深度强化学习笔记

基础知识

第一章习题：1、A；2、B；3、C；4、B；5、1
第二章习题：1、13.6；2、按一元正态分布采样 $f(x)$ 求均值；
3、构造随机变量 $Z_i = 4f(X_i, Y_i) - \pi$，易知其满足 Bernstein 概率不等式条件，可得

$$\mathbb{P}\left(\left|\frac{1}{n}\sum_{i=1}^{n} 4f(X_i, Y_i) - \pi \geq \epsilon\right|\right) \leq \exp\left(-\frac{\epsilon^2 n/2}{4\pi - \pi^2 + \epsilon\pi/3}\right)$$

令 $\epsilon = \frac{C}{\sqrt{n}}$，则右式为 $\exp\left(-\frac{C^2/2}{v + Cb/(3\sqrt{n})}\right)$，当 $n \to \infty$ 时概率上界为 $\exp\left(-\frac{C^2}{2v}\right)$，则对任意小的概率 δ，选取 $C = \sqrt{-2v\ln\delta}$ 可以使得 $|q_n - \pi|$ 大于 $\frac{C}{\sqrt{n}}$ 的概率不超过 δ，即为原题定理。
4、$n = 1$ 时成立，假设 $n = i - 1$ 时亦成立，

$$q_n = (1 - \frac{1}{n}) \cdot q_{n-1} + \frac{1}{n} \cdot f(x_n) = \frac{n-1}{n} \cdot \frac{1}{n-1}\sum_{i=1}^{n-1} f(x_i) + \frac{1}{n} \cdot f(x_n) = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

第三章习题：1、C；2、D；3、B；4、E
第四章习题：1、B；2、A；3、A；4、789，下；5、18，2；6、B，A；7、A
第五章习题：1、A；2、因为每一步都会利用策略函数 π 采样得到 a 后计算 TD 目标；3、因为更接近无偏估计 u_t

实际训练神经网络的时候,总是用 SGD（及其变体）而不用 GD。主要原因是 GD 用于非凸问题会陷在鞍点（saddle point），收敛不到局部最优。而 SGD 和小批量（mini-batch）SGD 可以跳出鞍点，趋近局部最优。另外，GD 每一步的计算量都很大，比 SGD 大 n 倍, 所以 GD 通常很慢(除非用并行计算)。

设 X 是 d 维随机变量，它的取值范围是集合 $\Omega \subset \mathbb{R}^d$。函数 $p(x)$ 是 X 的概率密度函数，设 $f : \Omega \mapsto \mathbb{R}$ 是任意的多元函数，它关于变量 X 的期望是：

$$\mathbb{E}_{X\sim p(\cdot)}[f(X)] = \int_{\Omega} p(x) \cdot f(x) \mathrm{d}x$$

可以在集合 Ω 上均匀采样利用蒙特卡洛计算其定积分，但收敛速度较慢，可以按照概率 $p(x)$ 进行非均匀采样，记作 $x_1, \cdots, x_n \sim p(\cdot)$，并对函数值 $f(x_1), \cdots, f(x_n)$ 求平均 q_n 作为期望 $\mathbb{E}_{X\sim p(\cdot)}[f(X)]$ 的估计值。
如果按照该方式进行采样需要存储 n 个函数值占用内存，可以初始化 $q = 0$，从 $t = 1$ 到 n，依次计算

$$q_t = \left(1 - \frac{1}{t}\right) \cdot q_{t-1} + \frac{1}{t} \cdot f(x_t)$$

易证这样得到的 q_n 等于 $\frac{1}{n}\sum_{i=1}^{n} f(x_i)$，可以进一步把上式中的 $\frac{1}{t}$ 替换为 α_t，得到公式

$$q_t = (1 - \alpha_t) \cdot q_{t-1} + \alpha_t \cdot f(x_t)$$

这个公式叫做 Robbins-Monro 算法，其中 α_n 称为学习步长或学习率，只要 α_t 满足下面的性质，就能保证算法的正确性

$$\lim_{n\to\infty}\sum_{t=1}^{n}\alpha_t=\infty,\quad \lim_{n\to\infty}\sum_{t=1}^{n}\alpha_t^2<\infty$$

我们可以用蒙特卡洛近似期望来理解随机梯度算法，神经网络的训练可以定义为如下优化问题

$$\min_{\omega}\mathbb{E}_{X\sim p(\cdot)}[L(X;\omega)]$$

目标函数 $\mathbb{E}_X[L(X;\omega)]$ 关于 ω 的梯度是（微分算子和积分算子的交换顺序需满足一定条件）

$$g\triangleq\nabla_{\omega}\mathbb{E}_{X\sim p(\cdot)}[L(X;\omega)]=\mathbb{E}_{X\sim p(\cdot)}[\nabla_{\omega}L(X;\omega)]$$

直接计算梯度 g 通常会比较慢，为了加速计算，可以对期望

$$g=\mathbb{E}_{X\sim p(\cdot)}[\nabla_{\omega}L(X;\omega)]$$

做蒙特卡洛近似，把得到的近似梯度 $\tilde{g}$ 称作随机梯度（stochastic gradient），用 $\tilde{g}$ 代替 g 来更新 ω，因为 $\mathbb{E}[\tilde{g}]=g$，它是 g 的一个无偏估计。在实际应用中，样本真实的概率密度函数 $p(x)$ 一般是未知的，在训练神经网络时，我们通常会手机一个训练数据集 $\mathcal{X}=\{x_1,\cdots,x_n\}$，并求解经验风险最小化问题

$$\min_{\omega}\frac{1}{n}\sum_{i=1}^{n}L(x_i;\omega)$$

这相当于用下面这个概率质量函数代替真实的 $p(x)$

$$p(x)=\begin{cases}\dfrac{1}{n}, & \text{if } x\in\mathcal{X}\\ 0, & \text{if } x\notin\mathcal{X}\end{cases}$$

强化学习的数学基础和建模工具是马尔可夫决策过程（Markov decision process，MDP）。一个 MDP 通常由状态空间 $\mathcal{S}$、动作空间 $\mathcal{A}$、状态转移函数 p、奖励函数 r 和折扣因子 γ 等组成。

书中提到“知道这局游戏的历史记录（即每一步是怎么走的），并不会给你提供额外的信息”，但是对手的风格也是重要信息，alpha go只利用前六步信息，是出于计算量的考虑还是信息和误差的考量?

通常假设奖励是当前状态 s、当前动作 a、下一时刻状态 s' 的函数，把奖励函数记作 $r(s,a,s')$。有时假设奖励仅仅是 s 和 a 的函数，记作 $r(s,a)$。我们总是假设奖励函数是有界的，即对于所有 $a\in\mathcal{A}$ 和 $s,s'\in s$，有 $|r(s,a,s')|<\infty$。

状态转移概率函数（state transition probability function），通常假设为平稳的，即不随着时刻 t 变化

$$p_t(s'\mid s,a)=\mathbb{P}(S'_{t+1}=s'\mid S_t=s,A_t=a)$$

轨迹（trajectory）是指一回合（episode）游戏中，智能体观测到的所有的状态、动作、奖励:

$$s_1,a_1,r_1,\quad s_2,a_2,r_2,\quad s_3,a_3,r_3,\cdots$$

回报（return）是从当前时刻开始到本回合结束的所有奖励的总和，所以回报也叫做累计奖励（cumulative future reward）。使用折扣回报（discounted return）作为折扣后的总奖励：

$$U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \cdots$$

MDP 的时间步可以是有限期（finite-horizon）或无限期（infinite-horizon）。有限期 MDP 存在一个终止状态（terminal state），该状态被智能体触发后，一个回合（episode）结束。与之对应的是无限期 MDP，即环境中不存在终止状态，在折扣率 $\gamma \geq 1$ 时会导致回报趋于无穷。

假设对于所有的 $s \in \mathcal{S}$ 和 $a \in \mathcal{A}$，回报函数有界 $|r(s,a)| < b$，那么对于 $\gamma \in [0,1)$，且 $r(s,a)$，有这样的性质：

$$\left| \lim_{n \to \infty} \sum_{i=t}^{n} \gamma^{i-t} r_i \right| \leq \frac{b}{1-\gamma}$$

假设观测到状态 s_t，选中动作 a_t，则回报 U_t 关于随机变量求条件期望，得到

$$Q_\pi(s_t, a_t) = \mathbb{E}_{S_{t+1}, A_{t+1}, \cdots, S_n, A_n}[U_t \mid S_t = s_t, A_t = a_t]$$

条件期望的结果 $Q_\pi(s_t, a_t)$ 被称作动作价值函数（action-value function），更准确地说应该是动作状态价值函数。

为了排除掉策略 π 的影响，只评价当前状态和动作的好坏，可以利用最优动作价值函数（optimal action-value function）：

$$Q_*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t), \quad \forall s_t \in \mathcal{S}, a_t \in \mathcal{A}$$

如果只想知道当前状态 s_t 是否对自己有利，可以利用状态价值函数（state-value function）：

$$V_\pi(s_t) = \mathbb{E}_{A_t \sim \pi(\cdot \mid s_t)}[Q_\pi(s_t, A_t)] = \sum_{a \in \mathcal{A}} \pi(a \mid s_t) \cdot Q_\pi(s_t, a)$$

状态价值函数 $V_\pi(s_t)$ 也是回报 U_t 的期望：

$$V_\pi(s_t) = \mathbb{E}_{A_t, S_{t+1}, A_{t+1}, \cdots, S_n, A_n}[U_t \mid S_t = s_t]$$

价值学习

DQN 与 Q 学习

最优动作价值函数可以用来做控制，一旦知道状态 s_t，可以根据对动作的打分进行选择。

近似学习 Q_* 最有效的方法是深度 Q 网络（deep Q network，DQN），记作 $Q(s,a;\omega)$。训练 DQN 最常用的算法是时间差分（temporal difference，TD）。

假如在出发前，模型预估北京到上海需要 $\hat{q} = 14$ 小时，经过 $r = 4.5$ 小时从北京到达济南后，再让模型预测从济南到上海需要 $\hat{q}' = 11$ 个小时，则整个旅程的最新估计时间为

$$\hat{y} \triangleq r + \hat{q}' = 4.5 + 11 = 15.5$$

TD 算法将 $\hat{y} = 15.5$ 称为 TD 目标（TD target），它比最初的预测 $\hat{q} = 14$ 更可靠，因为它是纯粹估计的，而 TD 目标中含有事实的成分。我们希望估计值 $\hat{q}$ 尽量接近 TD 目标 $\hat{y}$，所以用两

者差的平方作为损失函数：

$$L(\omega)=\frac{1}{2}\big[Q(\text{〃北京〃},\text{〃上海〃};\omega)-\hat{y}\big]^2$$

此处把 $\hat{y}$ 看做常数，尽管它依赖于 ω。计算损失函数的梯度：

$$\nabla_\omega L(\omega)=\underbrace{(\hat{q}-\hat{y})}_{\delta}\cdot\nabla_\omega Q(\text{〃北京〃},\text{〃上海〃};\omega)$$

此处的 $\delta=\hat{q}-\hat{y}=14-15.5=-1.5$ 称作 TD 误差（TD error）。

由下列两式推导

$$U_t=R_t+\gamma\cdot\underbrace{\sum_{k=t+1}^{n}\gamma^{k-t-1}\cdot R_k}_{=U_{t+1}},\qquad Q_*(s_t,a_t)=\max_\pi\mathbb{E}[U_t\mid S_t=s_t,A_t=a_t]$$

可得下列定理，该定理是最优贝尔曼方程（optimal Bellman equations）的一种形式

$$\underbrace{Q_*(s_t,a_t)}_{U_t\text{的期望}}=\mathbb{E}_{S_{t+1}\sim p(\cdot|s_t,a_t)}[R_t+\gamma\cdot\underbrace{\max_{A\in\mathcal{A}}Q_*(S_{t+1},A)}_{U_{t+1}\text{的期望}}\mid S_t=s_t,A_t=a_t]$$

贝尔曼方程的右边是个期望，我们可以对期望做蒙特卡洛近似，当智能体执行动作 a_t 后，环境通过状态转移函数 $p(s_{t+1}\mid s_t,a_t)$ 计算出新状态 s_{t+1}，那么当我们观测到 s_t,a_t,s_{t+1} 后则奖励 r_t 也被观测到，因此可以计算

$$r_t+\gamma\cdot\max_{a\in\mathcal{A}}Q_*(s_{t+1},a)$$

它可以看做是右式期望的近似。因此，如果用神经网络 $Q(s,a;\omega)$ 替换最优动作价值函数 $Q_*(s,a)$ 可以得到

$$\underbrace{Q(s_t,a_t;\omega)}_{\text{预测}\hat{q}_t}\approx\underbrace{r_t+\gamma\cdot\max_{a\in\mathcal{A}}Q(s_{t+1},a;\omega)}_{\text{TD 目标}\hat{y}_t}$$

应当鼓励 $\hat{q}_t\triangleq Q(s_t,a_t;\omega)$ 接近 $\hat{y}_t$，定义损失函数

$$L(\omega)=\frac{1}{2}[Q(s_t,a_t;\omega)-\hat{y}_t]^2$$

假设 $\hat{y}$ 是常数，计算 L 关于 ω 的梯度

$$\nabla_\omega L(\omega)=\underbrace{(\hat{q}_t-\hat{y}_t)}_{\text{TD 误差}\delta_t}\cdot\nabla_\omega Q(s_t,a_t;\omega)$$

做一步梯度下降，可以让 $\hat{q}_t$ 更接近 $\hat{y}_t$

$$\omega\leftarrow\omega-\alpha\cdot\delta_t\cdot\nabla_\omega Q(s_t,a_t;\omega)$$

收集训练数据：我们可以用任何策略函数 π 去控制智能体与环境交互，这个 π 就叫做行为策略（behavior policy）。比较常用的是 ϵ-greedy 策略（初始时将 ϵ 设置得比较大，然后逐渐衰减）：

$$a_t = \begin{cases} \arg\max_a Q(s_t, a; \omega), & \text{以概率}(1-\epsilon); \\ \text{均匀抽取 } \mathcal{A} \text{ 中的一个动作}, & \text{以概率 } \epsilon. \end{cases}$$

把智能体在一局游戏中的轨迹记作 $s_1, a_1, r_1, \quad , s_2, a_2, r_2, \cdots, \quad s_n, a_n, r_n$。把一条轨迹划分成 n 个 (s_t, a_t, r_t, s_{t+1}) 这种四元组，存入数组，这个数组叫做经验回放数组（replay buffer）。

更新 DQN 参数 ω：随机从经验回放数组中取出一个四元组，记作 (s_j, a_j, r_j, s_{j+1})，执行下列步骤进行参数更新：

1. 对 DQN 做正向传播，得到 Q 值：

$$\hat{q}_j = Q(s_j, a_j; \omega_{\text{now}}) \text{ 和 } \hat{q}_{j+1} = \max_{a \in \mathcal{A}} Q(s_{j+1}, a; \omega_{\text{now}})$$

2. 计算 TD 目标和 TD 误差：

$$\hat{y}_j = r_j + \gamma \cdot \hat{q}_{j+1} \text{ 和 } \delta_j = \hat{q}_j - \hat{y}_j$$

3. 对 DQN 作反向传播，得到梯度：

$$g_j = \nabla_\omega Q(s_j, a_j; \omega_{\text{now}})$$

4. 做梯度下降更新 DQN 的参数：

$$\omega_{\text{now}} \leftarrow \omega_{\text{now}} - \alpha \cdot \delta_j \cdot g_j$$

   可以在智能体每执行一个动作之后，对 ω 做几次更新，也可以在每完成一局游戏后，对 ω 做几次更新。

利用表格法实现 Q 学习，首先用表格表示 Q_*，即各状态下各动作的最优动作价值函数，用下列公式更新表格中元素

$$\hat{y}_t \triangleq r_t + \gamma \cdot \max_{a \in \mathcal{A}} \tilde{Q}(s_{t+1}, a); \qquad \tilde{Q}(s_t, a_t) \leftarrow (1-\alpha) \cdot \tilde{Q}(s_t, a_t) + \alpha \cdot \hat{y}_t$$

行为策略（behavior policy）的作用是收集经验（experience），即观测的状态、动作、奖励；目标策略是用来控制智能体的策略函数。同策略（on-policy）是指利用相同的行为策略和目标策略，异策略（off-policy）是指用不同的行为策略和目标策略。

异策略的好处是可以用行为策略收集经验，将 (s_t, a_t, r_t, s_{t+1}) 存储作为经验回放数组，利用这些数据训练目标策略被称作经验回放（experience relpay）。

SARSA 算法

传统的强化学习用 Q_π 作为确定性的策略控制智能体，但是现在 Q_π 通常被用于评价策略的好坏，而非用于控制智能体。Q_π 常与策略函数 π 结合使用，被称作 actor-critic（演员-评委）方法。

SARSA 算法的表格法与 Q 学习类似，利用下列贝尔曼方程

$$Q_\pi(s_t, a_t) = \mathbb{E}_{S_{t+1}, A_{t+1}}[R_t + \gamma \cdot Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s_t, A_t = a_t]$$

其中左式可近似为表格中的 $q(s_t, a_t)$，右式可根据 s_{t+1} 和策略 π 做随机抽样，得到新的动作

$$\tilde{a}_{t+1} \sim \pi(\cdot \mid s_{t+1})$$

并用观测到的 r_t、s_{t+1} 和计算出的 $\tilde{a}_{t+1}$ 对期望做蒙特卡洛近似，然后用表格 q 近似 Q_π 得到

$$\hat{y}_t \triangleq r_t + \gamma \cdot q(s_{t+1}, \tilde{a}_{t+1})$$

并用下式更新表格中 (s_t, a_t) 位置上的元素

$$q(s_t, a_t) \leftarrow (1-\alpha) \cdot q(s_t, a_t) + \alpha \cdot \hat{y}_t$$

然后用某种算法更新策略函数，注意这是隐性的更新，通过更新 Q 函数来间接改进策略。

SARSA 算法的目标是学到表格 q 作为动作价值函数 Q_π 的近似，经验回放数组里的经验 (s_j, a_j, r_j, s_{j+1}) 是过时的行为策略 π_{old} 收集到的，与当前策略 π_{now} 及其对应的价值 $Q_{\pi_{\text{now}}}$ 对应不上，因此不能使用经验回放，只能使用同策略。

可以用神经网络 $q(s, a; \omega)$ 来近似 $Q_\pi(s, a)$，称其为价值网络（value network），其训练流程基本与训练 DQN 完全相同，只是需要根据当前策略做抽样

$$\tilde{a}_{t+1} \sim \pi_{\text{now}}(\cdot \mid s_{t+1})$$

SARSA 计算 TD 目标只用到一个奖励 r_t，这样得到的 $\hat{y}_t$ 叫做单步 TD 目标，可将其扩展为多步 TD 目标。易知

$$U_t = \left(\sum_{i=0}^{m-1} \gamma^i R_{t+i} \right) + \gamma^m U_{t+m}$$

动作价值函数 $Q_\pi(s_t, a_t)$ 是回报 U_t 的期望，而 $Q_\pi(s_{t+m}, a_{t+m})$ 是回报 U_{t+m} 的期望，则可以得到下列定理：
设 R_k 是 S_k、A_k、S_{k+1} 的函数，$\forall k = 1, \cdots, n$，则

$$\underbrace{Q_\pi(s_t, a_t)}_{U_t\text{的期望}} = \mathbb{E}\left[\left(\sum_{i=0}^{m-1} \gamma^i R_{t+i} \right) + \gamma^m \cdot \underbrace{Q_\pi(S_{t+m}, A_{t+m})}_{U_{t+m}\text{的期望}} \middle| S_t = s_t, A_t = a_t \right]$$

所以已知当前状态 s_t，用策略 π 控制智能体与环境交互 m 次，得到轨迹（不需要 r_{t+m}）

$$r_t, \quad s_{t+1}, a_{t+1}, r_{t+1}, \quad \cdots, \quad s_{t+m-1}, a_{t+m-1}, r_{t+m-1}, \quad s_{t+m}, a_{t+m}$$

后可以计算 m 步 TD 目标用以更新价值网络和策略

$$\hat{y}_t = \left(\sum_{i=0}^{m-1} \gamma^i r_{t+i} \right) + \gamma^m \cdot q(s_{t+m}, a_{t+m}; \omega)$$

训练价值网络 $q(s, a; \omega)$ 时，可以将一局游戏进行到底，计算回报 $u_t = \sum_{i=0}^{n-t} \gamma^i r_{t+i}$，使用 u_t 作为目标鼓励价值网络接近 u_t 不是 TD 方法，而是蒙特卡洛，因为这是用实际观测 u_t 去近似期望。

- 蒙特卡洛的好处是无偏性：u_t 是 $Q_\pi(s_t, a_t)$ 的无偏估计，由于 u_t 的无偏性，拿其作为目标训练价值网络，得到的价值网络也是无偏的。
- 蒙特卡洛的坏处是方差大：随机变量 U_t 依赖于 S_{t+1}，A_{t+1}，$\cdots$，S_n，A_n 这些随机变量，其中不确定性很大，可能导致观测值 u_t 实际上离 $\mathbb{E}[U_t]$ 很远，因此拿 u_t 作为目标训

练价值网络收敛会很慢。

SARSA 算法中用价值网络自己做出的估计去更新价值网络自己，这属于自举（bootstrapping）。

- 自举的好处是方差小。单步 TD 目标的随机性只来自于 S_{t+1} 和 A_{t+1}，而回报 U_t 的随机性来自于 S_{t+1}，A_{t+1}，$\cdots$，S_n，A_n。很显然单步 TD 目标的随机性较小，因此方差较小，用自举训练价值网络，收敛比较快。
- 自举的坏处是有偏差。价值网络 $q(s,a;\omega)$ 是对动作价值 $Q_\pi(s,a)$ 的近似，假如 $q(s_{j+1},a_{j+1};\omega)$ 低估（或高估）真实价值 $Q_\pi(s_{j+1},a_{j+1})$，则会导致 $q(s_j,a_j;\omega)$ 低估（高估）$Q_\pi(s_j,a_j)$。也就是说，自举会让偏差从 (s_{t+1},a_{t+1}) 传播到 (s_t,a_t)。

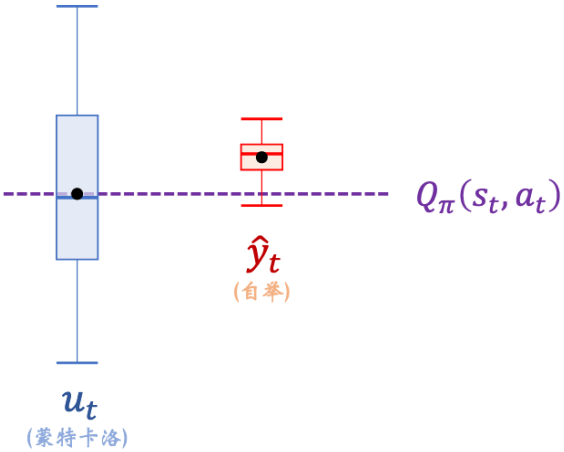

图 5.5: u_t 和 $\hat{y}_t$ 的箱型图（boxplot）示意。