅仅建模了 Q 的期望，没有建模 Q 的分布，这使得我们无法看到其中的风险。因此，我们使用直方图近似分布。一个 s 被送入 Q 网络，每个动作上都输出一个直方图

Noisy Network

向全连接层中加入参数化噪声，使得 agent 与环境的交互中加入一点不确定性，其中，独立的高斯噪声在 $y = wx + b$ 的 w 和 b 中都加噪声；分解的高斯噪声在

Prioritized Experienced Replay

所有的经验权重都一样吗？不！ 有些经验会使得 Agent 更高效地学习！
我们认为，一个经验的 Q 和目前的差距就代表了这个经验带来的信息

RAINBOW

以上算法的集大成

PYTHON

```python
a = 3
print(a)
```

PYTHON

```python
plt.plot([2,3],[4,5])
plt.show()
```