Rainbow

Abstract:
目前已经有几项针对 DQN 的改进措施，但不知道这些改进能不能有效地结合起来。我们尝试了其中的六种，并用实验去研究它们结合使用时的效果。实验显示结合使用在 Atari2600 上效果很好。我们也提供消融实验的结果，这展示了各部分对于总成绩的贡献。

Introduce:
1 首先介绍了 DQN，然后介绍几种改进。
2 几种改进
DoubleDQN：通过分离对 bootstrap action 选择和评估来解决 Q-learning 的过度估计偏差（在统计学中，Bootstrap 法具体是指用原样本自身的数据抽样得出新的样本及统计量。它是一类非参数 Monte Carlo 方法,其实质是对观测信息进行再抽样，进而对总体的分布特性进行统计推断。-机器学习中的 bootstrap 到底是什么？ - 知乎 (zhihu.com)）

Prioritized experience replay：通过 replay 更常见的 transitions 来提升数据效率

*Dueling network architecture：通过分别 represent 状态值 (state value) 和 action advantage 来概况 across action

Learning from multi-step bootstrap targets:转变 bias-variance 平衡并使新观察到的 reward 传播得比之前观察到的 state 要快

Distributional Q-learning：学习 return（总 reward？）的真实分布而不是去估计

Noisy DQN：用一层随机网络去观察，以增加噪声提高模型的稳固性（robust）
下边将会把这些方法一起用

Background:
RL：用 agent 在环境中进行一系列动作，并使 reward 最大化。

Agents and environments：每个离散时间，environment 给一个状态 St，agent 给一个动作 At，然后 env 给下一个状态 St+1 和 reward Rt+1 和衰减因子 y t+1。最终形成 MDP $\langle \mathcal{S}, \mathcal{A}, T, r, \gamma \rangle$，S、A 是 St，At 的集合，

$$T(s, a, s') = P[S_{t+1} = s' \mid S_t = s, A_t = a]$$

T 是（随机）转移函数（从 s 转移至下一个状态 s'）
r(s,a)是 reward 函数，最后那个是衰减因子，实验中衰减因子为一常量。

Agent 用策略 pi 来选择动作。

discounted return $G_t = \sum_{k=0}^{\infty} \gamma_t^{(k)} R_{t+k+1} \gamma_t^{(k)} = \prod_{i=1}^{k} \gamma_{t+i}$.

Agent 的目标是采用一个策略 pi 去最大化 Gt。

策略 pi 可以被直接训练，可以应用ε-greedy 进行更新。

DRL 和 DQN：在 DRL 中，我们用 Deep nn 训练例如 pi（s,a）q(s,a)等部分.
在 DQN 中，使用 CNN 去评估动作的值，用 SGD（RMSprop，SGD 的变种）作为优化器。

以 $(R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t))^2$ 为 Loss 函数，其中 t 为 replay memory 中任取的值。

DQN 的拓展方法：

Double Q-learning：改变 Loss 函数，防止过高的估计

$(R_{t+1} + \gamma_{t+1} q_{\bar{\theta}}(S_{t+1}, \operatorname{argmax}_{a'} q_\theta(S_{t+1}, a')) - q_\theta(S_t, A_t))^2$.

Prioritized replay：使用一个概率 pt 来选择需要 replay 的 transition，其中 pt 定义为：

$$p_t \propto \left| R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t) \right|^\omega$$

其中 w 是个超参数。

Dueling networks：value 和 advantage 分别计算，它们共用一个 convolution encoder，并用一个特殊的汇集器融和。 $q_\theta(s,a) = v_\eta(f_\xi(s)) + a_\psi(f_\xi(s), a) - \frac{\sum_{a'} a_\psi(f_\xi(s), a')}{N_{\text{actions}}}$

其中 $\theta = \{\xi, \eta, \psi\}$ 三个参数分别代表共用部分的参数、value 的参数和 advantage 的参数。

Multi-step learning：采用一个截短的 n 步 reward $R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}$. 而不是原先的∞步。

相应的，loss 函数也做了改动 $(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} q_{\bar{\theta}}(S_{t+n}, a') - q_\theta(S_t, A_t))^2$
通过调节 n 的值，可以实现快速训练。

*Distributional RL：没看懂

Noisy Nets：引进有噪音的线性层 $\boldsymbol{y} = (\boldsymbol{b} + \mathbf{W}\boldsymbol{x}) + (\boldsymbol{b}_{noisy} \odot \epsilon^b + (\mathbf{W}_{noisy} \odot \epsilon^w)\boldsymbol{x})$来替代普通的线性层。其中两个 ε 是随机变量，$\odot$ 代表 element-wise product（类似于点乘）。

The Integrated Agent：
融合以上方法的 agent，被命名为 rainbow。

Experimental Methods:
使用以前文章中的环境和评估方法，将 agent 训练 1M 步然后暂停学习，评估 500k frames。每个 episode 被截断为 108k frames（或者 30 分钟的仿真游戏）。Agent 的得分被归一化。作者还做了消融实验，结果见下

Agent	no-ops	human starts
DQN	79%	68%
DDQN (*)	117%	110%
Prioritized DDQN (*)	140%	128%
Dueling DDQN (*)	151%	117%
A3C (*)	-	116%
Noisy DQN	118%	102%
Distributional DQN	164%	125%
Rainbow	223%	153%

Discussion:
结果证明，融合亿堆方法的 rainbow 很成功，并且也得到了每项改进的效果，但还有很多改进 DQN 的措施没有被融进 rainbow，然后作者介绍了其他措施。

DQN2013

Abstract：应用 Q-learning 变种训练的 CNN 模型（DQN），能够玩游戏（输入游戏画面/原始像素，输出动作），效果很好。

Introduction：

1 原先的 RL 很难用高维输入（比如图像）控制 agent

2 DL 可以从高维输入中提取信息，所以我们打算用 DL 优化 RL

3 但是有一些问题

4 这篇 paper 将展示 CNN 可以解决这些问题

5 我们用 Atari 2600 去测试我们的方法。我们的目标是只要训练一个 agent 就能学会玩尽量多的游戏。Agent 知道的只有游戏画面、reward、终端信号、动作空间。结果很好

Backward：

1 流程：agent 输入代表游戏画面的像素向量和 reward，输出动作种类

需要注意的是总游戏得分可能会依赖于整个动作和画面（observation）的顺序；一个动作的 feedback 可能很久以后才能收到

2 Agent 很难只通过当前的画面就了解情况，我们考虑使用一系列动作和 observation 来训练 “st = x1, a1, x2, ..., at−1, xt”,序列在有限的时间后结束。这个序列可以看作是一个 MDP，所以我们可以用 MDP 的方法去做这玩意。

3 $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]$$

Rt 是衰减的、未来奖励的和。 最佳 S-A 函数

4 Q 函数遵守贝尔曼方程

5 利用贝尔曼方程更新 Q 函数在实践中是不行的，因为 Q 函数只估计一个 s 的，没有归纳一堆 s 的。常见的作法是用一个 function approximator（函数近似器）来估计 Q，可以用线性也可以用非线性的（比如神经网络）。我们用神经网络作为 Q-network。

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right],$$

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a\right]$$

Loss 函数定义，i 代表每次迭代，yi 是 i 的 target，p（s，a）是 s，a 的概率分布，作为动作（行为）分布。注意 target 依赖于 nn 的参数θ，这和监督学习不一样。

6 用 SGD 优化。如果 weights 每个时间戳都更新，并且 expectation 被动作分布和 emulator 各自替代，这就很像是 Q-learning

7 注意算法不依赖于模型，同时也是 off-policy 的。（∈-greedy strategy 没看懂）

Related work:

1 介绍 TD-Gammon 算法，这是一项应用在西洋双陆棋上的方法，使用时间差分学习(Temporal-Difference Learning)

2 TD 算法在国际象棋，围棋和跳棋上的应用不成功

3 TD 结合无模型算法、非线性函数近似器，off-policy 可能会使 Q 函数发散。后来更多的用线性近似器

4 最近，结合 DL 和 RL 的工作又火了。Deep nn 被用于评估环境，restricted Boltzmann machines 被用于评估 Value 函数或 policy。另外，Q 函数发散问题一定程度上被 gradient temporal-difference 解决

*5 和我们工作最相似的是 NFQ，这是一种用于解决 MDP 过程的算法。

6 使用 Atari 2600 emulator 作为强化学习的测试平台这一想法被 xx 文章最初提到。

DRL:

1 和上面的内容基本一致

2 TD-Gammon 是本方法的出发点

3 和 TD 不同，我们使用经验回放技术，即存储过去的经历

$$e_t = (s_t, a_t, r_t, s_{t+1}) \quad \mathcal{D} = e_1, ..., e_N,$$

我们用 Q-learning 进行更新。Agent 通过 ϵ-greedy policy 挑选动作

4 这个方法比 standard online Q-learning 要好：

首先，一次经验可应用在多次权重更新上，这可以提高数据利用效率。

其次，直接用连续的样本学习效率不高，因为样本之间的关联性比较高（样本很多信息之间是重复或者相似的，所以说学习相似的样本就相当于重复学习，不如相似度较低的样本那样有效）

第三，经验回放是 off-policy 的，不会像 on-policy 那样训练完样本就不能用了，效率会高很多。

实践中，DQN 算法只在缓冲区中存储了最后一个 N 经验，并且当更新时只从 D 中随机的抽取一项经验，这是因为缓冲区大小有限，并且不能分辨出更为重要的 translation（给予相同的重要性）。如果采用一种更加复制的采样（sampling）策略也许就能辨别出更重要的 translation。

DQN 算法的伪代码见下：

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**
  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 $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    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 3
  end for
end for

初始化 经验回放的缓冲区 D　#以存储 N
初始化 Q 函数 ； Q 函数中的权重随机化
For episode=1，M do

初始化列表 s={x}和预处理后的列表 $\phi_1 = \phi(s_1)$

For t=1,T do

根据概率ε随机选择一个动作 a

否则选择

在模拟器（环境）中执行 a 并返回 reward r 和 image x

使 $s_{t+1} = s_t, a_t, x_{t+1}$ 并作预处理

把转换四元组 $(\phi_t, a_t, r_t, \phi_{t+1})$ 储存到 D 中

在 D 中随机抽一小批转换组

使

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

#terminal，最后一个预处理后的状态

用前面提到的公式做梯度下降

结束循环
结束循环

预处理部分:
210x160 的图片 with128 种颜色=》110x84 的图片 with 灰度图=》84x84 的灰度图（二维卷积层需要方阵作为输入）。预处理函数Φ会把最近的四帧图像连起来作为 Q 函数的输入。

架构：
输入层 84x84x4（预处理后）
Hid1　16 个 8x8 的卷积核，stride=4
ReLU
Hid2　8 个 4x4 的卷积核，stride=2
ReLU
Hid3　全连接，有 256 个 ReLU 单元（到 256 维的线性层+ReLU ？）
输出层 全连接，输出维数为动作的种类数目。

这就是 DQN

Experiment：
采用 7 种游戏，奖励统一为 1 0 -1（正奖励 0 负奖励），采用 RMSprop 和ε-greedy 算法，训练了 10M 帧，经验回放缓冲区为 1M
和其他算法及人类的结果比较。

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Mnih2015

Abstract：
RL 提供了一种规范性说明，但是 agent 在处理高维信息时遇到了问题。人和动物使用 RL 和多级感官系统来学习。RL 只在低维状态空间中取得好效果。我们用 Deep nn 来创建一个新的模型，Deep Q-learning，它可以处理高维信息，并在多个游戏中取得很好的效果。

DQN 部分和 DQN2013 内容基本一样，只做了一些优化，例如微调了模型架构，误差裁剪，预处理部分做了微调等。