理论知识复习

强化学习是机器学习中的一个领域

- 从要解决的问题角度来看
  - 分类（classification）
  - 回归（regression）
  - 强化学习（以围棋为例：多次决策达成一个长期目标；每次落子都会改变棋局，产生深远影响；当前这一步走得好不好，还与后面的走法有关）
- 从学习的方式上来看
  - 监督式学习（supervised learning）
  - 强化学习（reinforcement learning OR unsupervised learning）

> 传统的机器学习依附于过往的经验，训练数据往往被打上了标签（label），相当于有一个确定的期望输出。而强化学习应用的场景中没有过往的数据可以借鉴，没法用一个标签去判断这次的输出是好还是坏。
>
> 从某个角度上理解，传统的机器学习任务专注于寻找已存在的数据中的“**特征**”（characteristic）和“**模式**”(pattern)；而强化学习则是倾向于用神经网络去模拟人脑的决策过程，通过`trail-and-error`的方式去试图为一个长期任务找出可行解。

- 监督学习？强化学习？

  标签(`label`)与奖赏(`reward`)

  `RL`:**不断尝试，总结经验，提升能力。**

  监督学习的标签是提前打好的，强化学习中即使是经验回放池里的经验也是通过探索得来的。

马尔可夫决策过程（MDP）

- 四元组：$< S, A, T, R >$刻画了任务环境（定义问题）。其中：
  - S表示状态（state）集合。状态可以理解为环境中所有信息的集合（一个全面的观察）
  - A表示动作（action）集合。
  - T表示状态转移（transition）。状态转移描述了在某个状态s下，执行某个动作a,转移到下一个状态$s_$并获得一定奖赏的可能性:

    $$T = p(s\_, r|s, a)$$

    有无确定的状态转移概率也是有模型学习和无模型学习的区别。
  - R表示在某个状态s下，执行某个动作a，并转移到下一个状态$s_$的得分（reward），记作:

    $$R = p(r|s, a, s\_)$$

- 有模型的学习？无模型的学习？

  区别是有没有给出状态转移概率。

- 策略（policy）是状态到动作的映射。一般性地，如果考虑随机策略，可以记作$\pi(a|s)$。
- **马尔可夫性**：下一状态的概率分布只与当前状态有关，$与时序序列中之前的状态均无关。

策略与值函数

- 强化学习的目标：最大化未来累计奖赏的期望。
- 回合制任务和持续性任务：
  - 任务有明确的终止状态称为回合任务

    可以将终止状态定义为不管执行什么动作，都转移到自身并且奖赏为0的持续性任务
  - 任务可以无穷尽地进行下去，则称为持续性任务

    累积奖赏可能**发散**（diverge）。需要引入折扣系数（$0 \leq \gamma < 1$）的概念：

    $$r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \cdots$$

    可见引入折扣系数的作用主要是为了**保证奖赏收敛。**
- 状态值函数：

  使用某个策略π，在某个状态s下，未来能够获得的累积奖赏的期望是多少，记作$v_\pi(s)$
- 动作值函数：

  使用某个策略π，在某个状态s下，执行了动作a，未来能够获得的累积奖赏的期望是多少，记作$q_\pi(s,a)$

> 二者的关系？

$$v_\pi(s) = \sum_a \pi(a|s) \times q(a|s)$$

- 蒙特卡罗方法（`Monto Carlo`）

  用随机数的方法解决计算问题。

  为了保证每个状态都能碰到，需要保证策略具有一定的随机性($\epsilon - greedy$)或者初始状态可能处于任意状态（exploration start）
- 策略的评估与改进

  假设改进后的策略为π'，我们希望改进之后$v_{\pi'}(s) \geq v_\pi(s), \forall s$。

  改进方法：$\pi'(s) = argmax_a q_\pi(s,a)$
- 时序差分学习（`Temporal difference learning`, aka `TD learning`）

  是一类无模型强化学习方法的统称，这种方法强调通过**从当前价值函数的估值中自举**的方式进行学习。这一方法需要像蒙特卡罗方法那样对环境进行取样，并**根据当前估值对价值函数进行更新**，宛如动态规划算法。（`wiki`:TD-learning)
  - 能否每一步都更新对值函数的估计，并更新策略?
  - 每一步都利用当前对Q的估计

    $$Q(s,a) \leftarrow r + \gamma Q(s',a')$$

  `on-policy` 时序差分 `SARSA` 和 `off-policy` 时序差分 `Q-learning`

深度学习算法介绍（单无人机，解决连续状态空间和连续动作空间）

- 神经网络编程相关概念

  损失函数，梯度下降，神经网络前向传播，反向传播与优化器（计算出损失函数后，计算损失函数关于神经网络参数的梯度，并使用梯度下降法最小化损失函数），均方差`MSE`

- 通用训练循环
  - load batch, compute loss
  - `.zero_grad()`,`.backward()`,`.step()`

从Q-learning到DQN（Deep-Q learning）：解决连续状态空间问题

- 核心思想：使用一个神经网络来拟合Q函数：$Q(s,a;\theta)$其中θ是神经网络参数向量（concatenation）。
- 输入：状态s，可以应用于连续状态空间
- 输出：$|A|$维向量（比如“上下左右”就是4维），仅适用于离散动作空间的任务
- 策略形式保持与`Q-learning`一致：$\pi(s)=argmax_a q_\pi(s,a)$
- 一个具体的四元组(s,a,r,s')成为DQN的一个样本
- 神经网络的更新目标：

$$Q(s,a)\leftarrow r+\gamma\cdot max_{a'}Q(s',a')$$
$$y=r+\gamma\cdot max_{a'}Q(s',a';\theta)$$

-
- 使用均方差函数来计算损失函数：

$$L(\theta)=\frac{1}{2}[Q(s,a;\theta)-y]^2$$

- **技巧一：经验回放**：
  - `problem1`：统计机器学习方法通常要求数据集样本之间满足独立同分布假设，而智能体采样时相邻的两个四元组间有很强的相关性
  - `problem2`：样本利用率低，一个样本只用一次就丢弃，智能体与环境交互往往是较为昂贵的
  - 解决方法：收集到的样本存放到一个`buffer`中，称为经验回放池，每次从池中随机抽取一小批样本用于神经网络更新
    - 打乱了样本间的相关性
    - 一个样本多次利用，节省成本
- **技巧二：使用目标网络**
  - `problem1`：自举导致偏差的传播。`DQN`的更新目标部分基于自己做出的估计（θ），这样的偏差会传播
  - `problem2`：不稳定的更新目标
  - 解决方法：引入一个目标网络，它与主网络结构相同。目标网络的参数**定期从主网络进行复制**，保证目标网络的参数在一段时间内保持**稳定。**更新目标改为：

$$y=r+\gamma\cdot max_{a'}Q(s',a';\theta^-)$$

- `DQN`算法训练流程：

1. 收集训练数据：使用$\epsilon-greedy$策略去控制智能体与环境交互，获得一个四元组样本(s,a,r,s')，然后将样本加入经验回放池。
2. 更新网络参数：

   随机从经验回放池中取出一小批样本，对其中每个样本(s_j,a_j,r_j,s_{j+1})，计算下面两个Q值:

   $$Q(s_j,a_j;\theta)\quad max_aQ(s_{j+1},a;\theta^-)$$

   然后计算TD目标和TD误差：

   $$y_j=r_j+max_aQ(s_{j+1},a;\theta^-)\quad L(\theta)=\frac{1}{2}[Q(s_j,a_j;\theta)-y_j]^2$$

   使用梯度下降更新参数θ
3. 更新目标网络参数：p定期将θ^-赋值为θ

从DQN到Double DQN：解决DQN中Q值高估的问题

- DQN算法存在过度估计问题，TD目标会高估真实价值
- 解决办法：将动作的选择和评估进行**解耦**，**使用主Q网络进行动作选择，使用目标Q网络进行评估**，TD更新目标改为：

  $$y=r+\gamma\cdot Q(s',argmax_{a'}Q(s',a';\theta);\theta^-)$$

  这样有：

  $$Q(s',argmax_{a'}Q(s',a';\theta);\theta^-)\le max_{a'}Q(s',a';\theta^-)$$

从DQN到DDPG（Deep Deterministic Policy Gradient）：解决连续动作空间问题

- 核心思想：使用一个神经网络来充当策略函数（称为`Actor`），输入是状态s，输出是动作a，记作$\mu(s;\theta^\mu)$使其（尽可能）输出$argmax_aQ(s,a;\theta^Q)$
- p并由另一个网络（称为`Critic`）来估计Q值

> actor网络的输入是state，输出是action。之后将state和action拼接之后作为critic网络的输入，输出是估计的Q值。回想DQN的输入是state，输出则是一个与维度大小与离散动作空间维度大小对应的Q值向量。
>
> 事实上，DQN的输入也相当于是state+action，只不过因为动作空间有限我们不需要显式地在输入中给出，直接枚举所有动作编码进运算过程中即可

- `DDPG`算法训练流程：
  1. 收集训练数据：按照$a=\mu(s;\theta^\mu)+N$去控制智能体与环境交互，N为随机噪声，用于鼓励智能体进行探索，将获得的样本加入经验回放池
  2. 更新网络参数：

     随机从经验回放池中取出一小批样本，对其中每个样本(s_j,a_j,r_j,s_{j+1})，计算$Critic$的TD目标:

     $$y_j=r_j+\gamma\cdot Q(s_{j+1},\mu(s_{j+1};\theta^{\mu-});\theta^{Q-})$$

     然后计算损失函数：

     $$L(\theta^Q)=\frac{1}{2}[Q(s_j,a_j;\theta^Q)-y_j]^2\quad L(\theta^\mu)=-Q(s_j,\mu(s_j;\theta^\mu);\theta^Q)$$

使用梯度下降更新参数θ^Q和θ^μ

3. 更新目标网络参数：基于指数移动平均更新目标网络参数θ^{Q-}和$\theta^{\mu-}$

- 注意，`critic`网络先更新，`actor`网络再更新。（另一个参数不动）

从DDPG到TD3（Twin Delayed DDPG）：对DDPG的进一步优化

在`DDPG`算法的基础上引入了下面三个技巧

- 截断的Double Q-Learning：缓解Critic的高估问题

  额外引入一个结构相同的价值网络，分别使用两个价值网络计算TD目标，取两者较小者为最终的TD目标

- 目标策略网络的平滑正则化：缓解Critic局部异常值对训练的影响

  往动作中添加噪声，可以避免价值网络局部异常值对训练的影响。通常取$\epsilon \sim clip(N(0,\sigma),-c,c)$，即带截断的正态分布噪声，防止噪声过大。

  > 还记得$\epsilon-greedy$吗？注意这两者的区别。此处的噪声相当于给输出的动作添加一个噪声，而$\epsilon-greedy$则是有一定的概率没有执行既定策略

- 策略网络和目标网络的延迟更新：待Critic更稳定些再更新Actor和目标网络

  实验表明应当让策略网络μ和三个目标网络的更新频率小于Critic网络。于是每轮更新一次Critic网络，但是每隔k轮才更新一次策略网络和三个目标网络

训练流程与DDPG类似。

多智能体强化学习

从MDP到Dec-POMDP：多智能体强化学习任务建模

- 顾名思义`Decentralized Partially Observable Markov Decision Process`：去中心化即多智能体 + 部分观测。
- 强化学习建模总结：（智能体数量，观测完整性，奖赏设计）
  - $MDP<S,A,T,R>$：单智能体
  - $POMDP<S,A,T,O,R>$：单智能体，部分观测
  - $Dec-POMDP<D,S,A,T,O,R>$：多智能体，部分观测，共享奖赏
  - $POSG<D,S,A,T,O,R>$：多智能体，部分观测，独立奖赏
- 观测往往只能获得状态的一部分
- 历史（history）和置信状态（belief）

  历史是智能体的**动作-观测序列**（可以理解为智能体获取的所有信息）

  置信状态$b(s_t)=Pr(s_t|h_t)$，表示在某条历史下智能体可能所处状态的概率分布。可以由历史来计算置信状态，从而近似“真正的状态”
- 在`POMDP`中，策略是$\pi:H\rightarrow A$，并且往往使用循环神经网络（RNN，LSTM）来处理历史H。

多智能体的策略求解：学习的方式

1. 完全集中式
   - 思路：将多个智能体看作单个智能体
   - 优势：信息丰富；隐含合作
   - 劣势：求解复杂（维数增加）；惰性智能体（任务集中于部分智能体，其他摆烂）

2. 完全分布式
    - 优势：简单快捷；要求低（不需要中央控制器或是通信环境）
    - 劣势：环境不稳定；虚假奖赏
3. 集中式训练，分布式执行
    - 评估器利用全局信息，对所有智能体的动作进行评估优化
    - 策略网络利用部分观测，为各个智能体独立决策
    - 训练时能够**利用全局信息优化各个独立的策略网络**，执行时智能体又可以独立运行

- 同构与参数共享（减少训练策略网络的成本）

从DQN到VDN：解决多智能体合作问题

- `VDN`(Value Decomposition Networks)顾名思义，值分解网络
- 核心：正确分解团队奖赏，促进个体学习
    - 集中式和分布式的本质问题：共享奖赏无法正确分解到个体奖赏，智能体无法得到正确反馈
    - 解决思路：假设全局奖赏、Q函数等于所有智能体的奖赏、Q函数之和
        - 奖赏的加性假设
        - 全局Q函数分解
        - 集中式学习，分布式执行：全局Q值根据总奖赏进行更新，每个智能体根据自己的Q值函数决策
- 技巧1：循环神经网络 LSTM
    - 借助“历史”得到智能体的“置信状态”，来近似当前状态
        - LSTM等循环神经网络能够记忆历史状态，并根据最新的（观测，动作）对状态进行更新
        - LSTM替换DQN中最后一个全链接层来提高鲁棒性和泛化能力
- 技巧2：Dueling DQN
    - 问题：如何更有效率的更新Q函数
    - 解决方法：将Q网络分为价值函数层V和优势函数层A分别进行更新。
        - $Q(s,a;\theta,\alpha,\beta)=V(s;\theta,\beta)+A(s,a;\theta,\alpha)$
- 技巧3：参数共享
    - 问题1：多智能体环境下可能出现“惰性智能体”问题。
    - 问题2：随着智能体的增多，网络规模不断增大。
    - 解决方法：同质智能体共享同一个Q函数网络。
- 通信：多智能体强化学习中的重要辅助设定
    - 通信对环境有要求
    - 低层通信：共享观测
    - 高层通信：共享历史

        > 信息共享实现隐式通信

从VDN到QMIX：改进VDN中的值函数分解

- 核心：扩展Q值分解方式
  - 相比于简单的加法，使用神经网络（函数嵌套）来分解全局Q值函数

从DDPG 到 MADDPG：解决多智能体问题

- 目标：最大化自己的总奖赏
- 所有智能体获得各自的奖赏
- 核心思想：集中式学习，分布式执行
  - 智能体拥有**独立**的`Actor`网络（用于决策）和`Critic`网络（用于学习异构奖赏）
  - 智能体的`Actor`网络μ_i，输入为自身的观测o_i，输出为自身的动作a_i
  - 智能体的`Critic`网络Q_i，输入为所有智能体的联合观测$(o_1,\ldots,o_n)$和联合动作$(a_1,\ldots,a_n)$，输出为动作对应的Q值$Q_i(o_i,a_i)$

    > 集中式学习的“集中”关键不在于是不是一个网络，而是网络的输入信息是个体的还是整体的

- `actor`网络的TD目标只换自己的动作，`critic`网络的TD目标需要换所有智能体的动作

飞控、动捕系统、虚实结合平台

略