**强化学习理论学习与代码实现**

# 深度强化学习-Nature深度Q网络(Nature DQN)

在上一章中我们讨论了DQN(NIPS 2013)[1]的算法原理，该算法虽然在Atari游戏中表现出了惊人的实力，但也不是没有问题，今天先讨论第一个问题。

注意到DQN（NIPS 2013）里面使用的目标Q值的计算公式如下：



$$

y\_{j}=\left\{\begin{array}{ll}{r\_{j}} & {\text { for terminal } \phi\_{j+1}} \\ {r\_{j}+\gamma \max \_{a^{\prime}} Q\left(\phi\_{j+1}, a^{\prime} ; \theta\right)} & {\text { for non-terminal } \phi\_{j+1}}\end{array}\right.

$$

这里目标Q值的计算使用到了利用当前要训练的Q网络参数计算得到的 $Q\left( {{\phi \_{j + 1}},{a^\prime };\theta } \right)$，而实际上又希望通过yj来更新Q网络参数，正是因为两者相互依赖、相互影响，导致当前Q值和目标Q值的相关性太强，不利于算法的收敛。

因此在2015年又是Mnih这位大神提出了一个改进版的DQN[2]，由于文章发表在《Nature》，一般将该算法称为“Nature DQN”。相比较于DQN(NIPS 2013)，Nature DQN增加了目标网络，通过双网络结构来减少目标Q值计算与待更新Q网络参数之间的依赖关系，从而极大地提高了DQN算法的稳定性。

## Nature DQN网络模型

Nature DQN使用如图1所示的深度卷积神经网络来参数化近似值函数 $Q\left(s, a ; \theta\_{i}\right)$，其中${{\theta \_i}}$为Q网络在第i次迭代时的参数（即权重）。*θ*i为了能够执行经验回放，将代理每一时间步t的经验 $e\_{t}=\left(s\_{t}, a\_{t}, r\_{t}, s\_{t+1}\right)$储存于数据集 $\mathcal{D}=e\_{1}, \dots, e\_{t}$中，在学习过程中，对从经验样本池中随机均匀抽取的经验样本$\left(s, a, r, s^{\prime}\right) \sim U(D)$采用Q学习更新。第i次迭代时Q学习更新使用如下的损失函数：

 （1）

$$

L\_{i}\left(\theta\_{i}\right)=\mathbb{E}\_{\left(s, a, r, s^{\prime}\right) \sim \mathrm{U}(D)}\left[\left(r+\gamma \max \_{a^{\prime}} Q\left(s^{\prime}, a^{\prime} ; \theta\_{i}^{-}\right)-Q\left(s, a ; \theta\_{i}\right)\right)^{2}\right]

$$

其中γ为折扣因子，θi为Q网络第i次迭代时的参数，θi-为第i次迭代用于计算目标的网络参数。目标网络参数θi-仅每隔C步才会使用Q网络参数（θi）进行更新，而在更新之前都将保持不变。对损失函数求权重上的导数可以得到如下梯度：

 （2）

$$

\nabla\_{\theta\_{i}} L\left(\theta\_{i}\right)=\mathbb{E}\_{s, a, r, s^{\prime}}\left[\left(r+\gamma \max \_{a^{\prime}} Q\left(s^{\prime}, a^{\prime} ; \theta\_{i}^{-}\right)-Q\left(s, a ; \theta\_{i}\right)\right) \nabla\_{\theta\_{i}} Q\left(s, a ; \theta\_{i}\right)\right]

$$

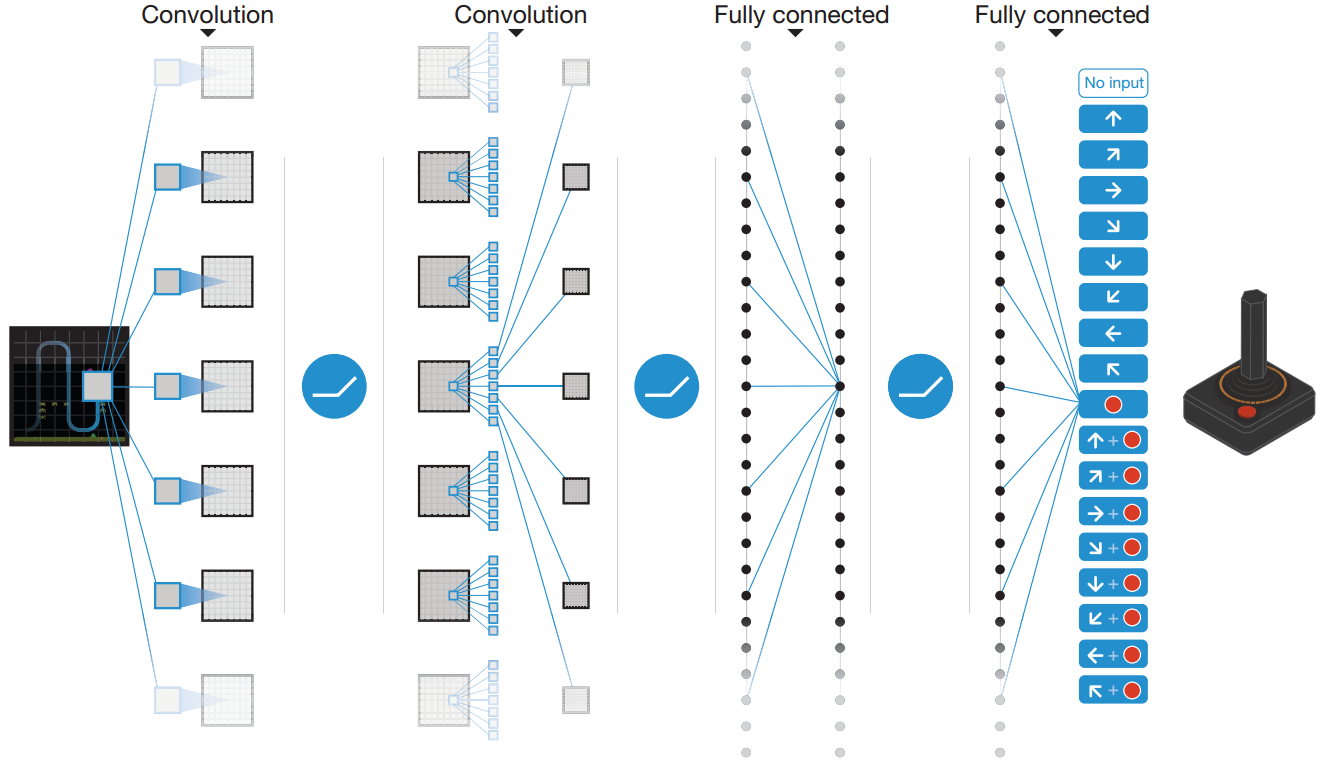


图1 卷积神经网络的示意图

## Nature DQN算法

Nature DQN相比较于标准的在线Q学习主要进行了两方面的改进，以适用于训练大规模神经网络且不会发散。

第一个就是经验回放技术，其原理已经在上一章中有详细介绍。为进一步提高神经网络方法的稳定性，在线Q-learning的第二个修改是在Q学习更新中使用一个单独的网络来生成目标yj，更准确地说，每C步克隆Q网络以获得一个目标网络$\hat{Q}$，并使用$\hat{Q}$生成Q学习目标yj。这个修改可以使得算法比标准的在线Q学习更加稳定，因为在标准的Q学习中，更新在增大Q(st,at)的同时也会增大所有a的Q(st+1,a)，因此也会导致增大目标yj，而这非常有可能导致策略震荡或发散。使用旧的参数集合生成目标可以在更新Q值和更新影响目标yj之间增加时间上的延迟，从而大大降低震荡或发散的可能性。

同时研究人员也还发现，将通过更新$r + \gamma {\max \_{a'}}Q(s',a';\theta \_i^ - ) - Q(s,a;{\theta \_i})$得到的误差项限制在-1到1是很有帮助的，因为损失函数的绝对值|x|对于x所有的负值均有导数为-1，对于x所有的正值均有导数为1，将平方误差限制在-1到1之间就相当于对位于(-1,1)区间外的误差采用损失函数绝对值，这种形式的误差限制可以进一步提升算法的稳定性。

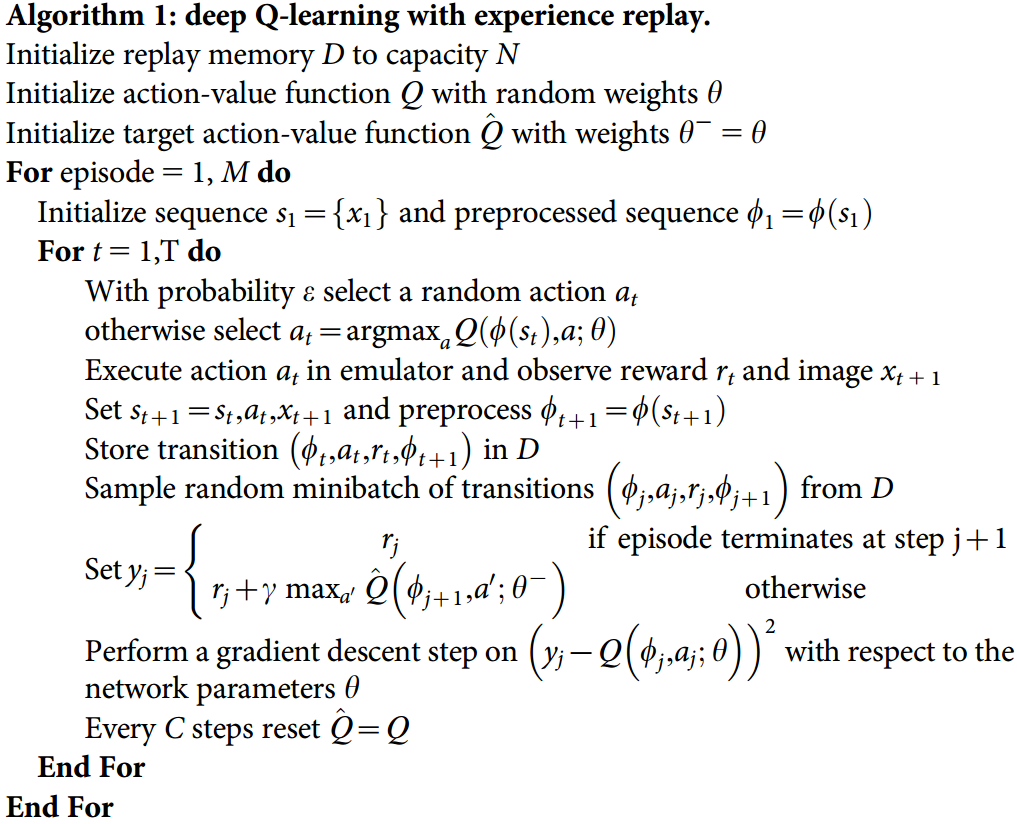


图2 Nature DQN算法流程

Step1：初始化经验池 $\mathcal{D}$，其容量为N（即可存储N个历史样本），用作历史经验回放。

Step2：设状态值函数Q作为预测网络，并随机初始化权重参数θ。

Step3：设状态值函数 $\hat Q$作为目标网络，并初始化权重参数${\theta ^ - }$=θ。

Step4：设定游戏片段总数为M，即代理最多执行游戏次数为M次。

Step5：根据当前状态s1，输入的图像帧为84x84x4的张量，并且并预处理得到Φ1=Φ(s1)，即在状态s1下获得游戏对应的固定序列游戏帧Φ1。

Step6：以概率ε随机选择代理需要执行的动作at，或以概率1-ε执行网络输出最大Q值所对应的动作。

Step7：代理执行动作at，获得环境反馈的奖励rt和下一时间步的游戏图像xt+1。

Step8：基于新的状态st+1=st，at，xt+1，根据Φt+1=Φ(st+1)计算下一时间步的固定序列游戏帧Φt+1。

Step9：将获得的状态转移(Φt,at,rt, Φt+1)存入经验池 $\mathcal{D}$中。

Step10：代理从 $\mathcal{D}$中随机采样最小批量转移。

Step11：计算每一个状态的目标值，代理通过执行动作后的奖励rj来更新Q值作为Q学习算法的目标值。

Step12：根据式（2），基于最小批量样本采用随机梯度下降法更新Q网络权重参数θ。

Step13：每经过C次迭代后，更新目标动作值函数 $\hat Q$的网络参数${\theta ^ - }$为预测网络的参数θ。

## 实验结果

### 参数设置

|  |  |  |
| --- | --- | --- |
| Hyperparameter超参数 | Value值 | Description描述 |
| minibatch size | 32 | 随机梯度下降最小批量采样样本大小 |
| replay memory size | 1000000 | 经验池大小，用于记录历史游戏帧 |
| agent history length | 4 | 代理记录历史游戏帧的长度 |
| target network update frequency | 10000 | 目标网络的更新频率 |
| discount factor | 0.99 | Q学习更新的折扣系数 |
| action repeat | 4 | 代理重复所选动作的次数，取值4表示代理只会每隔4帧进行一次观察 |
| update frequency | 4 | 代理在连续的SGD更新之间所执行的动作数量，即预测网络的更新频率，取值为4表示每相邻两次更细之间代理选择了4个动作 |
| learning rate | 0.00025 | RMSProp使用的学习率 |
| gradient momentum | 0.95 | RMSProp使用的梯度动量 |
| squared gradient momentum | 0.95 | RMSProp使用的平方梯度动量 |
| min squared gradient | 0.01 | RMSProp更新的分母中在平方梯度上加上的常量 |
| initial exploration | 1 | epsilon-贪婪探索中epsilon的初始值 |
| final exploration | 0.1 | epsilon-贪婪探索中epsilon的最终值 |
| final exploration frame | 1000000 | epsilon线性地下降至最终值所需的帧数 |
| replay start size | 50000 | 学习开始前，先运行该帧数的均匀随机策略，得到的经验用于经验池的初始数据 |
| no-op max | 30 | 代理在片段开始时不执行任何动作的最大帧数 |

### 算法效果

从图3中可知，深度强化学习DQN算法在多种Atari游戏中都取得了超越人类平均水平的成绩，远胜于此前提出的任何算法和模型。尤其值得注意的是，在代理的训练过程中并没有融入太多人工制定的游戏规则，而是让代理完全自主地学习、记忆、模仿和试错，这更说明了DQN算法的优越性。

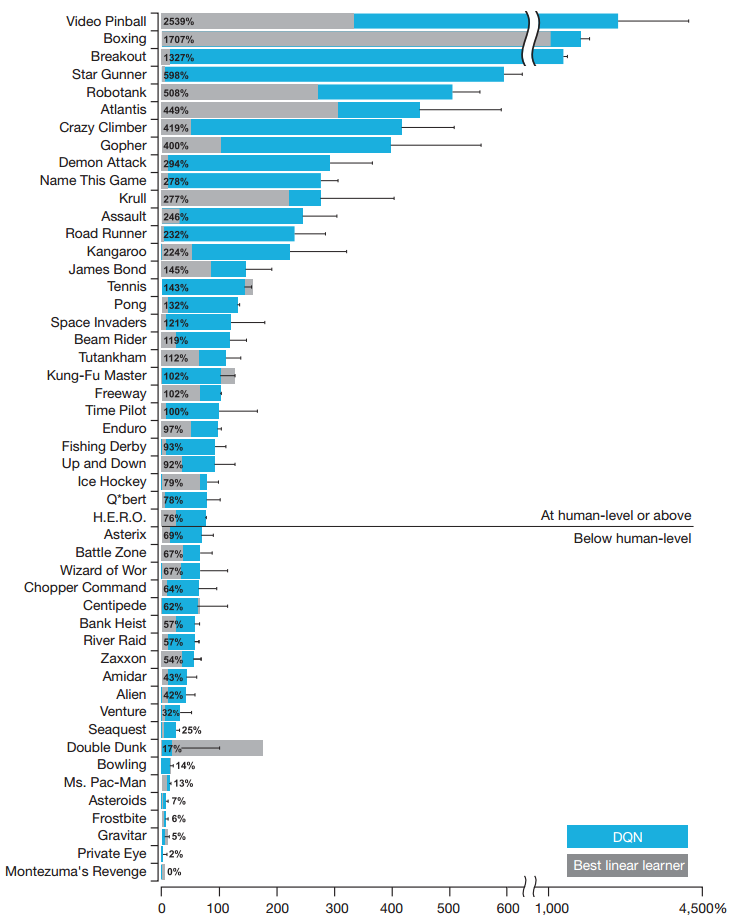


图3 DQN与最优线性学习方法的对比

## 结论

DQN通过引入目标函数、经验回放以及目标网络等技术，成功且有效地结合了深度学习和强化学习，为深度强化学习的发展夯实了基础。但需要注意的是，DQN算法使用单个价值网络，导致其训练效率低，极大地限制了DQN算法的应用范围，为了解决DQN算法本身存在的不足，后续研究者对DQN算法做了大量的改进，提出了更为先进的DQN改良版本，如Double DQN、Prioritized DQN、Dueling DQN等，后续再给大家详细介绍。

参 考 文 献

[1] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing Atari with Deep Reinforcement Learning[J]. Computer Science. 2013.

[2] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Human-level control through deep reinforcement learning[J]. Nature. 2015, 518(7540): 529-533.