强化学习大作业实验报告

華東師範大學

学　　院：数据科学与工程学院

小组成员：赵倩仪 51255903037　李新雨 51255903028

宋晏如 51255903111　刘冬煜 51255903088

李康恬 51255903049　毛科捷 51255903071

《强化学习》实验报告

基于强化学习的众包任务推荐

数据科学与工程学院 2022 级

赵倩仪 51255903037 李新雨 51255903028 宋晏如 51255903111

刘冬煜 51255903088 李康恬 51255903049 毛科捷 51255903071

一、 实验描述

1. 背景知识

众包模型旨在从相对开放和快速变化的参与者群组中获得服务。众包通常使用互联网吸引和划分工作任务给参与者，并等待工作完成后将结果聚合。非盈利机构可以使用众包模型开发公共服务，比如维基百科。此外，当前也存在一些商用系统。比如，Amazon 推出了 Amazon MTurk 系统，提供了收费的众包服务，以吸引人工解决计算机难以处理的问题。在 Amazon MTurk 上，请求者首先发布任务（比如：标注图片），并给出报酬（完成一张图片付费多少）。当参与者进入系统时，平台会为该参与者展示可获取的任务列表，并可查看该任务的详细信息，包括标题、描述、创建时间、结束时间等。最后，由参与者选择任务，并决定是否完成。

2. 问题描述

在众包系统中，当参与者进入平台时，涉及到任务排列的问题，即如何根据参与者选择展示的任务。为简化问题，我们假设系统只向参与者推荐一个任务。由于系统是盈利性的，因此众包系统需要同时满足参与者和请求者两方的利益：（1）参与者可以找到更多相关的、感兴趣的任务，以赚取更多的报酬；（2）请求者发布的任务可以得到更多、更高质量的回答。

与此同时，由于参与者和发布者是动态变化的，因此，系统在分配任务时应能考虑并处理这种动态性。

3. 具体任务

（1）如何将强化学习技术应用到众包任务推荐中，并最大化**参与者**的利益？

（2）如何将强化学习技术应用到众包任务推荐中，并最大化**请求者**的利益？

4. 实验要求

给出具体的实验流程和设计细节，并给出最终的实验结果。

要求同时实现并对比三种类别的方法：(1)**Value-based methods**; (2) **Policy-based methods**; (3) **Actor-critic methods**。

最多六人组队完成，于 7 月 14 日中午 12 点前提交实验报告至指定邮箱。

5. 实验数据

附件包括了 sample_read_data.py 文件用来读取数据（可做参考），worker_quality.csv 文件包含了 worker 的 quality 属性，project_list.csv 文件是 project_id + project_answer_num，project 和 entry 文件夹分别包含了 project 和 worker 的信息。为简化问题，可直接参考 sample_read_data.py 内容读取数据。在获取了数据后，自行划分训练集、验证集和测试集，并在实验报告中说明细节。

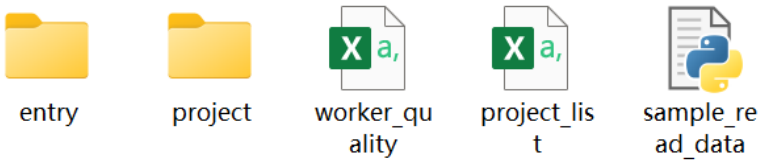

Figure 1 提供的实验数据

6. 提示

（1）如何针对参与者和请求者分别定义状态、行为、奖励等？

（2）如何定义当前状态和下一步状态？

（3）如何构造 Q 函数？

（4）仔细分析数据，思考如何构造特征。

（5）对任务创建时间、到期时间、以及 worker 到达时间进行排序，模拟 worker 的兴趣和完成情况。

二、 实验分析与设计

1. 数据分析

我们使用提供的真实数据进行实验，对 data 目录下的数据集文件及其所含内容进行整理，并进行了 project 和 worker 信息关键部分的提取，情况如下表所示。

其中，对于 project，我们主要选取 category 和 industry 为它的主要特征；对于 worker，则主要依据他的 quality 值和其与项目的联系情况来进行后续的训练分析。

Table 1 数据说明

文件名	文件内容/数据信息	
worker_quality.csv	worker_id	
	worker_quality	
project_list.csv	project_id	
	project_answer_num	
project 文件夹	5335 个 project_projectId.txt	
entry 文件夹	22805 个 entry_projectId_k.txt_	
project_projectID.txt	单个 project 信息	sub_category（项目所属子类别）
		category（项目所属类别）
		entry_count（所需参与者数量）
		start_date（项目开始时间）
		deadline（项目截止时间）
		industry
		……
entry_projectID_cnt.txt	多个 worker 信息 (项目对应参与者)	entry_create_at （参与者登录时间）
		id（项目参与者的 worker_id）
		……

2. 问题抽象

（1）设置

我们以时间为顺序，根据任务创建时间、到期时间、以及 worker 到达时间，构建场景模拟 worker 的兴趣和完成情况。任务以创建时间依序排列，在t_i时刻，有任务集合T_i等待被完成，有参与者集合W_i可被分配任务，系统根据算法处理结果，将任务依先后顺序分配给参与者。

（2）状态（State）

由于项目任务的到达存在时序逻辑关系，而推荐系统本身是实时的，所以我们定义 State 为当前新到达任务的抽象表征。而实际训练时使用 n-gram 模型，将历史$n-1$个状态与当前状态拼接作为输入送到模型中，得到行为的预测。

（3）项目任务的表征

项目存在若干特征，包括类别、子类别、产业、地域等数据。由于众包平台的线上交易等特点，我们将影响较小的地域、时间等信息排除，仅使用类别（category）与产业（industry）的$one-hot$表示的拼接作为项目的表征。

例如，一个任务的 category（$\in [1,10]$）为2（即$[0,1,0,0,0,0,0,0,0,0]$），该任务的 industry（$\in [1,3]$）为3（即$[0,0,1]$），那么它对应的表征向量即为$[0,1,0,0,0,0,0,0,0,0,0,0,1]$。

（4）行为（Action）

我们认为，推荐系统做的工作只有一件事，即将新产生的项目任务 project（包括需要标注的数据、需要设计的 logo 等）推荐给参与者 worker（数据标注者，logo 设计者）。因此，我们定义的 Action 为，将当前状态下的任务被推荐给了第k个 worker。

（5）状态转移

由于数据是时序的，因此我们认为推荐后的任务已被发配给对应的用户等待完成，接着开始执行下一个任务的推荐工作。因此在将所有任务按到达时间排序后，不管执行了什么行为，状态从第i个任务的表征转化为第$i+1$个任务的表征。特殊的，当所有任务的推荐工作都结束后，强化学习算法结束当前的 epoch，进行下一个 epoch 的学习。

（6）奖励值（Reward）

对于 worker 和 requester 两个不同利益最大化的任务，我们采取了不同的奖励值设置方案。

如果期望最大化 worker 的收益，即尽可能向 worker 推荐自己感兴趣的问题，那么如果算法向一个回答过 N 个问题的用户推荐了 category 为c的问题，而该用户回答过n_c个同类型的问题，那么奖励值为$\frac{n_c}{N}$。

如果期望最大化 requester 的收益，即尽可能收到质量更高的问题。

从线性的角度考虑，我们根据 worker_quality 提供的用户质量指标，将奖励值加入一项，即用户质量为$q \in [0,1]$时，获得的奖励值为$\alpha\frac{n_c}{N} + (1-\alpha)q$，记作 linear，其中$\alpha \in [0,1)$。

从非线性的角度，我们设计另外两种奖励值，分别记作 non-linear 1 和 non-linear 2，其表达式为$\frac{n_c}{N}q$和$1-\left(1-\frac{n_c}{N}\right)(1-q)$。

在后续的实验中，我们将分别根据这三类奖励值进行实验，最终分析评估出较优的一项奖励值。

（7）参与者与请求者

综上所述，对于两项任务，针对参与者与请求者，我们设置不同的奖励值，参与者的奖励值为$\frac{n_c}{N}$，请求者设置了三类奖励值分别为$\alpha\frac{n_c}{N} + (1-\alpha)q$、$\frac{n_c}{N}q$和$1-\left(1-\frac{n_c}{N}\right)(1-q)$。

参与者与请求者在状态、行为上的定义相同，即

- 状态：当前新到达任务的抽象表征；
- 行为：当前状态下的任务推荐。

3. API 设计

基于数据的分析和问题的抽象，我们对应设计实验相关的环境与数据的 API，其具体说明在 Table 2 与 Table 3 中有所描述。

（1）Environment

Table 2 Environment 相关 API 及说明

API	说明
reset(self) -> None	每个 epoch 开始时调用，清空上一个 epoch 的临时数据，并将 state 置为初始状态。
sample(self) -> Action	选择一个可行的状态。即将当前任务随机推荐给一个 worker。

perform(self, action: Action) -> float	执行行为 action，更新状态，并返回当前行为的奖励值以供模型训练。
is_done(self) -> bool	判断当前 epoch 是否终止（是否完成所有任务的推荐）。
get_state(self) -> npy.ndarray	返回当前状态，用于模型输入。
get_history_states(self, n: int) -> List[npy.ndarray]	返回历史最近的 n 个状态。用于 n-gram 模型获取模型历史输入和算法的优化方法。
get_state_dim(self) -> int	状态的维度，即 category 数+industry 数，用于搭建模型时输入层的设置。
get_output_dim(self) -> int	输出的维度，即 worker 数，用于搭建模型时输出层的设置。

（2）Data

Table 3 Data 相关 API 及说明

API	说明
get_data(self) -> None	读入数据。仅在训练开始前调用一次。
get_standard_reward(self, worker_id: int, project_id: int) -> float	获得标准奖励值，即$\frac{n_c}{N}$。
get_quality_reward(self, worker_id: int) -> float	获取 worker 的质量信息，即q

4. Value-Based（DQN）

（1）DQN 算法概述和核心思想

DQN（Deep Q-Network）是一种融合了深度学习和 Q-learning 的强化学习算法，用于解决马尔可夫决策过程（Markov Decision Process，MDP）问题。它的目标是学习一个动作值函数（action-value function），也称为 Q 函数（Q-function），以辅助智能体做出最优决策。

DQN 的核心思想是智能体通过与环境交互来收集经验，并使用经验回放和目标网络更新 Q 函数的参数。在每个时间步骤中，智能体根据当前状态选择动作，并观察奖励和下一个状态。通过计算 Q 值的目标（基于 Bellman 方程），智能体使用梯度下降方法来最小化预测 Q 值与目标 Q 值之间的差异。DQN 使用深度神经网络作为近似的 Q 函数。网络的输入是环境状态，输出是每个动作

的 Q 值。通常，卷积神经网络（CNN）用于处理具有图像输入的任务，而全连接神经网络（FCN）用于处理其他类型的状态。

（2）DQN 算法的探索性提升方法

DQN 还可以通过以下方法来提升探索性：

经验回放：DQN 引入了经验回放（experience replay）机制。智能体与环境进行交互时，将经验存储在经验回放缓冲区中。在训练过程中，随机从缓冲区中抽取一批经验样本进行训练。这样做的好处是减少样本间的相关性，平衡样本分布，提高训练的稳定性和效率。

ε-贪婪策略：DQN 使用 ε-贪婪策略来在探索（exploration）和利用（exploitation）之间进行平衡。在训练的早期阶段，智能体以一定的概率 ε 随机选择动作，以便探索环境；而在训练的后期阶段，智能体根据当前 Q 值选择最优动作，以提高性能。

5. Policy-Based（Reinforcement）

policy-based 的强化学习算法通过建模 policy，解决 value-based 算法无法处理连续 action 和解决随机策略的问题。

基于 policy-gradient 的思想，实现 policy 评价指标的最大化。对于评价指标$J(\theta)$，使用随机梯度上升方法$\theta_{t+1} = \theta_t + \alpha\nabla_\theta J(\theta)$更新其参数$\theta$，其中$\alpha$为学习率。对于这个梯度，因为没有真实的 value 与梯度，所以需要通过采样的方法进行估计，使其期望正比于真实梯度。因此 policy-based 算法主要任务之一便是用一个方法进行梯度估计。

Reinforce 算法是一个具体的 policy-based 算法，利用蒙特卡洛思想，每次采样一个 episode，并利用其返回值$G_t = \sum_{k=t+1}^{T} \gamma^{k-t} R$ 作为$Q_\pi(s_t|a_t)$，从而对θ进行更新，其中γ是权重超参数。对其梯度进行化简代入得$\theta_{t+1} = \theta_t + \alpha G_t \nabla \ln \pi(A_t|S_t, \theta_t)$。

6. Actor-Critic

（1）Actor-Critic 算法

Actor-Critic 算法是一种经典的强化学习算法，结合了策略评估（Critic）和策略改进（Actor），旨在解决在连续动作空间中的策略优化问题。该算法通过同时学习值函数和策略函数来提高强化学习的效率和性能。

在 Actor-Critic 算法中，Critic 部分负责评估当前策略的价值，使用值函数估计来衡量状态的好坏，指导策略的改进。Critic 网络通过学习从状态到值函数的映射，对策略进行评估和指导，帮助 Actor 网络优化策略。Actor 部分负责改

进策略，根据 Critic 的评估结果，采取更优的动作选择策略，以最大化长期累积奖励。

（2）合并网络的 Actor-Critic 算法

在传统的 Actor-Critic 算法中，通常使用两个独立的神经网络，一个用于 Actor 策略的学习，另一个用于 Critic 值函数的估计。然而，也有一种变体的 Actor-Critic 算法，将这两个网络合并为一个网络，共享网络参数。其主要目的是减少模型的复杂性和计算开销。通过共享网络参数，可以减少神经网络的层数和参数量，从而降低训练和推断的计算成本。

在合并网络的设计中，通常使用一个共享的隐藏层来提取状态特征，并在此基础上分别连接 Actor 和 Critic 的输出层。共享隐藏层可以学习通用的状态表示，以便 Actor 和 Critic 共享一些信息，并通过各自的输出层执行不同的任务。

具体而言，在合并网络中，网络的输入是状态观测，经过共享隐藏层后分为两个分支：一个分支连接到 Actor 的输出层，用于输出动作的概率分布；另一个分支连接到 Critic 的输出层，用于估计状态的值函数。通过这种方式，网络可以同时学习策略和值函数，实现 Actor 和 Critic 的协同训练。

合并网络的优点是节省了计算资源，减少了模型的复杂性和参数量。此外，共享隐藏层可以促进状态特征的学习和表示，提高模型的表达能力和泛化能力。

三、 实验内容

1. 参数设置

Table 4 参数设置表

参数	值
alpha	0.2
n_gram	5
gamma（DQN）	1.0
gamma（Reinforcement）	0.5
gamma（Actor-Critic）	0.99
epsilon（Reinforcement）	0.9
epsilon_decay（Reinforcement）	0.99
epsilon_min（Reinforcement）	0.01

2. 数据预处理

（1）数据集读取

分别从 worker_quality.csv 和 project_list.csv 中读入获取 worker 的 id 和质量以及所有 project 的 id；

利用 project 的 id 遍历 project 与 entry 目录下的数据，读入 project 的信息与 worker 参与 project 的情况；

填充 worker_quality: Dict[int, float]、worker_category: Dict[Tuple[int, int], int]、worker_project_cnt: Dict[int, int]三个 dictionary。

Table 5 worker 相关 dictionary 及其说明

dictionary	说明
worker_quality[worker_id]	worker 对应的 worker 的质量
worker_category[(worker_id, category)]	worker 对应参与过多少该类别的项目
worker_project_cnt[worker_id]	worker 对应参与的项目总数

（2）数据集划分

从读入的n条 project 的信息数据中随机取出$\frac{n}{6}$条作为验证集，其余$\frac{5n}{6}$条作为训练集进行训练。两组数据分别按任务到达时间进行排序，训练集依次作为$state$输入到网络中。

3. 实验主框架

（1）初始化环境与参数

首先调用 Environment 接口，对实验环境进行一个初始化（数据读取、初始化及相关参数的设置）。同时设置实验的相关参数（包括用于训练的算法、周期、奖励类型、优化器及学习率的设置）。

其中，算法包含 Policy-Based 的 DQM、Value-Based 的 Reinforcement 以及合并网络的 Actor-Critic；奖励类型包含 w、r、rn1、rn2，分别表示 worker、request（linear）、requester（non-linear 1）、requester（non-linear 2），对应的奖励值表达分别为$\frac{n_c}{N}$、$\alpha\frac{n_c}{N}+(1-\alpha)q$、$\frac{n_c}{N}q$和$1-\left(1-\frac{n_c}{N}\right)(1-q)$；优化器包含 Adam、RMSProp、Adagrad 以及 SGD。

（2）训练与绘图

在获取参数、算法模型与环境奖励之后，调用训练函数进行训练。最终，根据训练结果进行图表的绘制。

4. Value-Based（DQN）

（1）网络结构

构造一个四层 FCN。输入为智能体状态，即 project 的信息。输出为所有 action,每个输出层神经元对应一个 worker 的 Q 值。

（2）训练过程

a. 初始化网络，输入状态s_t，输出s_t下所有动作的 Q 值；

b. 利用策略(例如 E-greddy)，选择一个动作a_t，把a_t输入到环境中，获得新状态s_{t+1}和 reward；

c. 计算 target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; w)$；

d. 计算损失函数:$L = 1/2[y_t - Q(s, a; w)]^2$；

e. 更新网络，使得$Q(s_t, Q_t)$尽可能接近y_t，利用梯度下降做更新工作；

f. 输入新的状态，重复更新工作。

DQN 里没有 entropy 的概念，所以在本次实验中，该算法我们用 loss 代替 entropy。同时我们设置了一个_buffered_states 用于存放历史状态，每次输入时，不仅输入当前状态，还从缓冲区中抽取一批经验样本进行训练，来提升训练效率和稳定性。

5. Policy-Based（Reinforcement）

（1）网络结构

将环境的状态作为网络输入，输出为各 action 的概率。其中，我们使用历史状态序列作为输入，从而使结果在非随机策略的情况下更好。其中使用 Softmax entropy 对网络输出进行激活，从而使梯度计算更简单。

（2）训练过程

a. 获得一次对当前状态序列（在没有达到 n-gram 时 padding），将其输入神经网络，并得到输出的概率;

b. 得到概率最大的 action 序号，并取其概率对数，同时计算得到熵值作为 reward。返回 action 序号和概率的对数以及熵值;

c. 在环境中执行选取的 action，得到 reward;

d. 重复 1-3 若干次（达到上限步数或收敛），得到一个 episodes 的结果;

e. 对于该 episode 中估计的梯度进行计算。首先初始化 R=0，超参数为 gamma，输出的数组从最后一步开始遍历输出。记当前步为第 i 步，其对应的 reward 为 rewards[i]，对应的对数概率为 log_probs[i]。

```
    for i in reversed(range(n_steps)):
        R = gamma * R+ rewards[i]
        loss = loss - log_probs[i] * R
```

在遍历完后，取`loss = loss/n_steps`，并梯度回传进行更新。

f. 重复执行若干 episodes。

6. Actor-Critic

（1）网络结构

算法根据环境的状态维度和动作维度构建一个前馈神经网络（FCNet）作为 Actor-Critic 网络。

（2）训练过程

a. 初始化环境：重置环境，并获取初始状态和历史状态。

b. 进入循环直到环境结束：

i. 状态转换：根据当前状态和历史状态，将其转换为张量输入到网络中。

ii. Actor 网络：使用 Actor 部分的网络预测动作概率分布，并根据分布进行动作采样。

iii. 环境交互：执行采样的动作，并获取奖励、下一个状态和历史状态。

iv. Critic 网络：使用 Critic 部分的网络预测当前状态的值函数估计和下一个状态的值函数估计。

v. 计算目标和优势：根据奖励、下一个状态的值函数估计和终止标志计算目标和优势。

vi. 计算损失：根据动作概率、优势和值函数估计计算 Actor 和 Critic 的损失。

vii. 更新网络参数：使用优化器对损失进行反向传播，并更新网络参数。

viii.记录信息：记录每个时间步的奖励和动作概率分布的熵。

c. 返回算法名称：返回算法的名称作为标识。

四、 实验结果与分析

1. 训练集 reward 和 entropy

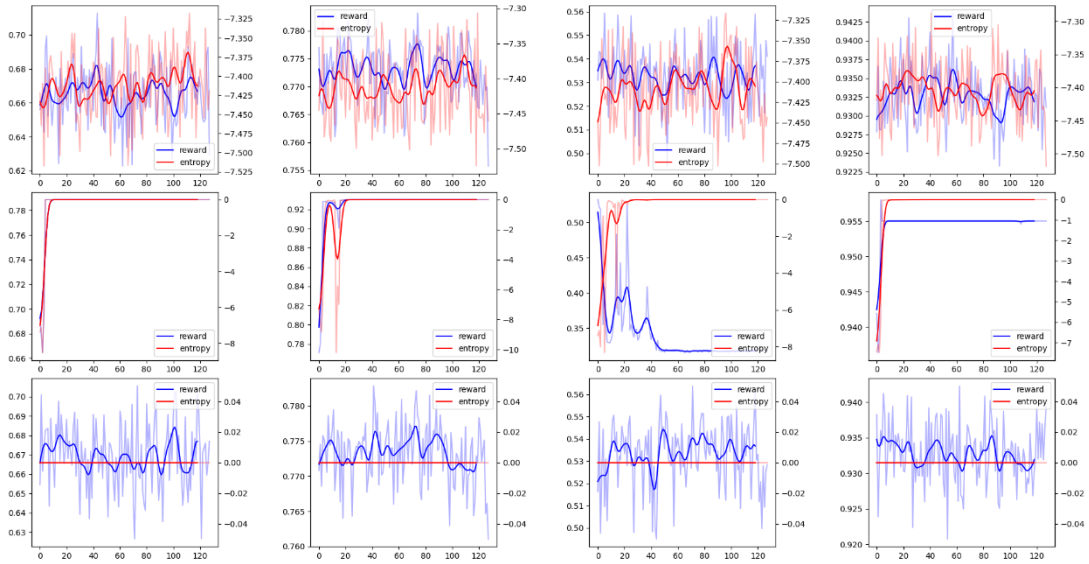

Figure 2 实验结果图一览（从上到下依次为 AC、Policy、Value，
从左到右依次为 w、r、rn1、rn2 的奖励值类型）

（1）任务一：最大化参与者的利益

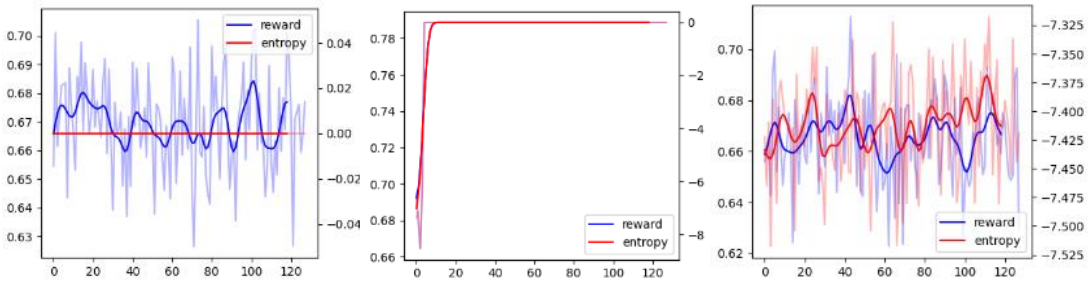

Figure 3 任务一 DQN、Reinforcement 和 AC 模型的 reward、entropy 曲线

如图 3 所示，针对最大化参与者的利益的任务，在训练数据过程中，可以看到 Value-Based 的 DQN 方法的 reward 值在反复震荡，均值在 0.67 左右，而其 entropy 保持为定值 0；Policy-Based 的 Reinforcement 方法的 reward、entropy 值收敛较快，且 reward 均值保持在 0.785 左右，entropy 值稳定在 0；Actor-Critic 方法的 reward 和 entropy 值均在反复震荡，其 reward 均值保持在 0.67 左右，entropy 均值一直为负数但有一定幅度的上升。

总体而言，对该部分的实验来说，在 120 轮的训练中，Reinforcement 模型的收敛速度和 reward 值均优于 DQN 和 Actor-Critic，其他两者均有反复震荡的情况，且 reward 值较接近，可能需要通过增加训练轮次和内部调参来进行优化。

（2）任务二：最大化请求者的利益

对于该任务，设置了三种类型的 reward，分别为 linear、non-linear 1 和 non-linear 2，在前文中有简单介绍，其奖励值的设置分别为$\alpha\frac{n_c}{N}+(1-\alpha)q$、$\frac{n_c}{N}q$和$1-\left(1-\frac{n_c}{N}\right)(1-q)$。针对三种奖励值，对应实验给出的结果（reward 曲线和 entropy 曲线）为图 3、图 4 与图 5。

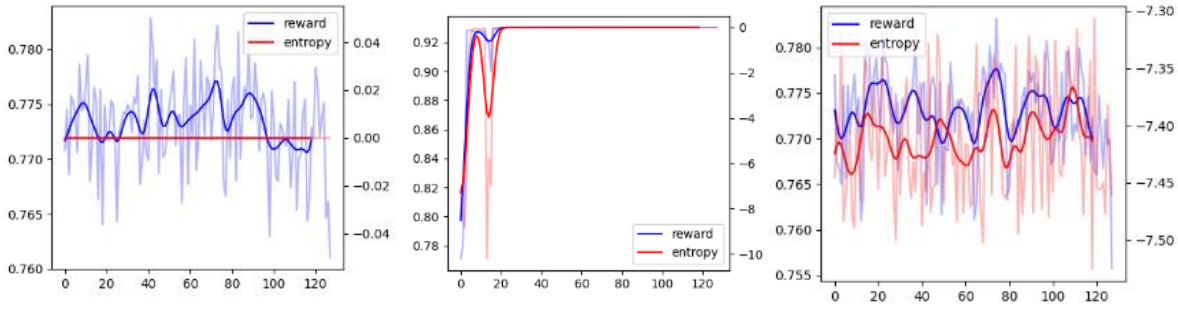

Figure 4 任务二设置 linear 的 DQN、Reinforcement 和 AC 模型的 reward、entropy 曲线

如图 4 所示，采用了 linear 奖励值的针对最大化请求者利益的任务，在训练数据过程中，可以看到 Value-Based 的 DQN 方法的 reward 值在反复震荡，均值在 0.773 左右，其 entropy 保持为定值 0；Policy-Based 的 Reinforcement 方法的 reward、entropy 值在 30 轮左右达到了收敛的状态，且 reward 均值保持在 0.926 左右，entropy 值稳定在 0；Actor-Critic 方法的 reward 和 entropy 值均在反复震荡，其 reward 均值在 0.773 左右，entropy 均值一直为负数但有一定幅度的上升。可以看出，在该部分的实验中，Reinforcement 的收敛和 reward 都较优于其他两者。

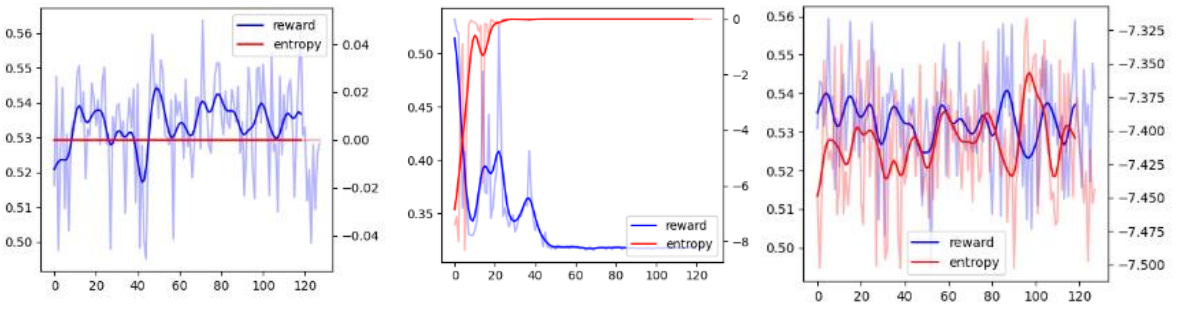

Figure 5 任务二设置 non-linear1 的 DQN、Reinforcement、AC 模型的 reward、entropy 曲线

如图 5 所示，采用了 non-linear 1 奖励值的针对最大化请求者利益的任务，在训练数据过程中，可以看到 Value-Based 的 DQN 方法的 reward 值在反复震荡，但有一定幅度的上升和收敛趋势，均值在 0.537 左右，其 entropy 保持为定

值 0；Policy-Based 的 Reinforcement 方法的 reward、entropy 值在 50 轮左右达到了收敛的状态，但 reward 相对表现较差，均值在 0.35 以下，entropy 值收敛后稳定在 0；Actor-Critic 方法的 reward 和 entropy 值均在反复震荡，且 entropy 虽有一定幅度的上升但震荡幅度有变大的趋势，而 reward 则有较微幅度的下降，其均值在 0.535 左右。可以看出在该部分的实验中，DQN 和 Actor-Critic 的 reward 较接近且优于 Reinforcement，总体而言 DQN 表现较优。

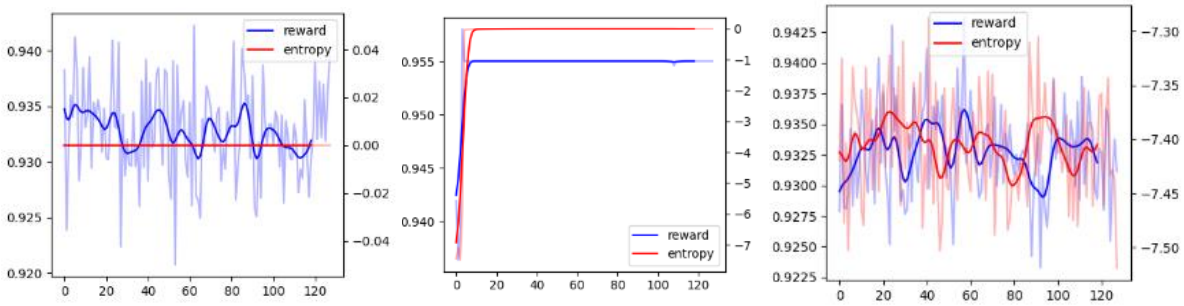

Figure 6 任务二设置 non-linear2 的 DQN、Reinforcement、AC 模型的 reward、entropy 曲线

如图 6 所示，采用了 non-linear 2 奖励值的针对最大化参与者的利益的任务，在训练数据过程中，可以看到 Value-Based 的 DQN 方法的 reward 值在反复震荡，均值在 0.932 左右，而其 entropy 保持为定值 0；Policy-Based 的 Reinforcement 方法的 reward、entropy 值收敛较快，且 reward 均值保持在 0.955 左右，entropy 值稳定在 0；Actor-Critic 方法的 reward 和 entropy 值均在反复震荡，其 reward 均值在 0.933 左右，entropy 均值一直为负数。可以看出在该部分的实验中，Reinforcement 的收敛和 reward 均优于 DQN 和 Actor-Critic，表现较优。

综合三种类型的奖励值设定来看，linear 和 non-linear 2 的实验结果相对比较理想，其中 non-linear 2 的表现会优于 linear。从实验结果上来看，non-liner 1 显然没有 linear 和 non-linear 2 的表现优秀，差距相当大，在实际应用中可以被排除。

2. 实验结果评估

我们使用测试数据集，根据四种奖励值的设置，分别在 Value-Based 的 DQN、Policy-Based 的 Reinforcement、Actor-Critic 三种模型上进行训练后，在测试训练集上进行评估，实验结果如表 6 所示。

观察表格我们可以发现，在任务一上，三种算法模型中，Reinforcement 的表现最好，reward 达到了 0.813594，而 DQN 和 Actor-Critic 均保持在 0.68 左

右，表现相近。这与前面模型训练过程中基本一致，即 Reinforcement 表现优于 DQN 与 Actor-Critic。

在任务二上，三种奖励值的设置对应有不同的实验结果，其中 non-linear 2 的表现最好，linear 的表现较优，non-linear 1 的表现最差，non-linear 1 的最高值甚至远低于另外两者的最低值。这与前面模型训练过程中大体一致，即奖励值 non-linear 2 的设置表现最优。

在任务二上，三种算法模型中，Reinforcement 的表现最好，在 linear 和 non-linear 2 两种奖励值下 reward 均达到了 0.935 以上，DQN 和 Actor-Critic 的表现同任务一相同为比较相近的情况，在 non-linear 2 中为 0.93 左右，linear 中为 0.77 左右。

综合以上结果与分析，在模型的选择上 Reinforcement 表现最优，在 request 的奖励设置上 non-linear 2 的表现最优。

Table 6 DQN、Reinforcement、Actor-Critic 模型对应验证集 reward

算法	worker	requester(linear)	requester (non-linear1)	requester （non-linear2）
	$\frac{n_c}{N}$	$\alpha\frac{n_c}{N}+(1-\alpha)q$	$\frac{n_c}{N}q$	$1-\left(1-\frac{n_c}{N}\right)(1-q)$
Value （DQN）	0.686530	0.778994	0.550110	0.933731
Policy (Reinforce-ment)	0.813594	0.935251	0.319876	0.961575
Actor-Critic	0.685567	0.775289	0.551093	0.937683

五、 实验总结

本实验旨在用强化学习的方式尝试探索众包系统中的任务分配问题，我们分别从参与者与请求者双方的角度进行设计，分别考虑最大化参与者以及最大化请求者的利益。为了达成这两项目标，我们采用了基于强化学习的三种方法：Value-Based（DQN）算法、Policy-Based（Reinforcement）算法和 Actor-critic 算法。从结果上看，不同的算法和不同的奖励策略得到的结果存在一定的区别，但多数能取得比较有效的结果，说明了模型的有效性。

在我们的实验过程中，我们团队成员通过 GitHub 进行协作，项目以 AGPL-v3.0 协议开源，项目地址为 https://github.com/Viola-Siemens/RL-Crowdsourcing-Recommendation。

事实上，对于该实验来说，我们的实验设计还存在许多可以优化的地方，在数据处理、行为状态奖励的设计、算法模型上都存在很大的优化空间。

通过本次实验，尝试了强化学习的实际应用的我们对其有了更深的了解，针对参与者与请求者两方进行状态、行为、奖励的设计，我们体会到了这些重要概念的实际意义，通过对三种不同算法进行的尝试与探索，也让我们体会到不同算法在应用上的区别，启示我们根据场景选择合适的算法、调合适的参数以更好地达到我们的目标。相信在未来不断的突破和探索下，我们能进一步的掌握好如何应用强化学习这项技术，从而更好地将它应用到实践中去。

项目分工：

任务	**算法**	**负责人**
API 设计与环境搭建	-	刘冬煜
数据读入、处理与分析	-	赵倩仪
Value-based	DQN	李新雨
Policy-based	Reinforcement	宋晏如
Actor-critic	Actor Crtic	毛科捷
报告撰写	-	李康恬

使用方法：

```
python main.py [-h] [--algo {actor-critic,policy-based,value-based}] [--epochs EPOCHS] [--reward_type {w,r,rn1,rn2}] [--optimizer {adam,rmsprop,adagrad,sgd}] [--lr LR]
```

参数解释：

-h, --help	显示帮助信息并退出程序
--algo	强化学习算法
--epochs	训练轮数
--reward_type	奖励值类型，与任务有关
--optimizer	优化器类型
--lr	学习率

例：

```
python main.py --algo policy-based --epochs 512 --reward_type w --optimizer rmsprop --lr 0.005
```