法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

第6讲　值函数逼近法

强化学习

主讲人：叶梓

上海交通大学博士

主要研究方向：机器学习、深度学习、人工智能

本章内容

- 维数灾难
- 值函数逼近法概述
- 参数化逼近
- 主要研究问题
- 开源库介绍
- 环境搭建

表格型强化学习方法

- 状态值（状态-行为值）函数可以利用DP，MC或TD的方法计算得到。
- 这时的值函数表达为一个表格。
  - 对于状态值函数，其索引是状态；
  - 对于行为值函数，其索引是状态-行为对。
- 值函数迭代更新的过程实际上就是对这张表进行迭代更新。
- 因此，之前讲的强化学习算法又称为表格型强化学习。
- 对于状态值函数，其表格的维数为状态的个数 $|S|$，其中 S为状态空间。

维数灾难

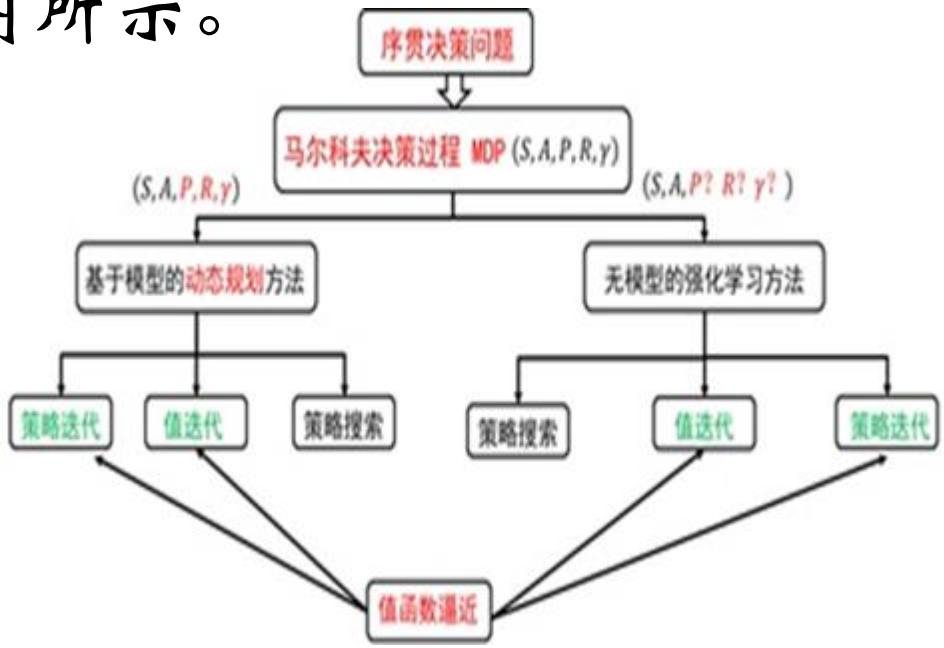
- DP、MC、TD等方法都有一个基本的前提条件，就是状态空间和动作空间是离散的，而且状态空间和动作空间不能太大。
- 考虑一下围棋：

$3^{361}=1.74*10^{172}$

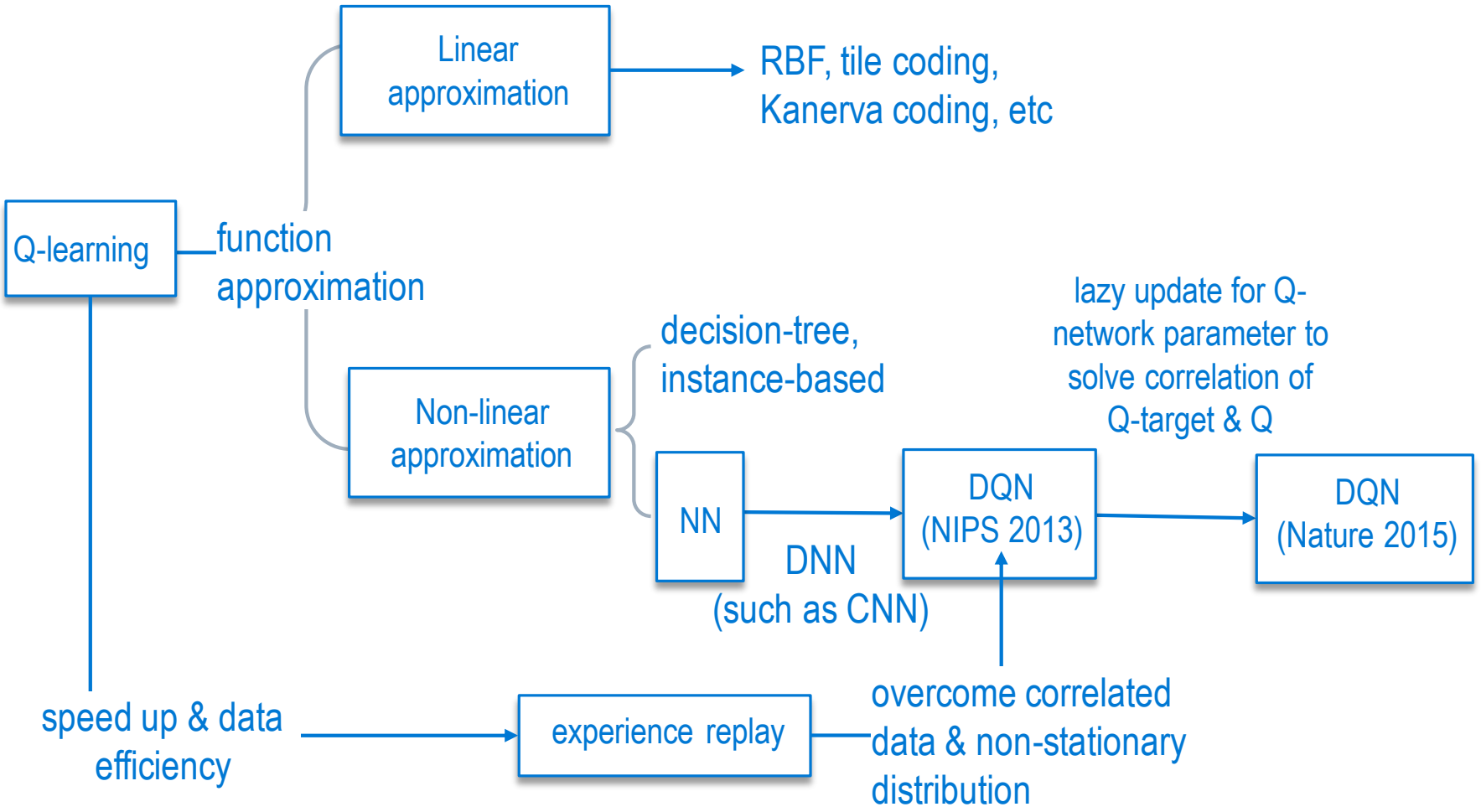
$3^{261}=3.38*10^{124}$

值函数逼近方法

- 若状态空间的维数很大，或者状态空间为连续空间，此时值函数无法用一张表格来表示。
- 这时，我们需要利用函数逼近的方法对值函数进行表示。如图所示。
- 当值函数利用函数逼近的方法表示后，就可以利用策略迭代和值迭代方法了。



基于Q学习的值函数逼近法的演变



值函数逼近方法

- 函数逼近方法可以分为参数逼近和非参数逼近，因此强化学习值函数估计可以分为参数化逼近和非参数化逼近。
- 这里主要介绍参数化逼近。所谓参数化逼近，是指值函数可以由一组参数θ来近似。我们将逼近的值函数写为$\hat{v} = (s, \theta)$。
- 其中参数化逼近又分为：
  - 线性参数化逼近(例如：基函数)
  - 非线性化参数(例如：神经网络)逼近。

与表格型RL的异同点

- 表格型强化学习和值函数逼近方法的强化学习值函数更新时的异同点：

- 表格型强化学习进行值函数更新时，只有当前状态S_t处的值函数在改变，其他地方的值函数不发生改变。

- 值函数逼近方法进行值函数更新时，因此更新的是参数θ，而估计的值函数为$\hat{v} = (s, \theta)$，所以当参数θ发生改变时，任意状态处的值函数都会发生改变。

值函数逼近

与表格法相比，值函数逼近表示目标函数的优点

- 可以降低输入维度，减少计算量
- 可以提高泛化能力，避免过适应（over-fitting）
- 可以使目标函数对于参数可微，使用基于梯度的计算方法

FA分为两大类：线性和非线性的

- 前者一般是用一系列特征的线性组合，它们的权重作为参数。优点自然是计算方便，算法易于实现
- 后者就比如用神经网络（NN），优点是表达能力大大加强，缺点是训练起来也麻烦得多

参数化逼近

- 当逼近的值函数结构确定时，则值函数的逼近就等价于参数的逼近。
- 值函数的更新也就等价于参数的更新。即，需要利用试验数据来更新参数值。
- 先复习一下表格型强化学习值函数更新的公式：

MC法，值函数更新公式为：

$$Q(s,a) \leftarrow Q(s,a) + \alpha(G_t - Q(s,a))$$

TD法，值函数更新公式为：

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

TD(λ)法，值函数更新公式为：

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[G_t^{\lambda} - Q(s,a)\right]$$

- 值函数的更新过程：值函数更新过程是向着目标值函数靠近。

参数化逼近

- 从表格型值函数的更新过程，不难总结出不管是蒙特卡罗方法还是TD方法，都是朝着一个目标值更新的，
  - 这个目标值在蒙特卡罗方法中是G_t，在TD方法中是$r+\gamma Q(s', a')$，TD（λ）中是G_t^λ。
- 函数逼近$\hat{v}=(s,\theta)$的过程是一个有监督学习的过程，其数据和标签对为(S_t, U_t)，其中U_t等价于蒙特卡罗方法中的G_t，TD方法中$r+\gamma Q(s', a')$，以及TD（λ）中的G_t^λ
- 训练的目标函数为：

$$argmin_\theta (q(s,a) - \hat{q}(s,a,\theta))^2$$

V(s)　　V(s)

增量方法与批量方法

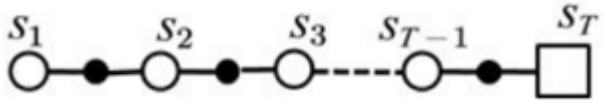
- 值函数更新可分为增量式学习方法和批量学习方法：
  - 增量式方法参数更新过程随机性比较大，尽管计算简单，但样本数据的利用效率并不高。
  - 批量的方法，尽管计算复杂，但计算效率高。
- 随机梯度下降法是最常用的增量式学习方法。
- 由训练的目标函数，可以得到参数的随机梯度更新为：

学习步长　误差　梯度

$$\theta_{t+1} = \theta_t + \alpha \left[U_t - \hat{v}\left(S_t, \theta_t\right) \right] \nabla_\theta \hat{v}\left(S_t, \theta\right)$$

增量式学习方法

- 如果是基于蒙特卡罗方法的函数逼近，
- 给定要评估的策略π，产生一次试验：



- 值函数的更新过程实际是一个监督学习的过程，其中监督数据集从蒙塔卡罗的试验中得到，其数据集为：

$$\langle s_1, G_1 \rangle, \langle s_2, G_2 \rangle, \cdots, \langle s_T, G_T \rangle$$

- 值函数的更新：

$$\Delta\theta = \alpha \left(G_t - \hat{v}\left(S_t, \theta\right) \right) \nabla_\theta \hat{v}\left(S_t, \theta\right)$$

其中α值比较小，在随机梯度下降法中，需要一个能平衡所有不同状态误差的值函数逼近。α值取得比较小可以维持这种平衡。

参数化逼近

- 如果是基于时间差分方法的函数逼近
- TD(0)方法中目标值函数为：

$$U_t = R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \theta\right)$$

- 即目标值函数用到了bootstrapping的方法。此时，要更新的参数θ不仅出现在要估计的值函数中，还出现在目标值函数中。
- 若只考虑参数θ对估计值函数的影响而忽略对目标值函数的影响，这种方法并非完全的梯度法，只有部分梯度，因此称为半梯度法：

$$\theta_{t+1} = \theta_t + \alpha\left[R + \gamma \hat{v}\left(S', \theta\right) - \hat{v}\left(S_t, \theta_t\right)\right] \nabla \hat{v}\left(S_t, \theta_t\right)$$

固定

线性逼近：常用基函数

- 先讨论线性逼近 $\hat{v}(s,\theta)=\theta^T\phi(s)$
- 线性逼近的好处是只有一个最优值，因此可以收敛到全局最优。
- 其中 $\phi(s)$ 为状态s处的特征函数，或者称为基函数。常用的基函数的类型为：
- 多项式基函数，如：

$$(1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, \cdots)$$

- 傅里叶基函数： $\phi_i(s)=\cos(i\pi s),\ s\in[0,1]$

- 径向基函数： $\phi_i(s)=\exp\left(-\frac{\|s-c_i\|^2}{2\sigma_i^2}\right)$

参数更新公式

- 将线性逼近值函数带入随机梯度下降法和半梯度下降法中，可以得到参数的更新公式，如下：
- 蒙特卡罗方法值函数更新为：

$$\begin{aligned}\Delta\theta &= \alpha\left[U_t(s) - \hat{v}(S_t,\theta_t)\right]\nabla\hat{v}(S_t,\theta_t)\\ &= \alpha\left[G_t - \theta^T\phi\right]\phi\end{aligned}$$

- TD(0)线性逼近值函数更新为：

$$\begin{aligned}\Delta\theta &= \alpha\left[R + \gamma\theta^T\phi(s') - \theta^T\phi(s)\right]\phi(s)\\ &= \alpha\delta\phi(s)]\end{aligned}$$

- 正向视角的TD(λ)更新为：

$$\Delta\theta = \alpha\left(G_t^\lambda - \theta^T\phi\right)\phi$$

- 后向视角的TD(λ)更新为：

$$\Delta\theta = \alpha\delta_t E_t \qquad \delta_t = R_{t+1} + \gamma\theta^T\phi(s') - \theta^T\phi(s) \qquad E_t = \gamma\lambda E_{t-1} + \phi(s)$$

参数化逼近：批量的方法

- 批量的方法：是指给定经验数据集

$$LS(\theta) = \sum_{t=1}^{T} \left(v_t^{\pi} - \hat{v}_t^{\pi}(s_t, \theta)\right)^2$$

- 找到最好的拟合函数，

$$D = \left\{ \langle s_1, v_1^{\pi} \rangle, \quad \langle s_2, v_2^{\pi} \rangle, \quad \cdots, \quad \langle s_T, v_T^{\pi} \rangle \right\}$$

- 使得：$\hat{v}(s, \theta)$ 最小可利用线性最小二乘逼近：

$$\Delta\theta = \alpha \sum_{t=1}^{T} \left[v_t^{\pi} - \theta^T \phi(s_t)\right] \phi(s_t) = 0$$

取得最优解的地方，就是参数不再改变的位置

参数化逼近：批量的方法

- 最小二乘蒙特卡罗方法参数为：

$$\theta = \left(\sum_{t=1}^{T} \phi(s_t)\, \phi(s_t)^T \right)^{-1} \sum_{t=1}^{T} \phi(s_t)\, G_t$$

- 最小二乘差分方法为：

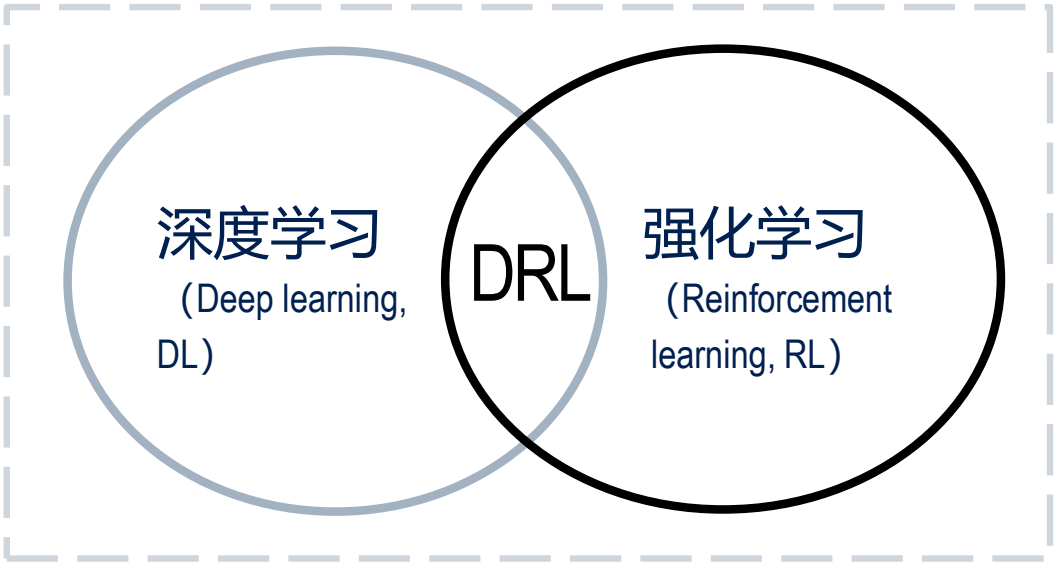
$$\theta = \left(\sum_{t=1}^{T} \phi(s_t)\, (\phi(s_t) - \gamma\phi(s_{t+1}))^T \right)^{-1} \sum_{t=1}^{T} \phi(s_t)\, R_{t+1}$$

- 最小二乘 TD(λ) 方法为：

$$\theta = \left(\sum_{t=1}^{T} \mathrm{E}_t (\phi(s_t) - \gamma\phi(s_{t+1}))^T \right)^{-1} \sum_{t=1}^{T} E_t R_{t+1}$$

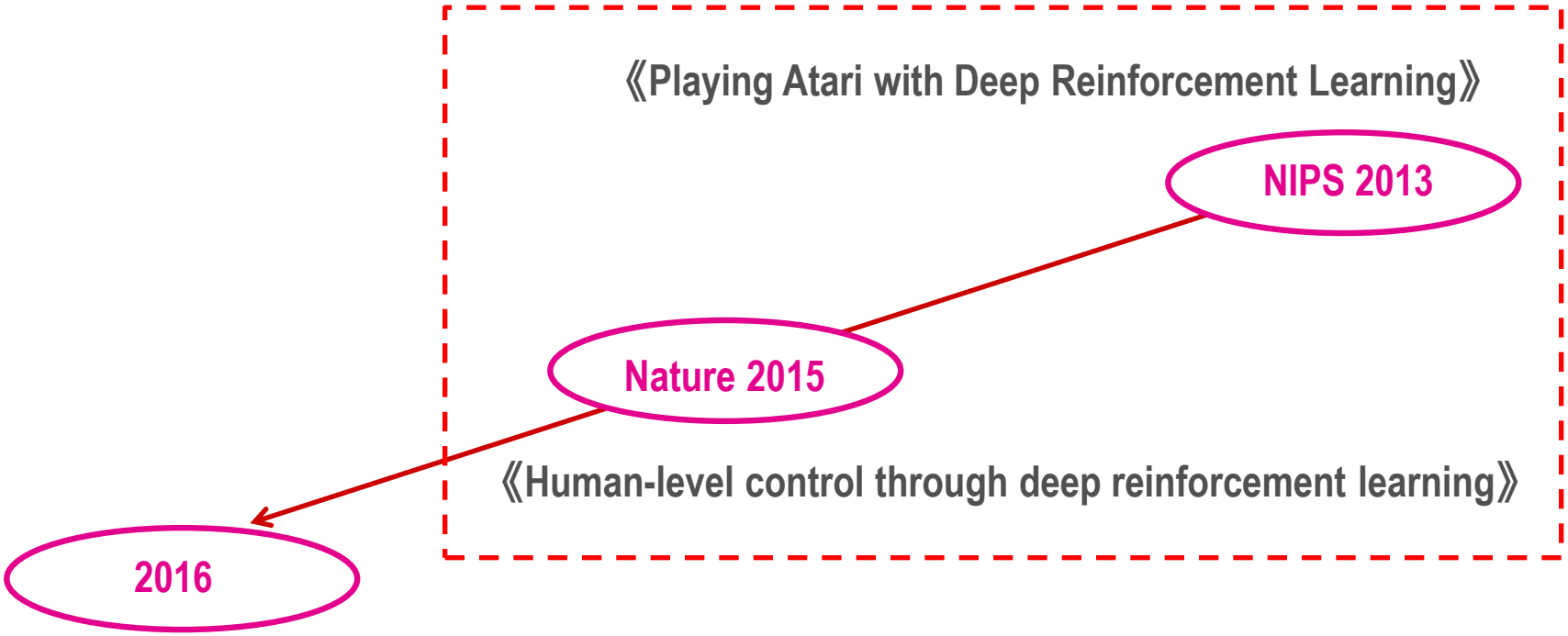
DRL：深度强化学习

- DQN（Deep Q-Learning）可谓是深度强化学习（Deep Reinforcement Learning，DRL）的开山之作，
- 它是将深度学习与强化学习结合起来从而实现从感知（Perception）到动作（Action）的端对端（End-to-end）学习的一种全新的算法。



两篇Nature上的奠基性论文（**DQN和AlphaGo**）使得此交叉方向变得炙手可热，成功地开创了新的方向，既具有极大学术价值也具有很高的商业价值。

DRL：里程碑论文的发表



1. AAAI 2016 《Deep Reinforcement Learning with Double Q-learning》
2. ICLR 2016《Prioritized Experience Replay》
3. ICML 2016《Dueling Network Architectures for Deep Reinforcement》
4. Nature 2016 《Mastering the game of Go with deep neural networks and tree search》

DQN的起源

- 在普通的Q-learning中，当状态和动作空间是离散且维数不高时可使用Q-Table储存每个状态动作对的Q值，而当状态和动作空间是高维连续时，使用Q-Table不现实。
- 利用神经网络逼近值函数的做法在强化学习领域早已有之，通常做法是把Q-Table的更新问题变成一个函数拟合问题，相近的状态得到相近的输出动作。
- 但学者们发现利用神经网络，尤其是深度神经网络去逼近值函数有很大困难，因为常常出现不稳定或不收敛的情况，直到DeepMind出现。
- DeepMind在NIPS 2013上发表DQN V1，后又在Nature 2015上提出DQN V2。

DQN的基本思路

- 前些年涌现的一些成果，比如CNN，极大地增强了DL处理raw pixels数据的信心。DQN中的输入采用的是原始图像数据，这是DQN最有意义的一步。
- DQN的算法对所有游戏是通用的（甚至超参数也是通用的），而这种通用性的重要基础之一就是它能使用raw pixels。
- 同时，它的关键步骤还包括：experience replay。该方法在Long-Ji Lin 1993年的毕业论文中有较详细的介绍，其主要作用是克服经验数据的相关性（correlated data）和非平稳分布（non-stationary distribution）问题。
- 这是NIPS 2013论文中提出的DQN基本思路。

DQN的思路-对Q学习的修改

- DQN中的CNN作用是对在高维且连续状态下的Q-Table做函数拟合。
- 而对于函数优化问题，监督学习的一般方法是先确定Loss Function，然后求梯度，使用随机梯度下降等方法更新参数。DQN是基于Q-Learning来确定Loss Function。
- DQN对Q-learning的修改主要体现在三个方面：
  - DQN利用深度卷积神经网络逼近值函数
  - DQN利用了经验回放对强化学习的学习过程进行训练
  - DQN独立设置了目标网络来单独处理时间差分算法中的TD偏差。

DQN v1的思路-构造目标函数

- DQN则基于Q-Learning来确定Loss Function。
- Q-Learning的更新公式：

$$Q^*(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

- DQN的Loss Function为 $L(\theta) = E[(TargetQ - Q(s,a;\theta))^2]$
- 其中θ是网络参数，目标为 $TargetQ = r + \gamma \max_{a'} Q(s',a';\theta)$
  - Loss Function是基于Q-Learning更新公式的第二项确定的，都是使当前的Q值逼近Target Q值。
- 然后求L(θ)关于θ的梯度，使用SGD等方法更新网络参数θ。

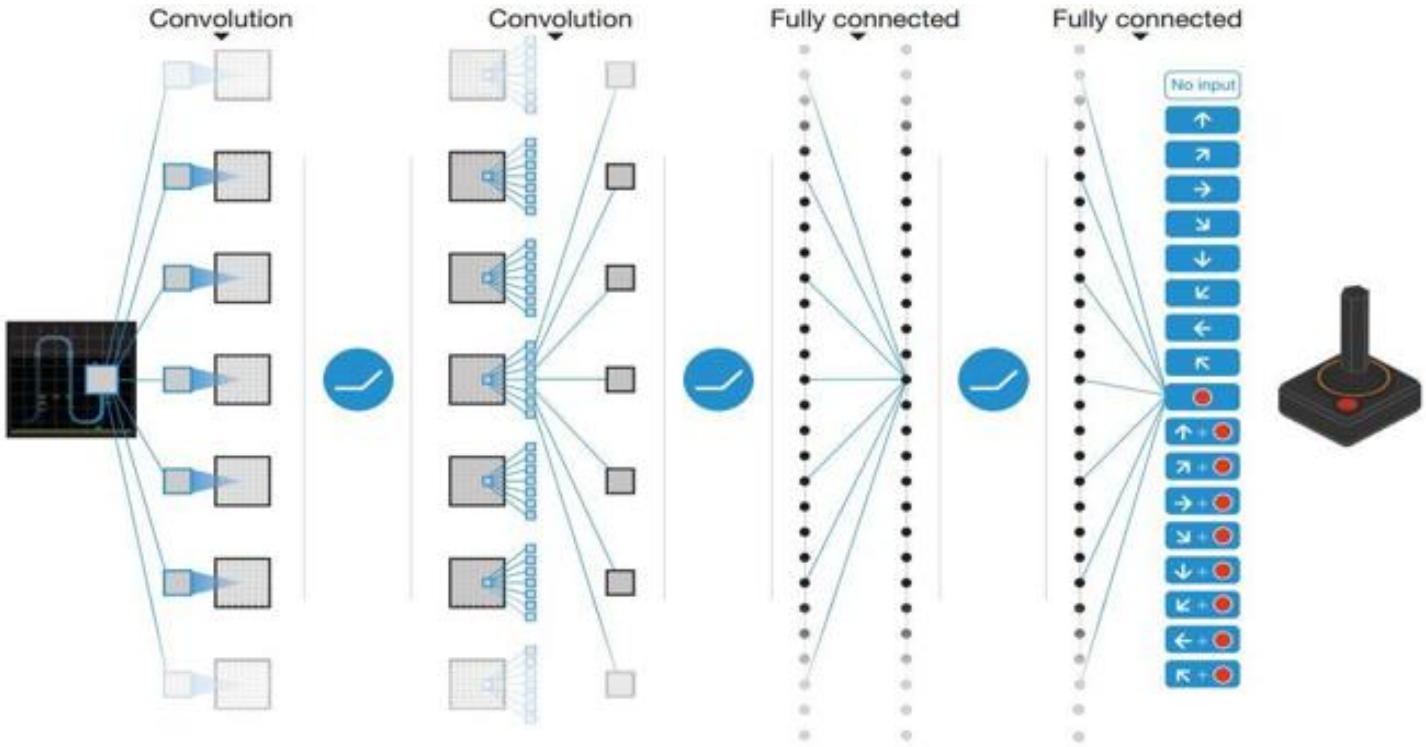
DQN的思路-经验池

- 经验池的功能主要是解决相关性及非静态分布问题。
- 具体做法是把每个时间步agent与环境交互得到的转移样本 (s_t, a_t, r_t, s_{t+1}) 储存到回放记忆单元，要训练时就随机拿出一些（minibatch）来训练。
- 其实就是将游戏的过程打成碎片存储，训练时随机抽取就避免了相关性问题。这样至少有两个好处：
  - 数据利用率高，因为一个样本被多次使用。
  - 连续样本的相关性会使参数更新的方差（variance）比较大，该机制可减少这种相关性。

DQN的思路-目标网络

- 在Nature 2015版本的DQN中提出了这个改进，使用另一个网络（称为TargetNet）产生Target Q值。
- $Q(s,a;\theta_i)$ 表示当前网络MainNet的输出，用来评估当前状态动作对的值函数；
- $Q(s,a;\theta_i^-)$ 表示TargetNet的输出，代入这里求TargetQ值的公式中得到目标Q值。 $TargetQ = r + \gamma \max_{a'} Q(s', a'; \theta)$
- 根据上面的Loss Function更新MainNet的参数，每经过N轮迭代，将MainNet的参数复制给TargetNet。
- 引入TargetNet后，再一段时间里目标Q值使保持不变的，一定程度降低了当前Q值和目标Q值的相关性，提高了算法稳定性。

DQN网络模型



- 输入的是被处理成灰度图的最近4帧 84×84 图像，经过几个卷积层（没有池化层）后接两个全连接层，输出是所有动作的Q值。

DQN算法通述

Step 1： 用一个深度神经网络来作为Q值的网络，参数为ω

$$Q(s,a,\omega) \approx Q^{\pi}(s,a)$$

Step 2： 在Q值中使用均方差来定义目标函数objective function

也就是loss function

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

Step 3： 计算参数ω关于loss function的梯度

$$\frac{\partial L(\omega)}{\partial \omega} = E[(\underbrace{r + \gamma \cdot max_{a'} Q(s',a',\omega)}_{Target} - Q(s,a,\omega))\frac{\partial Q(s,a,\omega)}{\partial \omega}]$$

Step 4： 使用SGD实现End-to-end的优化目标

DQN算法分析

- 网络的输入是4个84×84的灰度游戏屏幕。
- 网络的输出是是每一个可能动作的Q值（Atari中有18个动作），
- DQN网络结构图如下。

Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

DQN_V1

NIPS 2013提出的DQN

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N

Initialize action-value function Q with random weights

for episode $= 1, M$ **do**

特征是状态的函数

Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

With probability ϵ select a random action a_t

otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

通过ϵ-greedy进行探索

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

存储

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 $\mathcal{D}$

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$

Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation

end for

end for

DQN_V1

- 1、初始化replay memory D 容量为N
- 2、用一个深度神经网络作为Q值网络，初始化权重参数
- 3、设定游戏片段总数M
- 4、初始化网络输入，大小为84*84*4，并且计算网络输出
- 5、以概率ϵ随机选择动作a_t，以概率1-ϵ通过网络输出的Q(max)值选择动作a_t
- 6、得到执行a_t后的奖励r_t和下一个网络的输入

DQN_V1

- 7、根据当前的值计算下一时刻网络的输出
- 8、将四个参数作为此刻的状态一起存入到D中（D中存放着N个时刻的状态）
- 9、随机从D中取出minibatch个状态
- 10、计算每一个状态的目标值（通过执行a_t后的reward来更新Q值作为目标值）
- 11、通过SGD更新weight

DQN_V2

- Nature 2015文章，对DQN作了改进和完善，其中对于算法上的变化最主要是引入了单独的Q函数网络。
- 研究者在实践中发现当使用如NN这样的非线性函数逼近器逼近Q函数时RL学习过程并不稳定。这种不稳定有2种原因：
  - 经验数据（即观察序列）具有相关性。
  - Q函数的微小改变会引起策略（policy）的巨大改变，进而改变训练数据分布，以及Q函数与Q函数目标之间的差值。
- 解决方法：
  - 前者可以用experience replay解决（参见NIPS 2013论文）。
  - 后者可采用迭代式更新（iterative update）解决（Nature 2015引入）。

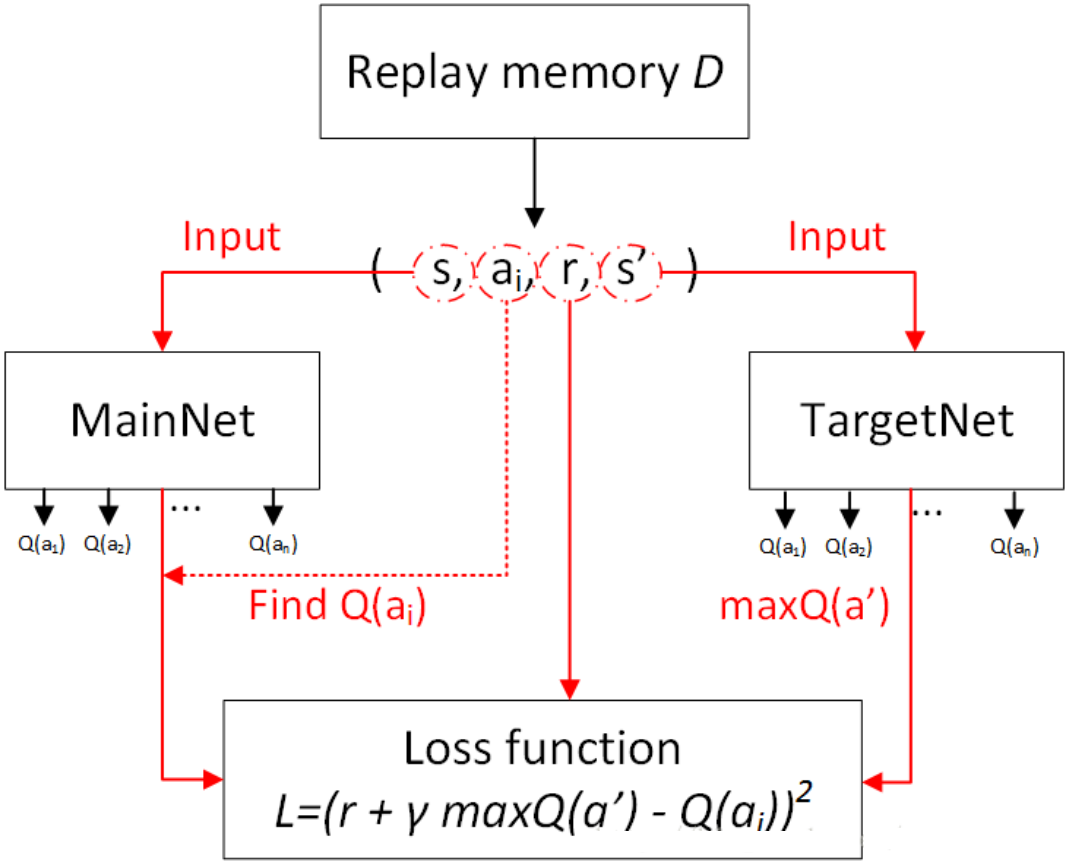
DQN_V2

NIPS 2015提出的DQN

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 $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

DQN_V2

Loss Function 的构造



DQN_V2

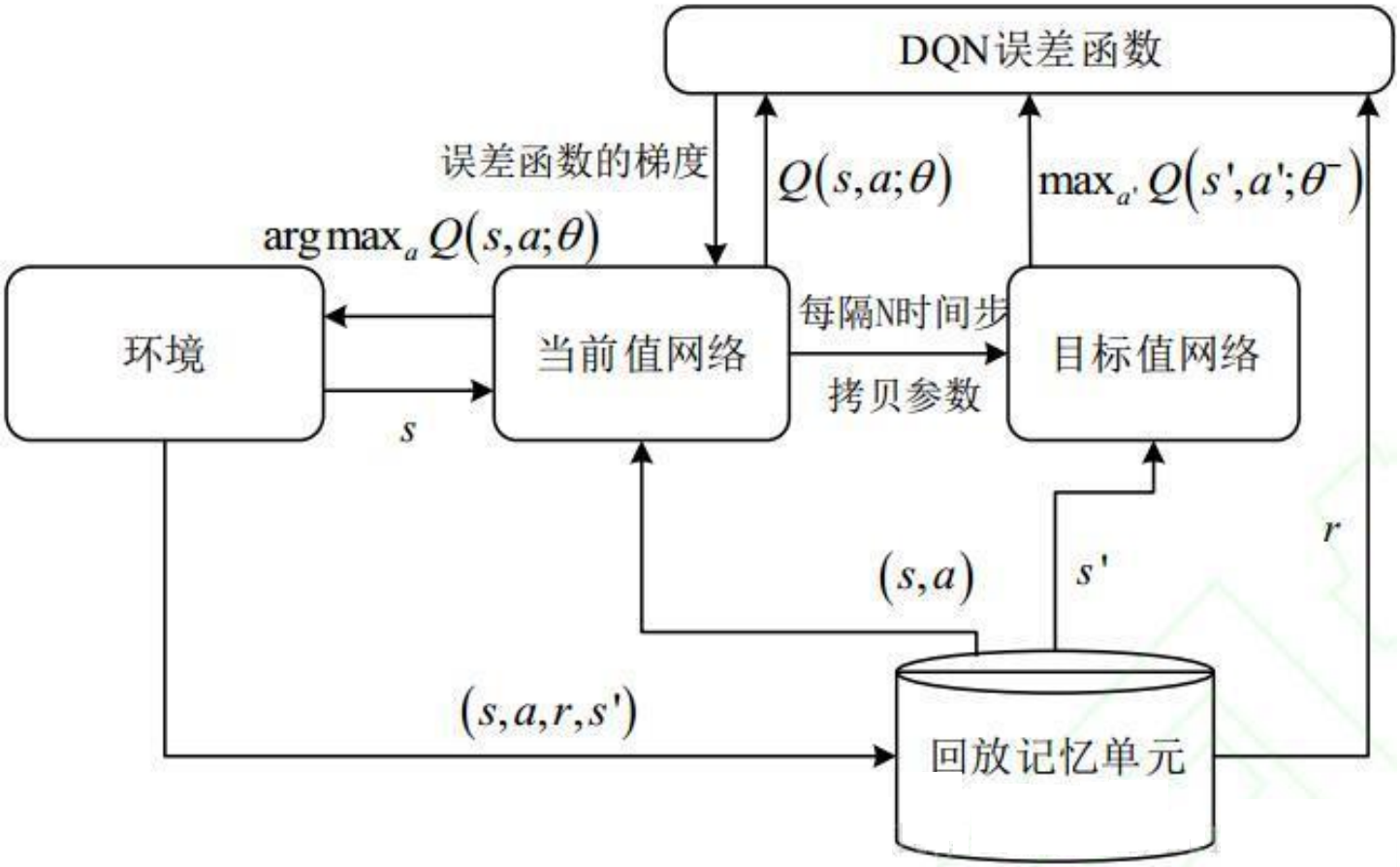
- 1、初始化replay memory D，容量为N，用于存储训练的样本
- 2、初始化action-value function的Q卷积神经网络，随机初始化权重参数θ
- 3、初始化 target action-value function的$\hat{Q}$卷积神经网络，结构以及初始化权重θ^-和θ相同
- 4、设定游戏片段总数M
- 5、初始化网络输入，大小为84*84*4，并且计算网络输出
- 6、根据概率ϵ（很小）选择一个随机的动作或者根据当前的状态输入到当前的网络中（用了一次CNN）计算出每个动作的Q值，选择Q值最大的一个动作（最优动作）

DQN_V2

- 7、得到执行a_t后的奖励rt和下一个网络的输入
- 8、将四个参数作为此刻的状态一起存入到D中（D中存放着N个时刻的状态）
- 9、随机从D中取出minibatch个状态
- 10、计算每一个状态的目标值（通过执行at后的reward来更新Q值作为目标值）
- 11、通过SGD更新weight
- 12、每C次迭代后更新target action-value function网络的参数为当前action-value function的参数

DQN_V2

主要流程图


DQN误差函数
误差函数的梯度
$Q(s,a;\theta)$
$\max_{a'} Q(s',a';\theta^-)$
$\arg\max_a Q(s,a;\theta)$
环境
当前值网络
每隔N时间步
拷贝参数
目标值网络
s
r
(s,a)
s'
(s,a,r,s')
回放记忆单元

DQN算法总结

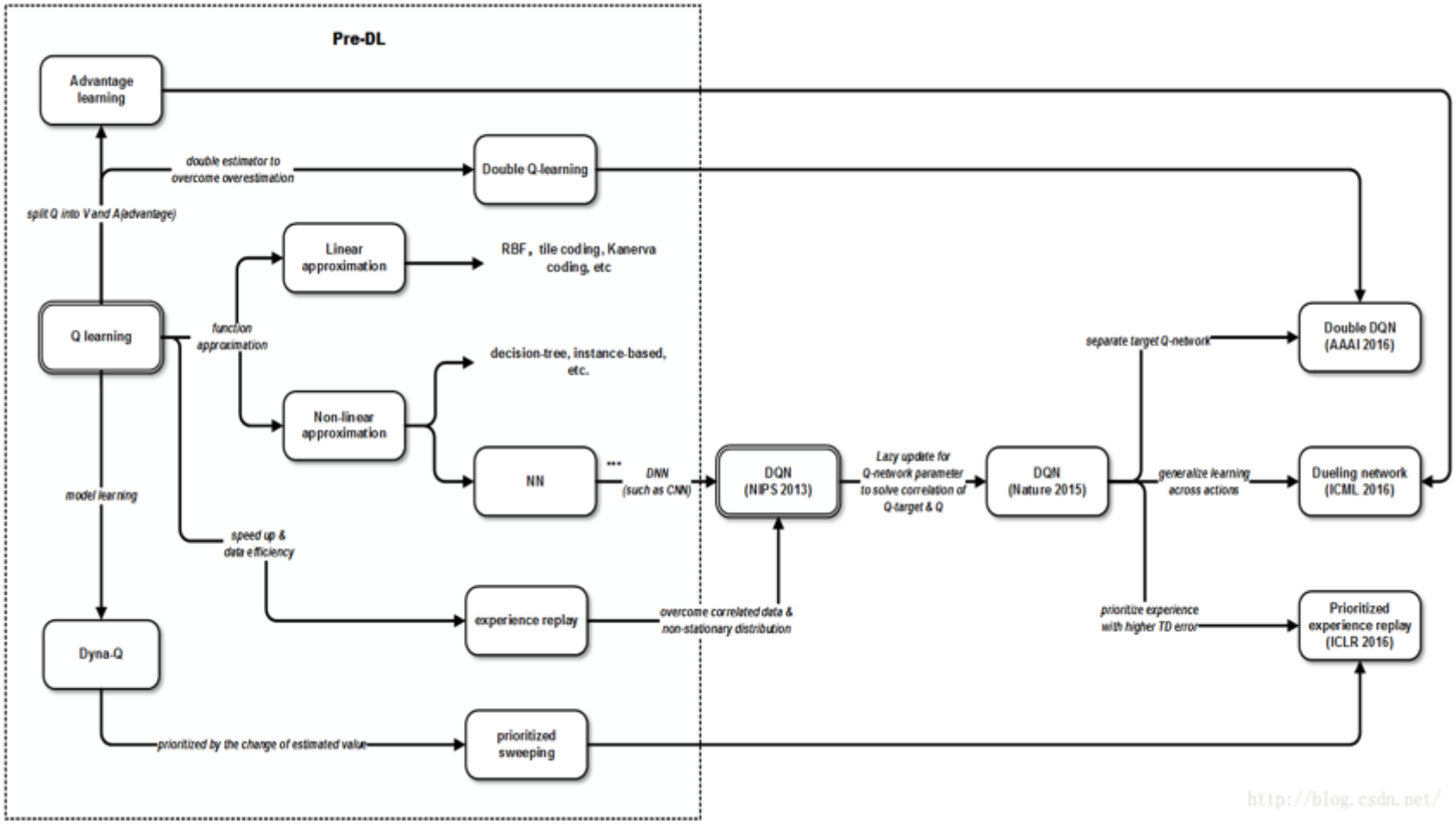
- 创新点：
  - 基于Q-Learning构造Loss Function（也不算很新）。
  - 通过experience replay（经验池）解决相关性及非静态分布问题；
  - 使用TargetNet解决稳定性问题。
- 优点：
  - 算法通用性，可玩不同游戏；
  - End-to-End 训练方式；
  - 可生产大量样本供监督学习。
- 缺点：
  - 无法应用于连续动作控制；
  - 只能处理只需短时记忆问题，无法处理需长时记忆问题；
  - CNN不一定收敛，需要仔细地调参。

2016年，DQN大发展

- 2015年之后，DeepMind的大牛们一刻不停地在AAAI，ICML等顶级会议中相继对DQN作了改进，使其成绩与性能有了质的飞跃。按时间顺序主要有下面论文：

- AAAI 2016 《Deep Reinforcement Learning with Double Q-learning》
- ICLR 2016《Prioritized Experience Replay》
- ICML 2016《Dueling Network Architectures for Deep Reinforcement》
- Nature 2016 《Mastering the game of Go with deep neural networks and tree search》

各种改进的DQN


Pre-DL
Advantage learning
double estimator to overcome overestimation
Double Q-learning
split Q into V and A(advantage)
Linear approximation
RBF，tile coding, Kanerva coding, etc
Q learning
function approximation
decision-tree, instance-based, etc.
Non-linear approximation
NN
DNN (such as CNN)
DQN (NIPS 2013)
Lazy update for Q-network parameter to solve correlation of Q-target & Q
DQN (Nature 2015)
separate target Q-network
Double DQN (AAAI 2016)
generalize learning across actions
Dueling network (ICML 2016)
model learning
speed up & data efficiency
experience replay
overcome correlated data & non-stationary distribution
prioritize experience with higher TD error
Prioritized experience replay (ICLR 2016)
Dyna-Q
prioritized by the change of estimated value
prioritized sweeping
http://blog.csdn.net/

Double Q-learning

- 研究者发现，Q学习中的overestimation问题（在确定状态下Q值估计偏高）可能导致非最优解和学习过程稳定性下降。
- 最初Thrun & Schwartz开始探讨该问题，证明了在使用函数逼近器时overestimation可能导致非最优解。
- 之后van Hasselt发现即使用表格表示法的情况下，环境中的噪声也能导致overestimation，并且提出了解决方案Double Q-learning。
- 而DQN是基于Q-learning，所以本质上也有这个问题。因此将Double Q-learning结合进DQN可以改善。其基本思想是将选择和评估动作分离，让它们使用不同的Q函数（网络）。

Double Q-learning

- 其中一个用于产生贪婪策略（greedy policy），另一个用于产生Q函数估计值。实现时会有两个Q函数网络：原DQN中的Q函数网络称为在线网络（online network），后者称为目标网络（target network）。
- 由于Nature版DQN已经引入了单独的Q目标网络，所以Double DQN对DQN架构基本不需什么改动，只需把目标网络的参数和在线网络的参数独立训练即可。
- 注意和本文方法相比，Nature 2015上的方法相当于是参数的延迟更新，在同一步更新的动作选取和函数估计中还是用的同一个参数。

Prioritized Experience Replay

- 在RL与DL结合的实践中起到比较关键作用的experience replay算法灵感可以说部分来自生物学。最原始的RL是每次观察到一次状态转移（表示为s, a, R, γ, S'）只更新一次参数。
- 这样一来有几个问题：
  - 参数更新具有时间上的相关性，这与随机梯度下降算法的假设不符。
  - 那些出现次数少的状态转移（经验）很快就会被“遗忘”掉。
- DQN中使用了experience replay来缓解这两个问题。该方法不仅稳定了函数训练过程，也提高了经验数据的利用率。缺点是需要更多内存来存储经验池（experience pool）。
- 这种方法的效果很好，但目前的做法还是对以往经验均匀采样的。下一步自然是根据经验的重要程度进行有侧重的replay。

Prioritized Experience Replay

- 在RL领域，上个世纪90年代就有类似的想法，即prioritized sweeping，应用在model-based的规划问题中。
- 直觉上，我们知道一部分经验比其它经验要对参数的训练产生更大的作用。基于此，该方法的基本思想是使参数更新倾向于使值估计变化更大的经验。
- 而怎么衡量哪些经验对值估计的贡献呢，这就需要有定量的测度。在model-free的场景中，这个测度一般选用TD error。

Prioritized Experience Replay

- 具体地，TD error越大，也就是expected learning progress越高的经验数据，可以让它们replay的次数越频繁。基于这种思想实现的greedy TD-error prioritization算法将经验数据和其TD error按序存在replay memory中，每次取最大TD error的经验进行replay，同时参数更新的量也与之同比。另外新的经验会设成最大优先级以保证它至少被训练一次。
- 论文还提到这种做法会导致loss of diversity问题和引入bias，但文中分别用stochastic prioritization和importance sampling方法来减轻和纠正。总得来说，按文的原话说，experience replay使得参数的更新不再受限于实际经验的顺序，prioritized experience replay继而使之不再受限于实际经验的出现频率。

Dueling Network

- 这篇论文提出了针对model-free RL的dueling network框架。它是对传统DQN架构层面上的改动，将基于状态的V函数（value function）和状态相关的advantage函数（advantage function）分离。
- Advantage函数的思想基于1993年Baird提出的advantage updating。除了传统的V函数外，引入的advantage函数A(x, u)的定义是当采取动作u相比于采取当前最优动作能多带来多少累积折扣回报。
- 简单粗暴得说，就是选这个动作比当前最优动作（或其它动作）好多少。

Dueling Network

- 基于这个思想，同一个网络会同时估计V函数和advantage函数，它们结合起来可以得到Q函数。
- 从架构上来说，这是个一分为二，再合二为一的过程。它的出发点是因为对于很多状态，其实并不需要估计每个动作的值。
- 可以预见到，引入advantage函数后，对于新加入的动作可以很快学习，因为它们可以基于现有的V函数来学习。它直觉上的意义在于将Q函数的估计分为两步。
- 这样，可以先估计哪些状态更能获得更多回报，而不受该状态下不同动作的干扰。文中举了典型的赛车游戏的例子。可以看到V函数专注于远处（地平线）和分数，也就是长期目标，advantage函数专注于附近障碍，也就是短期目标。这说明V函数和advantage函数分别学习到了两个层次的策略。

Dueling Network

- 这种分层学习的做法有几个好处：
  - 一是V函数可以得到更多的学习机会，因为以往一次只更新一个动作对应的Q函数。
  - 二是V函数的泛化性更好，当动作越多时优势越明显。直观上看，当有新动作加入时，它并不需要从零开始学习。
  - 三是因为Q函数在动作和状态维度上的绝对数值往往差很多，这会引起噪声和贪婪策略的突变，而用该方法可以改善这个问题。

参考资料

- https://zhuanlan.zhihu.com/p/26007538
- https://blog.csdn.net/u013236946/article/details/72871858
- https://blog.csdn.net/jinzhuojun/article/details/52752561
- https://blog.csdn.net/m0_37600149/article/details/78570410

问答互动

在所报课的课程页面，
1、点击“全部问题”显示本课程所有学员提问的问题。
2、点击“提问”即可向该课程的老师和助教提问问题。

联系我们

小象学院：互联网新技术在线教育领航者

– 微信公众号：**小象学院**