Deep Reinforcement Learning 深度增强学习资源(持续更新)

Deep Reinforcement Learning 深度增强学习资源(持续更新)

DQN从入门到放弃1 DQN与增强学习

DQN 从入门到放弃1 DQN与增强学习

★ 前言

深度增强学习Deep Reinforcement Learning是将深度学习与增强学习结合起来从而实现从Perception感知到Action动作的端对端学习End-to-End Learning的一种全新的算法。

★ 预备条件

概率论、线性代数、Python编程基础

- 深度学习(Deep Learning)
- 增强学习(Reinforcement Learning)

★ 增强学习是什么

DEF: 智能体Agent

表示一个具备行为能力的物体，比如机器人，无人车，人等等。
增强学习考虑的问题就是**智能体Agent**和**环境Environment**之间交互的任务。

DEF: 反馈值Reward

所谓的Reward就是Agent执行了动作与环境进行交互后，环境会发生变化，变化的好与坏就用Reward来表示。

DEF: 观察Observation

Observation表示Agent获取的感知信息。
增强学习的任务就是找到一个最优的策略Policy从而使Reward最多。

DQN 从入门到放弃2 增强学习与MDP

DQN 从入门到放弃2 增强学习与MDP

⋆ 增强学习的世界观

- 世界的时间是可以分割成一个一个时间片的，并且有完全的先后顺序，因此可以形成

$$s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_t, a_t, r_t$$

这样的状态，动作和反馈系列。

- 增强学习中每一次参数的调整都会对世界造成确定性的影响。

⋆ MDP(Markov Decision Process)马尔科夫决策过程

DEF: Markov

一个状态S_t是Markov当且仅当

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_t, s_{t-1}, \ldots, s_1, s_0)$$

P为概率。

一个基本的MDP可以用(S, A, P)来表示，S表示状态，A 表示动作，P表示状态转移概率，也就是根据当前的状态s_t 和a_t转移到s_{t+1}的概率。如果我们知道了转移概率P，也就是称为我们获得了**模型Model**，有了模型，未来就可以求解，那么获取最优的动作也就有可能，这种通过模型来获取最优动作的方法也就称为Model-based 的方法。但是现实情况下，很多问题是很难得到准确的模型的，因此就有Model-free的方法来寻找最优的动作。

⋆ 回报Result

DEF: 回报Result

回报Return来表示某个时刻t的状态将具备的回报：

$$G_t = R_{t+1} + \lambda R_{t+2} + \cdots = \sum_{k=0}^{\infty} \lambda^k R_{t+k+1}$$

上面R是Reward反馈，λ是discount factor折扣因子，一般小于1，就是说一般当下的反馈是比较重要的，时间越久，影响越小。

用价值函数value function $v(s)$来表示一个状态未来的潜在价值。

DEF: value function

value function就是回报的期望：

$$v(s) = \mathbb{E}[G_t|S_t = s]$$

DQN 从入门到放弃3 价值函数与Bellman方程

DQN 从入门到放弃3 价值函数与Bellman方程

⋆ Value Function 价值函数

⋆ 再谈增强学习的意义

L深度学习给了计算机“神经网络大脑”，RL给了计算机学习机制。

⋆ Bellman方程

$$\begin{aligned} v(x) &= \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \lambda R_{t+2} + \lambda^2 R_{t+3} + \dots |S_t = s] \\ &= \mathbb{E}[R_{t+1} + \lambda(R_{t+2} + \lambda R_{t+3} + \dots)|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \lambda G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \lambda v(S_{t+1})|S_t = s] \end{aligned}$$

DEF: Bellman方程

$$v(s) = \mathbb{E}[R_{t+1} + \lambda v(S_{t+1})|S_t = s]$$

DQN 从入门到放弃4 动态规划与Q-Learning

DQN 从入门到放弃4 动态规划与Q-Learning

⋆ 上文回顾

Bellman方程透出的含义就是价值函数的计算可以通过迭代的方式来实现。

⋆ Action-Value function 动作价值函数

DEF: 动作价值函数Action-Value function

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}[r_{t+1} + \lambda r_{t+2} + \lambda^2 r_{t+3} + \dots |s,a] \\ &= \mathbb{E}_{s'}[r + \lambda Q^\pi(s',a')|s,a] \end{aligned}$$

⋆ Optimal value function 最优价值函数

最优动作价值函数和一般的动作价值函数的关系：

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \lambda \max_{a'} Q^*(s',a')|s,a]$$

⋆ 策略迭代Policy Iteration

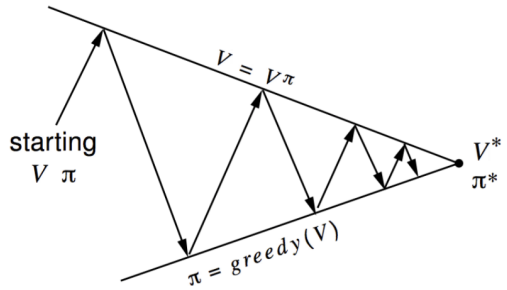
Policy Iteration的目的是通过迭代计算value function 价值函数的方式来使policy收敛到最优。

Policy Iteration本质上就是直接使用Bellman方程而得到的：

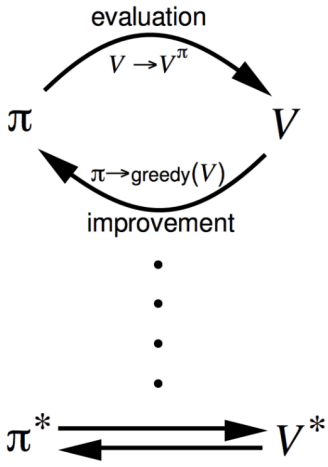
$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \end{aligned}$$

Policy Iteration一般分成两步：

1. Policy Evaluation 策略评估。目的是更新Value Function，或者说更好的估计基于当前策略的价值。
2. Policy Improvement 策略改进。使用greedy policy 产生新的样本用于第一步的策略评估。



Policy evaluation Estimate v_π
Any policy evaluation algorithm
Policy improvement Generate $\pi' \geq \pi$
Any policy improvement algorithm



1. Initialization

   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

   Repeat

   $\Delta \leftarrow 0$

   For each $s \in \mathcal{S}$

   $v \leftarrow V(s)$

   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   Until $\Delta < \theta$ (a small positive number)

3. Policy Improvement

   $policy - stable \leftarrow true$

   For each $s \in \mathcal{S}$

   $a \leftarrow \pi(s)$

   $\pi(s) \leftarrow argmax_a \sum_{s',r} p(s', r|s, a)[r + \lambda V(s')]$

   if $a \neq \pi(s)$, then $policy - stable \leftarrow false$

   if $policy - stable$, then stop and return V and π; else go to 2

Figure 1: Policy iteration算法

这里policy evaluation部分的迭代需要知道state状态转移概率p，也就是说依赖于model模型。

⋆ Value Iteration 价值迭代

Value Iteration则是使用Bellman 最优方程得到

$$\begin{aligned} v_* &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \end{aligned}$$

然后改变成迭代形式

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \end{aligned}$$

value iteration的算法如下：

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that

$\pi(s) = argmax_a \sum_{s',r} p(s', r|s, a)[r_\gamma V(s')]$

Figure 2: Value iteration算法

value iteration较之policy iteration更直接，不过问题也都是一样，需要知道状态转移函数p才能计算。

⋆ Q-Learning

Q Learning提出了一种更新Q值的办法：

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

具体的算法如下：

初始化$Q(s,a)$, $\forall s \in S$, $a \in A(s)$, 任意的数值，并且$Q(terminal-state,\cdot)=0$

重复(对每一节episode)

初始化状态S

重复(对episode中的每一步)

使用某一个policy比如$(\epsilon-greedy)$根据状态S选取一个动作执行

执行完动作后，观察reward和新的状态S'

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

$S \leftarrow S'$

循环直到S终止

Figure 3: Q-Learning算法

★ Exploration and Exploitation 探索与利用

- 以Q-Learning算法叫做Off-policy的算法。
- Q-Learning完全不考虑model模型也就是环境的具体情况，只考虑看到的环境及reward，因此是model-free的方法。

选择怎样的policy来生成action呢？有两种做法：

- 随机的生成一个动作。
- greedy policy贪婪策略，如ϵ-greedy 策略。

DQN从入门到放弃5 深度解读DQN算法

DQN从入门到放弃5 深度解读DQN算法

★ 详解Q-Learning

★ 维度灾难

★ 价值函数近似Value Function Approximation

★ 高维状态输入，低维动作输出的表示问题

★ Q值神经网络化！

★ DQN算法

Q网络训练的损失函数就是

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

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 $(\phi_j, a_j, r_j, \phi_{j+1})$ form $\lceil$

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

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

Figure 4: Deep Q-learning with Experience Replay

⋆ DQN训练

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！

DQN实战篇1 从零开始安装Ubuntu, Cuda, Cudnn, Tensorflow, Op

DQN实战篇1 从零开始安装Ubuntu, Cuda, Cudnn, Tensorflow, OpenAI Gym