# 基于深度强化学习算法的游戏账号估值平台

## 1 需求分析

随着互联网技术和计算机技术的高速发展，市面上各式各样的游戏层出不穷。当代人对于娱乐活动和社交活动的需求日益增加，游戏行业拥有广阔的市场。玩家要想获得好的游戏体验，就必须付诸时间和精力，在快节奏的现代生活要付出大量的时间和精力用于账号升级显然不太现实，游戏账号交易拥有广阔的市场空间。然而游戏种类层出不穷，游戏账号的估值仍为一个空白的领域仍需开拓。

本文档的主要目的是对基于第五人格游戏的网页进行全面的需求分析，以明确网页所需的功能、性能和约束条件等方面的要求。具体而言，本文档将详细阐述对用户输入数据的处理方式、估值算法的设计原理和实现方法、以及用户界面的设计和实现等方面的需求。通过对这些方面的需求进行分析和规划，我们将确保所开发的网页能够满足用户需求、具有良好的用户友好性和数据安全性，并且能够稳定、高效地运行。

### 1.1 功能需求

| 角色 | 功能 |
| --- | --- |
| 用户（user） | 该平台的主要功能包括用户输入游戏中的道具数量以及等级等参数，由后台程序进行计算 |

a. 用户输入：用户应该能够输入他在游戏中的道具数量和各方面参数，包括英灵、皮肤、装备等方面的数据。

b. 数据处理：系统应该能够对用户输入的数据进行处理和计算，以获得账号的估值，包括对各种道具和属性的加权计算、市场行情的参考等方面的算法设计。

c. 估值展示：系统应该能够向用户展示他的账号估值，包括以文字或图形方式呈现账号的价值、市场情况和推荐售价等信息。

### 1.2 用例展示

| 用例名称 | 账号估值 |
| --- | --- |
| 实现名称 | account_evaluation |
| 用例描述 | 用户通过这个用例向系统请求账号估值 |
| 参与者 | 用户 |
| 前置条件 | 输入页面中的参数值 |
| 后置条件 | 输出对应的结果 |
| 主事件流 | 用户根据实际情况输入账号的信息<br>用户点击计算按钮<br>后台检查输入的值，并进行计算<br>如果输入值正确，输出最终结果 |
| 备用事件流 | 输入值的类型错误，网页提示用户重新输入 |
| 业务规则 | 用户可以通过输入参数值得到最终结果 |
| 涉及实体 | User |

### 1.3 非功能需求

a. 用户友好性：系统应该具有良好的用户友好性，包括简洁明了的用户界面、易于操作的输入方式、以及准确和可靠的估值结果等方面的要求。

b. 数据安全性：系统应该具有良好的数据安全性，包括用户数据的保密性、完整性和可靠性等方面的保障。

### 1.4 用户需求

a. 用户群体：本系统的用户主要是喜欢玩第五人格游戏的玩家，他们希望通过估值来了解自己账号的市场价值和竞争力。

b. 使用场景：用户可能在游戏中的某个时刻使用本系统，例如在想要出售自己账号或者是要更好地了解自己账号价值的情况下。

### 1.5 系统需求

技术平台：本系统需要基于 Web 技术实现，包括 HTML、CSS、JavaScript 等方面的技术支持。

## 2 基于 flask 和 vue 的系统设计

### 2.1 基类设计

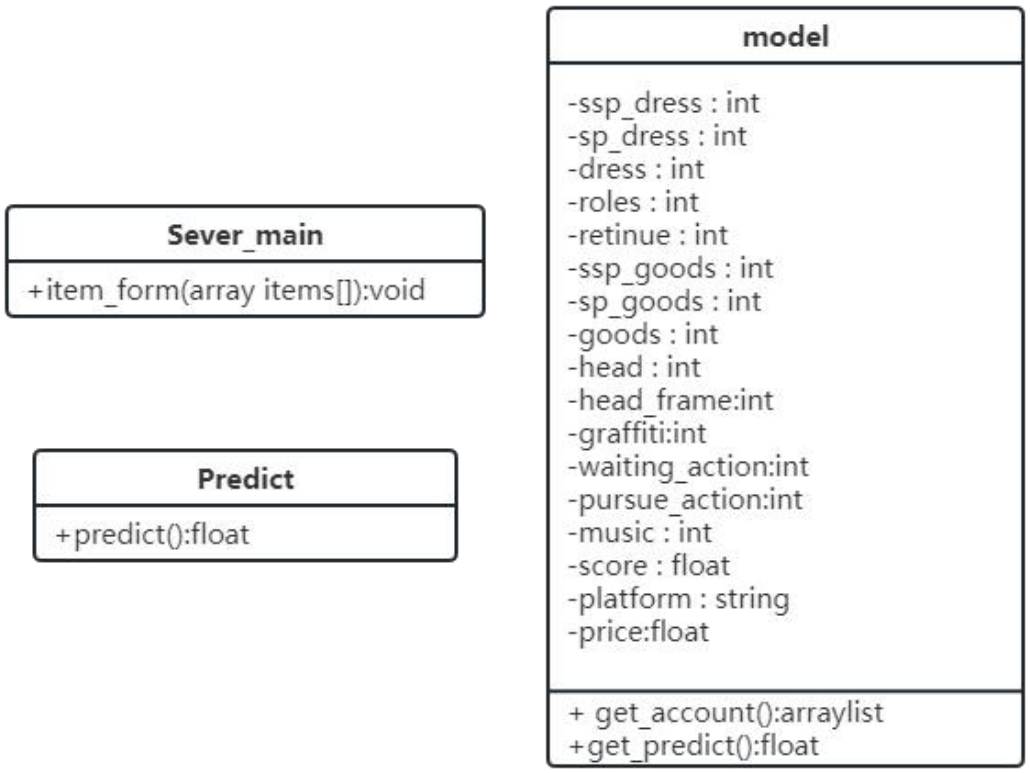


**Fig 2.1 基类设计**

## 2.2 页面 ui 设计

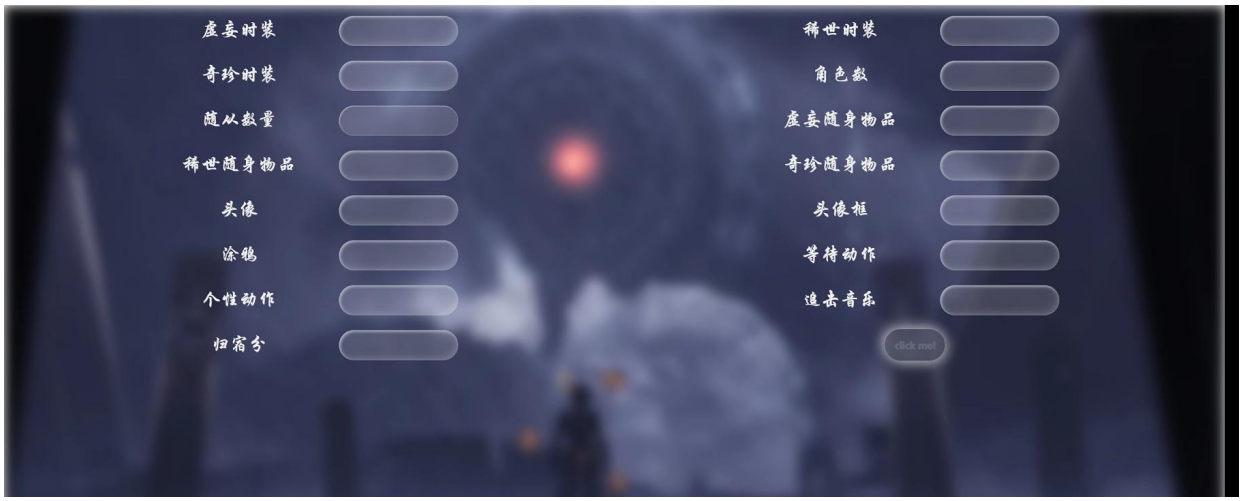


**Fig 2.2 UI 设计**

## 2.3 业务流程设计

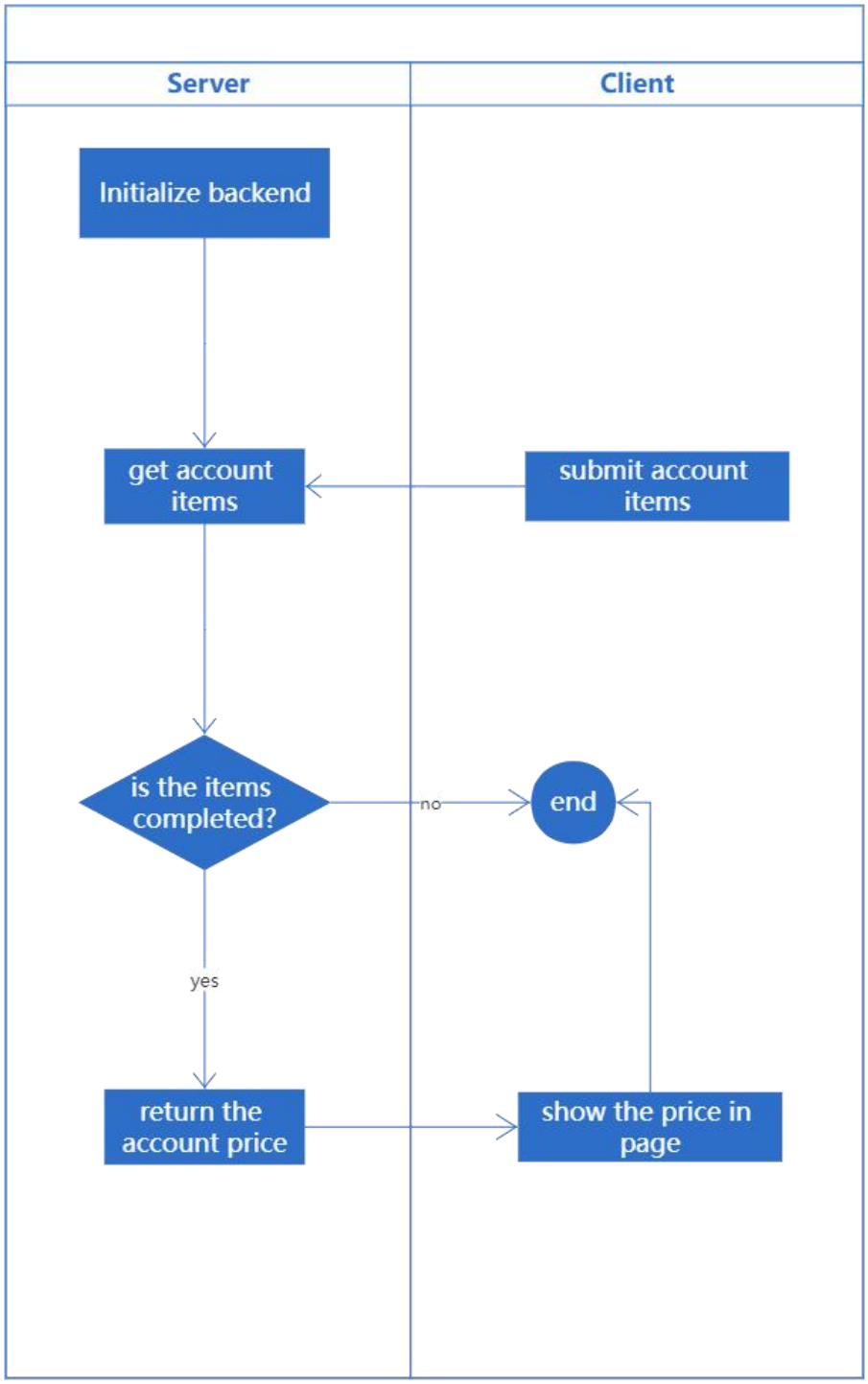


Fig 2.3 业务流程设计

# 3 基于深度强化学习的游戏帐户评估算法设计

## 3.1 综述

在这一部分中，详细介绍了基于 DDPG 的游戏账户评估算法。数据集中包含了交易猫、藏宝阁在 2023 年 1 月至 3 月中 800 个账户的待交易的账户信息。算法将数据的特征值生成一组状态（State）输入神经网络得出权重 γ 作为输出动作，通过设计的奖励函数得到该组数据的奖励值（Reward），再将这组状态，权重，奖励值存入经验池（Buffer）中，用于神经网络的更新。

## 3.2 状态空间

状态空间包括：{虚妄时装，稀世时装，奇珍时装，角色数，随从数量，虚妄随身物品，稀世随身物品，奇珍随身物品，头像，头像框，涂鸦，等待动作，个性动作，追击音乐，归宿分，交易平台，价格}(ssp_dress,sp_dress,dress,roles,retinue,ssp_goods,sp_goods,goods,head,head_frame,graffiti,waiting_actoion,music,socre,platform,price)17 个参数组成状态空间。

**3.3 动作空间**：输出的预测价格( γ )

**3.4 奖励函数**：奖励函数的结果由单步网络输出动作与专家指导的偏差大小决定，单步最大奖励值为 0.1，偏差越大，奖励值越小。奖励函数为：

$$r = 0.1 - |\gamma - \beta|$$

其中，γ 为输出估价的归一化值，β 为该项目的专家估价的归一化值。

## 3.5 模拟专家经验描述

我们以线性初等函数模拟游戏账号专家对于账号评估后给出的价格：

```
P=state[0]*0.15+state[1]*0.08+state[2]*0.06+state[3]*0.06+state[4]
*0.05+state[5]*0.15+state[6]*0.08+state[7]*0.06+state[8]*0.05+state[9]
*0.04+state[10]*0.06+state[11]*0.05+state[12]*0.06+state[13]*0.04+
state[14]*0.08*state[16]*0.13
```

## 3.6 数据归一化描述

为了更好的模拟专家经验以及满足模型训练的要求，对于训练数据我们都预先进行了归一化。同一特征值下的数据在归一化后，变为无量纲的数据，成为标量。所采用的归一化公式为：$x_{\text{归一化}} = \frac{x_{\text{原始}} - x_{\min}}{x_{\max} - x_{\min}}$

## 3.7 神经网络描述

a）网络更新流程：DDPG 算法输出的是一个动作并且，我们可以把 DDPG 中的 Actor 网络看作一块“黑布”，当我们输入一组状态（state）的时候，这块“黑布”就会被沿着 state 的位置剪开，沿着“黑布”的边缘，就是在这个状态下，不同动作对应的 Q 值，而 Actor 的任务就是通过梯度上升，不断寻找这个最大值。Critic 网络的作用是用来预估 Q 值，它的输入有两个：动作（Action）和状态（State），需要一起输入进 Critic 网络中，其用到的 loss 值与 AC 相同，采用 TD-error，作为当前行为策略的评判依据。

b）网络结构图：

actor 网络结构图：

actor 网络采用 17-30-30-1 的全网络连接结构，共三层。其中，输入层的节点数目为 17，隐藏层节点数目为 30，输出层节点数目为 1；输入层和隐藏层采用 relu 激活函数，输出层采用 sigmoid 激活函数，将结果压缩在[0,1]之间。

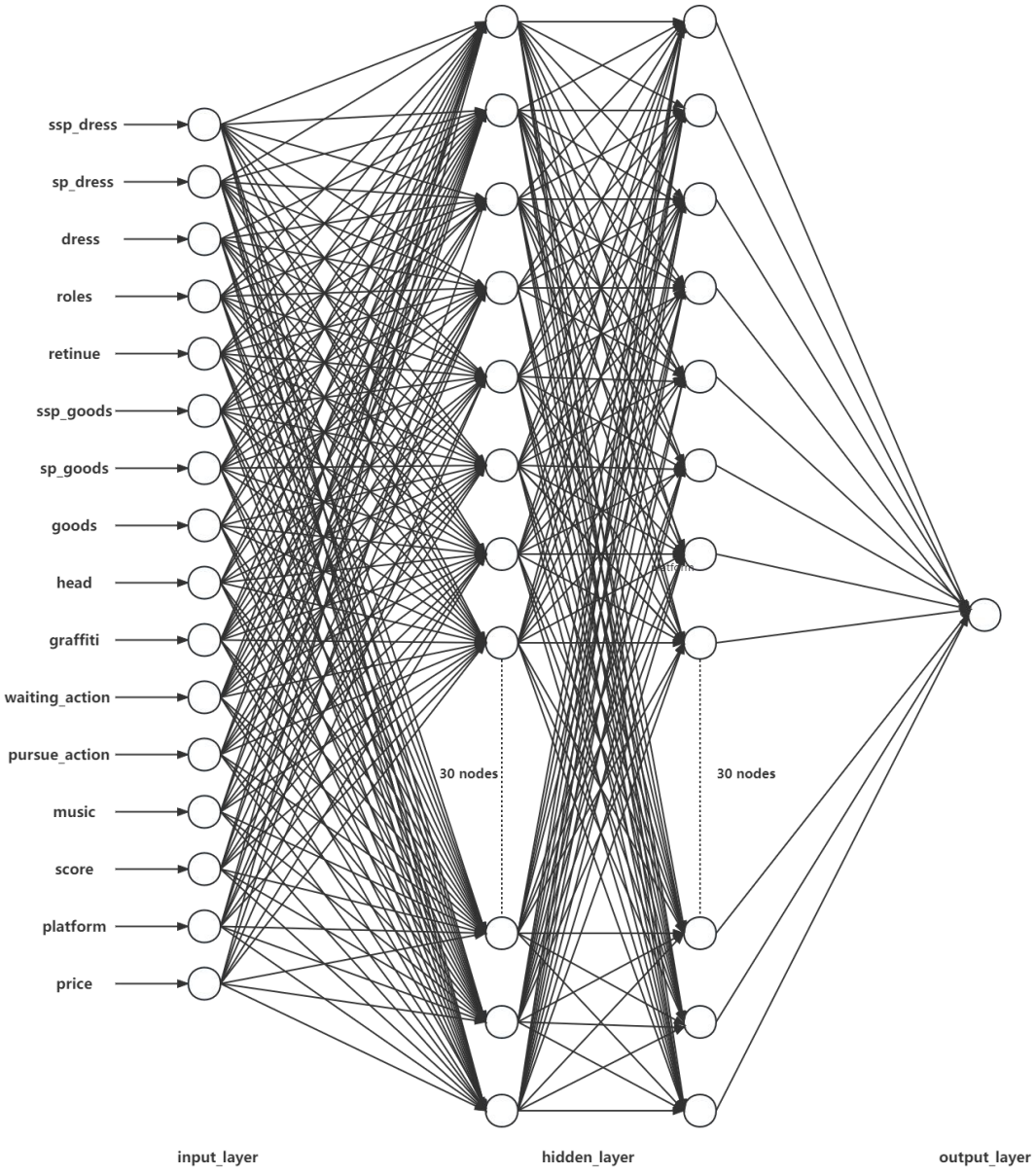


**Fig 3.1 神经网络结构**

critic 网络结构图：

critic 网络采用 7-60-60-1 的全网络连接结构。输入层大小与状态空间大小和动作空间大小之和相对应，为 18 个节点，隐藏层有 60 个节点，输出层有 1 个节点。其中，隐藏层采用 relu 激活函数。

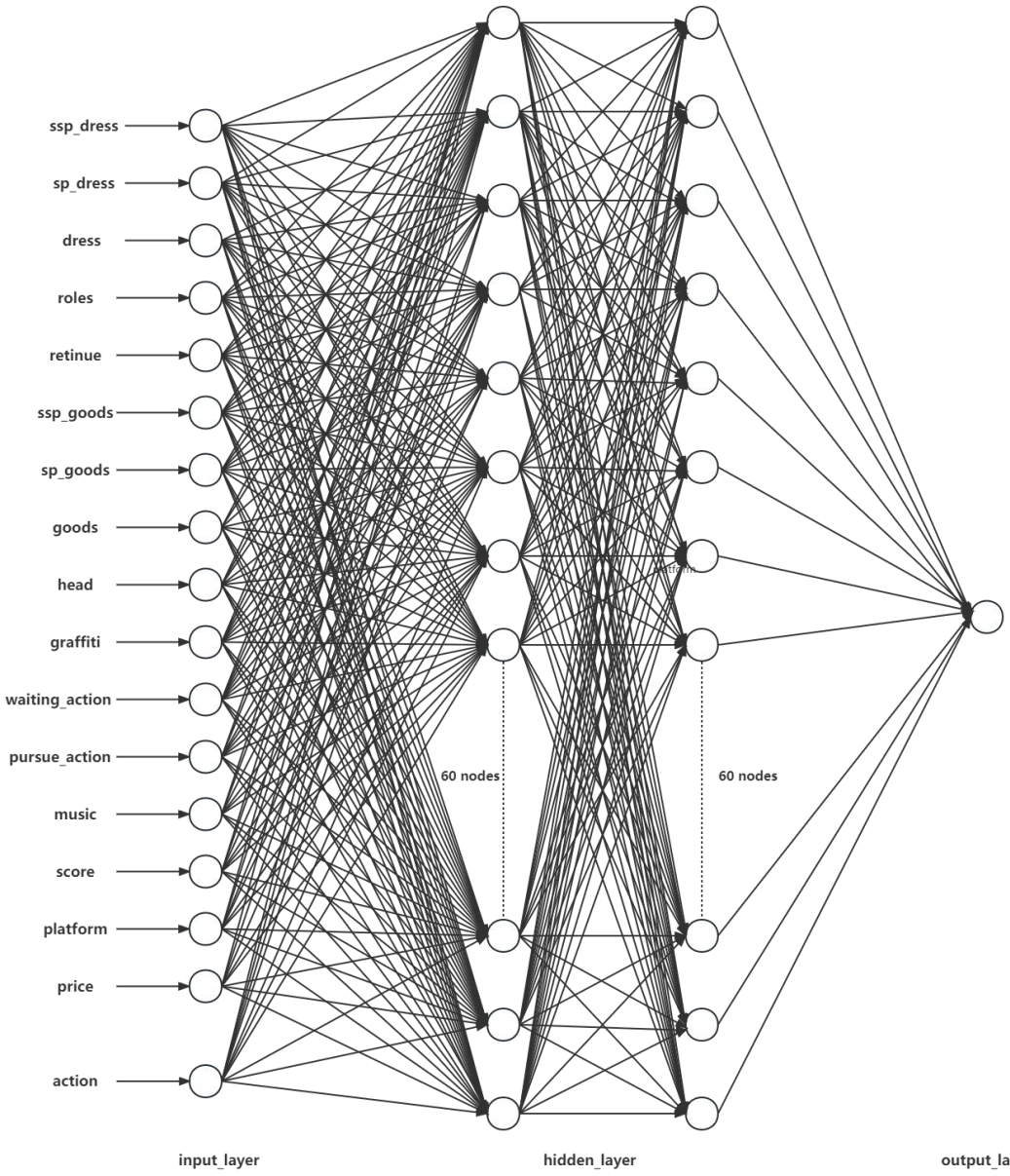


**Fig3.2 Critic 网络**

**3.8 训练算法的详细描述：**1 至 5 行描述了算法初始化参数，6 至 18 行描述了算法获取当前动作和状态的过程，19 至 29 行描述了算法每次从缓冲区选取样本，并利用梯度上升的方式更新参数的过程。

| 算法 3.1 DDPG 训练算法 |
| --- |
| 输入：Actor 网络和 Critic 网络<br>缓冲区尺寸 buffer_size<br>批尺寸 batch_size<br>每次更新中采用迭代次数 sample_update_times<br>输出：训练好的 Actor-Critic 模型 |

算法步骤：

```
1. Initialize    net_actor;//初始化  Actor  网络
2. Initialize    net_critic;//初始化  Critic  网络
3. Initialize    buffer←get_buffer(buffer_size);//初始化缓冲区  buffer
4. Initialize    Rlist ← [ ];//初始化各时间步的长期累积回报  R  列表
5. Initialize    i← 0;//初始化当前训练步数为  0
6. while receive a sample ε from the sampling node do//接收训练样本
7.          buffer.add_tail(ε);//存储样本
8,          if    buffer.size > buffer_size:
9.            i←i+ 1;
10.             Initialize    R_temp← 0;
11.           foreach ε in getReversList(buffer) do
12.             if    getIndex(buffer,ε) = buffer.size() − 1 do
13.               s ←ε.getState();
14.               V← net_critic.getValueFunc(s);
15.               R_temp←V+r*R_temp;
16.             else
17.               R_temp←getReward(ε)+r*R_temp;
18.               Initialize    j← 0;//迭代次数
19.               while    j < step    do//多次迭代
20.               foreach ε in sampleSubBuffer(buffer,batch_size) do//采样学习
21.                 s ← ε.getState();
22.                 a ← a.getActoin();
23.                 index← getIndex(buffer, ε);
24.                 R ← Rlist. getValueByIndex(index);
25.                 V← net_critic_getValueByFunc(s);
26.                 A← R-V;
27.                 actor_loss = getActorLoss(A,r(Θ),ε);//基于式（5.8）计算误差
28.                 ctitic_loss = getCriticLoss(A);//基于式（5.9）计算误差
```

```
29.                 updateActorNet(actor_loss);//更新 Actor 网络
30.                 updateCriticNet(critic_loss);//更新 Critic 网络
31.             end for
32.             j ← j+ 1;
33.         end if
34.       end
35.    end for
36   end
```

## 3.9 实验及实验分析

### 3.9.1 超参数选择与分析

为了探究合适的超参数以提升网络效率，分别选取了 Actor 网络学习率为 0.01，Critic 网络学习率为 0.001，批尺寸为 32；Actor 网络学习率为 0.01，Critic 网络学习率为0.001，批尺寸为 64；Actor 网络学习率为 0.001，Critic 网络学习率为 0.0001，批尺寸为 32 作为对照。

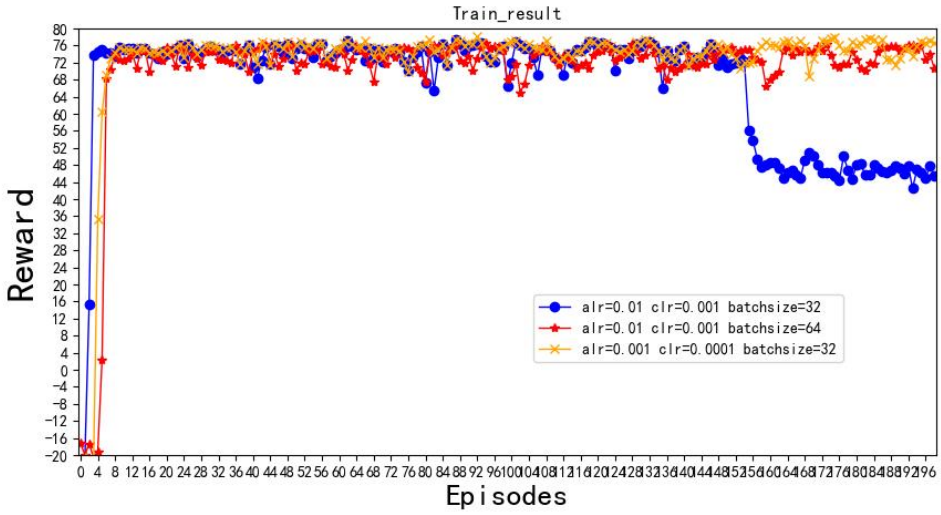


**Fig 3.3 实验及实验分析**

结果分析：在 200 周期的实验总数里，我们不难发现 DDPG 算法能够快速收敛神经网络。分析实验对比结果，当 Actor 网络学习率为 0.001，Critic 网络学习率为 0.0001，批尺寸为 32 时，网络的训练效率最高。因此，我们可以选取该组超参数下的网络作为深入测试的网络。

### 3.9.2 测试结果分析

我们选取 50 个账号数据作为测试集。图 3.4 为专家预估账号价格和神经网络输出价格的对比图像。图 3.5 为测试集上的单步输出奖励值，当 reward 越接近 0.1，专家预估与网络输出的偏差值就越小。

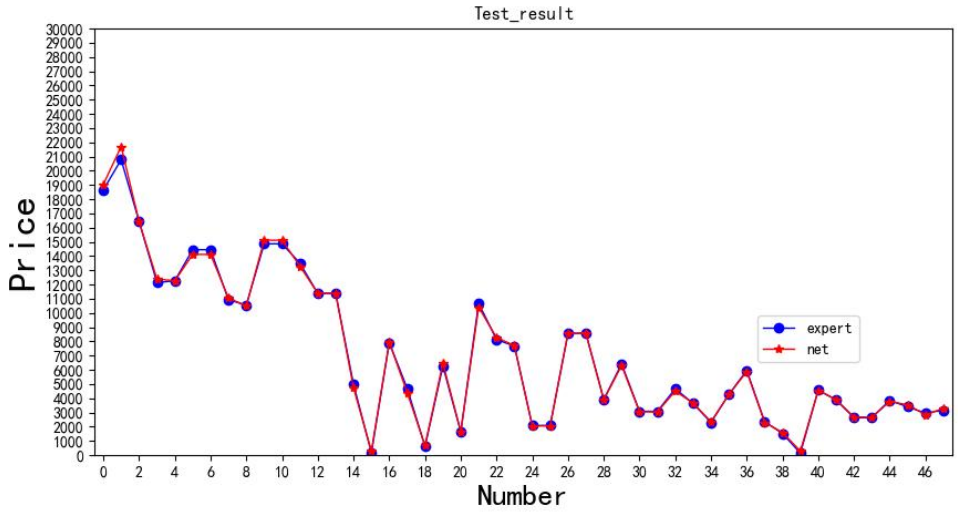


**Fig 3.4 Price 图表**

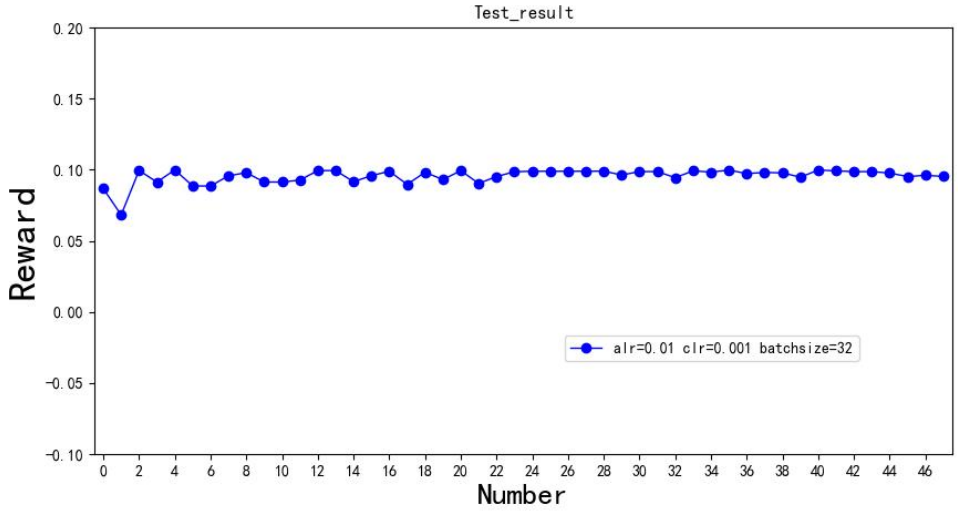


**Fig3.4 Reward 图表**

结果分析：测试结果反映出网络对于专家经验的拟合度非常高，神经网络能够准确给出该账号的预估价格。