中山大学计算机学院

人工智能

本科生实验报告

（2025 学年春季学期）

课程名称：Artificial Intelligence

教学班级		专业（方向）	
学号		姓名	

一、 实验题目

感知机算法房价预测任务

data.csv 数据集包含 4 个特征，共 10000 条数据，其中 longitude 和 latitude 表示房子经纬度，housing_age 表示房子年龄，homeowner_income 表示房主的收入（单位：十万元）。

利用感知机算法预测房价，根据对数据集进行训练，画出数据可视化图、loss 曲线图。

二、 实验内容

1. 算法原理

一、多层感知机模型（Multi-Layer Perception，MLP）： 包含三个层次，一个输入层、若干隐藏层、一个输出层，输入层和输出层神经元固定，隐藏层可自由指定的多层神经网络

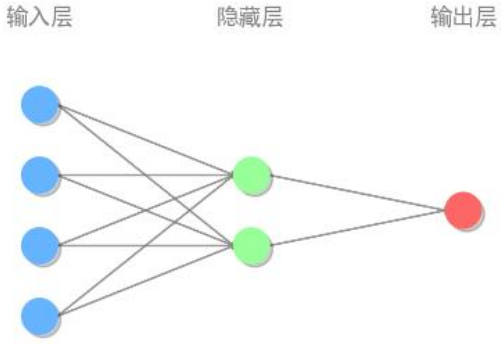


图 1 MLP 神经网络示意图

二、模型数学原理：

前向传播

$$z_1 = W_1 \cdot X + b_1 \quad (1)$$

$$A_1 = g(z_1) \quad (2)$$

$$\hat{y} = W_2 \cdot A_1 + b_2 \quad (3)$$

其中 $g(x)$ 称为激活函数，W_1 为输入到隐藏的权重矩阵，W_2 为隐藏层到输出层的权重矩阵，X 为输入的特征矩阵，b_i 为各层的偏置值，$g(x)$ 为激活函数

反向传播

各个参数的梯度：

$$\frac{\partial Loss}{\partial \hat{y}} = \frac{1}{n}(\hat{y} - y) \tag{4}$$

$$\frac{\partial \hat{y}}{\partial W_2} = A_1^T \tag{5}$$

$$\frac{\partial \hat{y}}{\partial b_2} = 1 \tag{6}$$

$$\frac{\partial \hat{y}}{\partial A_1} = W_2^T \tag{7}$$

$$\frac{\partial A_1}{\partial z_1} = g'(z_1) \tag{8}$$

$$\frac{\partial z_1}{\partial W_1} = X^T \tag{9}$$

$$\frac{\partial z_1}{\partial b_1} = 1 \tag{10}$$

其中 $Loss$ 为损失函数，X 为输入的特征矩阵，y 为输入的真实值

由链式法则可得$Loss$对各个参数的梯度

$$\frac{\partial Loss}{\partial W_2} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W_2} = \frac{1}{n}(\hat{y} - y) \cdot A_1^T \tag{11}$$

$$\frac{\partial Loss}{\partial b_2} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b_2} = \sum \frac{1}{n}(\hat{y} - y) \tag{12}$$

$$\frac{\partial Loss}{\partial W_1} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial A_1} \cdot \frac{\partial A_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial W_1} = \frac{1}{n}(\hat{y} - y) \cdot W_2^T \cdot g'(z_1) \cdot X^T \tag{13}$$

$$\frac{\partial Loss}{\partial b_1} = \frac{\partial Loss}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial A_1} \cdot \frac{\partial A_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1} = \sum \frac{1}{n}(\hat{y} - y) \cdot W_2^T \cdot g'(z_1) \tag{14}$$

损失函数

均方误差损失函数（Mean Square Error，MSE）

$$Loss_{MSE} = \frac{1}{2n}\sum_{i=0}^{n}(y_i - \hat{y}_i) \tag{15}$$

激活函数

$Sigmoid$激活函数

$$Sigmoid(x) = \frac{1}{(1 + e^{-x})} \tag{16}$$

$ReLU$激活函数

$$ReLU(x) = max(0, x) \tag{17}$$

梯度下降法

$$\begin{aligned} W^{[l+1]} &= W^{[l]} - \eta \frac{\partial Loss}{\partial W^{[l]}} \\ b^{[l+1]} &= b^{[l]} - \eta \frac{\partial Loss}{\partial b^{[l]}} \end{aligned} \tag{18}$$

其中 $W^{[l]}$ 为第 l 轮训练的权重矩阵，$b^{[l]}$ 为第 l 轮的偏置值，η 为学习率

三、学习率衰减机制

$$\eta^{[l+1]} = \gamma \cdot \eta^{[l]} \tag{19}$$

其中γ为衰减系数，$\eta^{[l]}$为第 l 轮训练学习率

应用学习率衰减机制使得前期学习率较大，后期学习率变小，有助于加速获得最小值以及防止越过最小值，优化了学习率过大和过小的影响

四、早停机制

当损失值 $\mathcal{L}oss$ 连续 100 轮训练下降值小于我们预定的容忍值时，我们结束训练，这时我们认为模型训练已经收敛，结束训练防止过拟合

五、数据预处理

我们对数据集的 10000 条数据进行划分，**其中的 80%作为我们的训练集**，用于我们模型的训练，**剩下的 20%作为我们的测试集**，用于测试我们模型训练的效果

使用 MIN_MAX 方法对数据进行归一化，将数据都转换到[0,1]的范围里，归一化有助于统一各个特征的尺度，有助于加速模型的收敛、加强训练效果

$$x^{'} = \frac{x - min}{max - min} \tag{20}$$

其中，min、max 分别为数据中的最小值和最大值

2. 伪代码

Algorithm: Multi-Layer-Perceptron

Input: X, y // 特征矩阵以及目标值向量
η // 学习率
epochs // 训练轮数
$sigmoid(x)$ // 激活函数

Return: model // 训练好的模型

Initialize: W_1、b_1、W_2、b_2

for epoch=0 to epochs **do**

// 前向传播

$Z_1 \leftarrow W_1 \cdot X + b_1$

$A_1 \leftarrow sigmoid(z_1)$

$Z_2 \leftarrow W_2 \cdot A_1 + b_2$

Loss $\leftarrow \mathcal{L}oss_{MSE}(Z_2)$

// 反向传播

$\frac{\partial \mathcal{L}oss}{\partial W_2} \leftarrow \frac{1}{n}(Z_2 - y) \cdot A_1^T$

$\frac{\partial \mathcal{L}oss}{\partial b_2} \leftarrow \sum \frac{1}{n}(Z_2 - y)$

$\frac{\partial \mathcal{L}oss}{\partial W_1} \leftarrow \frac{1}{n}((Z_2 - y) \cdot W_2^T \cdot sigmoid^{'}(Z_1) \cdot X^T$

$\frac{\partial \mathcal{L}oss}{\partial b_1} \leftarrow \sum \frac{1}{n}(Z_2 - y) \cdot W_2^T \cdot sigmoid^{'}(Z_1)$

// 更新参数

$W_2 \leftarrow W_2 - \eta \frac{\partial \mathcal{L}oss}{\partial W_2}$

$b_2 \leftarrow b_2 - \eta \frac{\partial \mathcal{L}oss}{\partial b_2}$

$W_1 \leftarrow W_1 - \eta \frac{\partial \mathcal{L}oss}{\partial W_1}$

$b_1 \leftarrow b_1 - \eta \frac{\partial \mathcal{L}oss}{\partial b_1}$

end for

3. 关键代码展示

3.1 数据的加载和预处理

使用 np 中的 genfromtxt 函数来读取 csv 数据集，同时将数据集前四列分出来成为我们的特征矩阵，最后一列作为我们的真实值，并将数据的 80%作为我们的训练集，20%作为我们的测试集.

```
    def load_data(self,filename):
        '''加载数据集'''
        data = np.genfromtxt(filename, delimiter=',', skip_header=1)
        data_size=data.shape[0]
        # 特征矩阵
        X=data[ : , :4].T
        # 标签向量
        y=data[ : , 4].reshape(1,-1)
        # 划分数据集 训练集：数据集 = 8：2
        index=int(data_size*0.8)
        train_X=X[ : , :index]
        train_y=y[ : , :index]
        self.test_X,self.test_X_MIN,self.test_X_MAX=self.MIN_MAX(X[ : ,
index: ])
        self.test_y,self.test_y_MIN,self.test_y_MAX=self.MIN_MAX(y[ : ,
index: ])
        X,self.X_MIN,self.X_MAX=self.MIN_MAX(train_X)
        y,self.y_MIN,self.y_MAX=self.MIN_MAX(train_y)
        return X,y
```

3.2 MLP 模型初始化

输入输入层、隐藏层、输出层的维度、学习率这些参数，通过随机的方式生成初始的权重矩阵以及偏置值

```
    def __init__(self,data,input_dim,hidden_dim,output_dim,learn_rate):
        # 加载数据
        self.X,self.y=self.load_data(data)
        # MLP 实例的参数初始化
        self.input_dim=input_dim
        self.hidden_dim=hidden_dim
```

```
        self.output_dim=output_dim
        self.learn_rate=learn_rate
        # 记录训练过程的损失值
        self.predictions=[]
        self.test_losses=[]
        # 偏置值、权重矩阵的初始化
        np.random.seed(50)
        self.W1=np.random.randn(hidden_dim,input_dim)* np.sqrt(2 / input
_dim)
        self.b1=np.zeros((hidden_dim,1))* np.sqrt(2 / input_dim)
        self.W2=np.random.randn(output_dim,hidden_dim)* np.sqrt(2 / hidd
en_dim)
        self.b2=np.zeros((output_dim,1))* np.sqrt(2 / hidden_dim)
```

3.3 forward 前向传播

```
    def forward(self,X,y):
        '''前向传播'''
        # 隐藏层
        self.Z1=np.dot(self.W1,X)+self.b1
        self.A1=self.ReLU(self.Z1)
        #self.A1=self.sigmoid(self.Z1)
        # 输出层，回归问题采用线性输出，即不进行激活函数激活
        self.Z2=np.dot(self.W2,self.A1)+self.b2
        return self.Z2
```

3.4 backward 反向传播

```
    def backward(self,X,y):
        '''反向传播'''
        # 输出层 grad
        dZ2=self.dMSE(y,self.Z2)
        dW2=np.dot(dZ2,self.A1.T)
        db2=np.sum(dZ2,axis=1,keepdims=True)
        # 隐藏层 grad
```

```
        dA1=np.dot(self.W2.T,dZ2)
        dZ1=dA1 * (self.Z1 > 0)
        #dZ1=dA1 * self.A1 * (1-self.A1)
        dW1=np.dot(dZ1,X.T)
        db1=np.sum(dZ1,axis=1,keepdims=True)
        # update 参数
        self.W1-=self.learn_rate*dW1
        self.W2-=self.learn_rate*dW2
        self.b1-=self.learn_rate*db1
        self.b2-=self.learn_rate*db2
```

3.5 fit 方法训练模型

这是 MLP 模型的核心函数，在这个函数中对训练集进行训练，并设置了**早停机制**和**学习率衰减机制**，每次训练都由一次向前传播和反向传播组成，同时每次训练结束后都计算一次训练集和测试集的损失值，用于绘制 Loss 曲线和用于反向传播

```
    def fit(self,epochs=10000,tolerant_error=1e-6):
        '''基于数据集训练模型'''
        self.losses=[]
        self.test_losses=[]
        best_loss=float('inf')
        cnt=0
        for epoch in range(epochs):
            if epoch == 0:
                self.predictions.append(self.predict(test_X,test_y))
            # 学习率衰减机制
            self.learn_rate*=0.999
            # 前向传播
            cur_output=self.forward(self.X,self.y)
            # 反向传播
            self.backward(self.X,self.y)
            # 计算损失函数
            loss=self.MSE(self.y,cur_output)
```

```
            self.losses.append(loss)
            self.predict(self.test_X,self.test_y)
            # 早停机制
            if loss < best_loss:
                if np.abs(loss-best_loss) < tolerant_error:
                    cnt+=1
                    if cnt >= 100:
                        break
                best_loss=loss
            else :
                cnt+=1
                if cnt >= 100:
                    break
            # 每隔 100 次训练输出一次损失值
            if epoch % 1 ==0:
                print(f'Epoch: {epoch}: Loss: {loss}')
        self.predictions.append(self.predict(self.test_X,self.test_y))
```

3.6 predict 方法

MLP 类中提供的使用 MLP 类中训练好的模型对输入的数据进行计算，从而计算出模型的输出值

```
    def predict(self,X,y):
        '''测试集测试'''
        # 隐藏层
        Z1=np.dot(self.W1,X)+self.b1
        A1=self.ReLU(Z1)
        #A1=self.sigmoid(Z1)
        # 输出层，回归问题采用线性输出，即不进行激活函数激活
        Z2=np.dot(self.W2,A1)+self.b2
        loss=self.MSE(self.test_y,Z2)
        self.test_losses.append(loss)
        return Z2
```

4. 创新点&优化

- **学习率衰减机制**，设置了学习率衰减系数，每次训练都都让当前学习率 η 乘以一个小于 1 的衰减系数 γ ，这可以使得学习率前期较大，后期较小，即**加速收敛的同时，又使得学习率后期不会过大，错过最小值**
- **早停机制**，设定一个极小预定的阈值，如果当前训练的 $\mathcal{L}oss$ 值的下降值小于这个阈值，我们认为这一轮训练没有效果，如果连续 100 轮训练没有效果，我们认为已经收敛，提早结束训练，这样可以**减少训练的时间成本**，同时在保证训练效果的同时**减少过拟合的风险**，提高训练的质量

三、 实验结果及分析

1. 实验结果展示示例

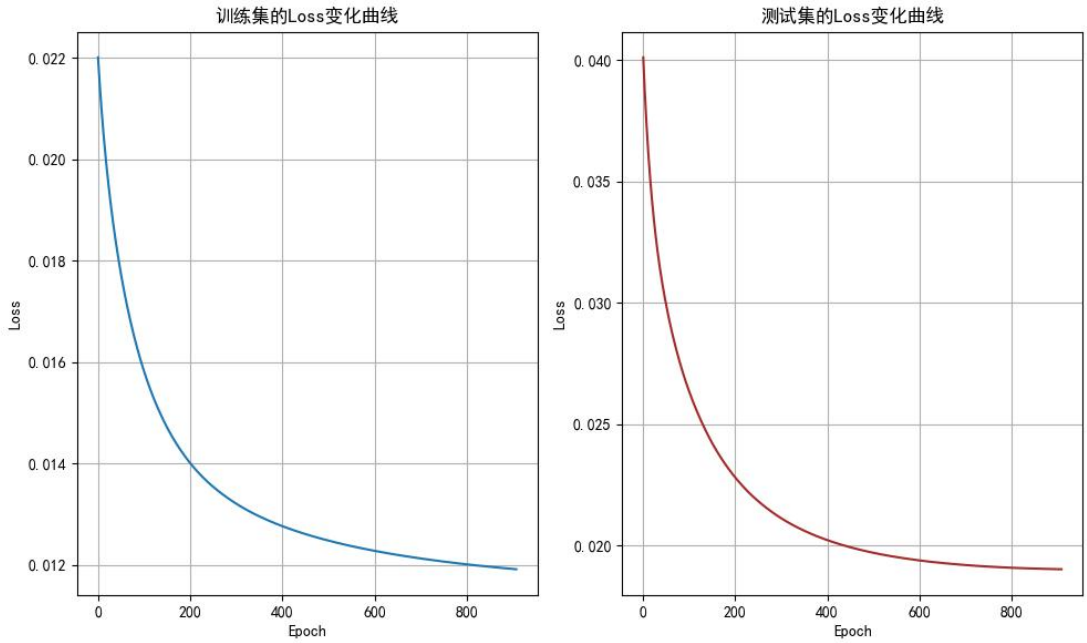
这一模块分别展示一层 MLP 和两层 MLP 的最好效果，对于不同参数对 MLP 的影响将会在第二部分介绍

1.1 一层隐藏层 MLP

采用了 $\boldsymbol{ReLU}$ **激活函数，16 个隐藏神经元，学习率为 0.1**，最后的 $\mathcal{L}oss$ 值为 0.011915247091

损失值 $\mathcal{L}oss$ 曲线

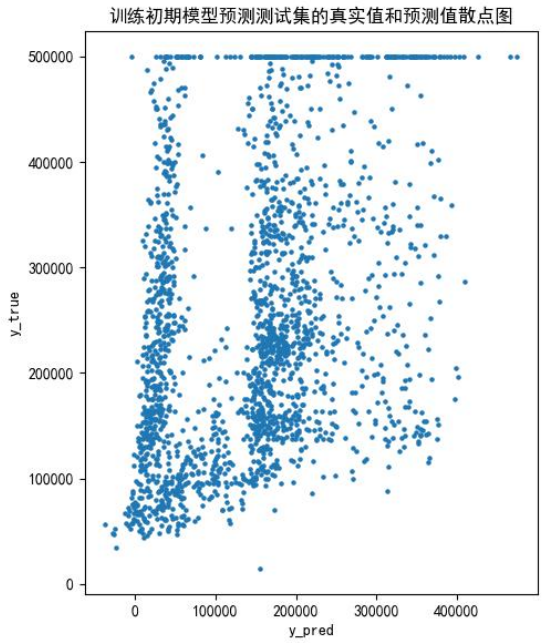
可以看到我们的 $\mathcal{L}oss$ 值在不断的下降最后收敛到一个极小的值，说明随着训练的进行，我们的模型的预测效果越来越好。

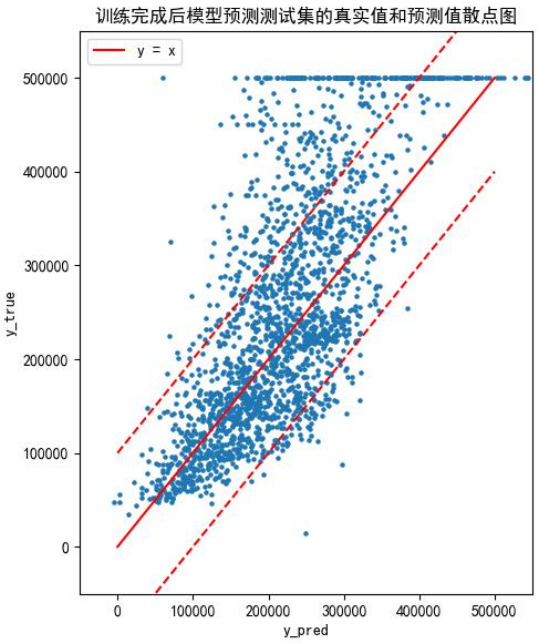


模型预测值和真实值的散点图的前后对比图

其中**红色直线表示预测值等于真实值，散点越靠近红色直线说明预测的模型的效果越好**

观察图像我们可以看到在模型训练前期散点都是无规律的散乱分布，当我们完成 MLP 的训练之后，我们的图都聚集到了红色直线的两侧，说明我们训练是有效的。

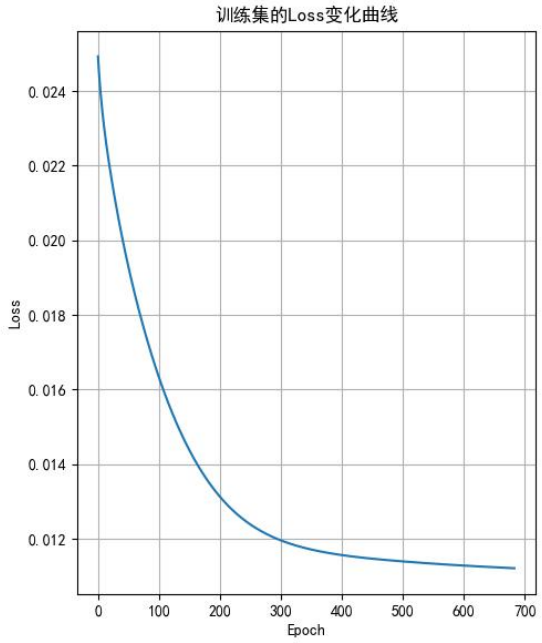


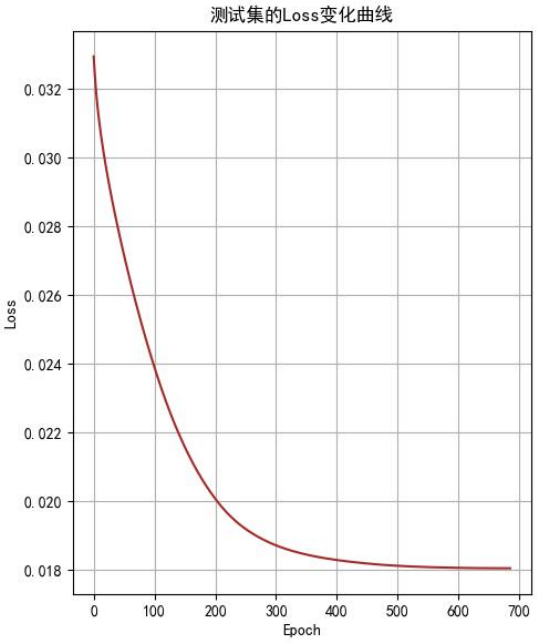


1.2 两层隐藏层 MLP

采用了 ***ReLU*** **激活函数，64×16 的隐藏神经元，学习率为 0.1**，最后的 $\mathcal{L}oss$ 值为 0.0112140

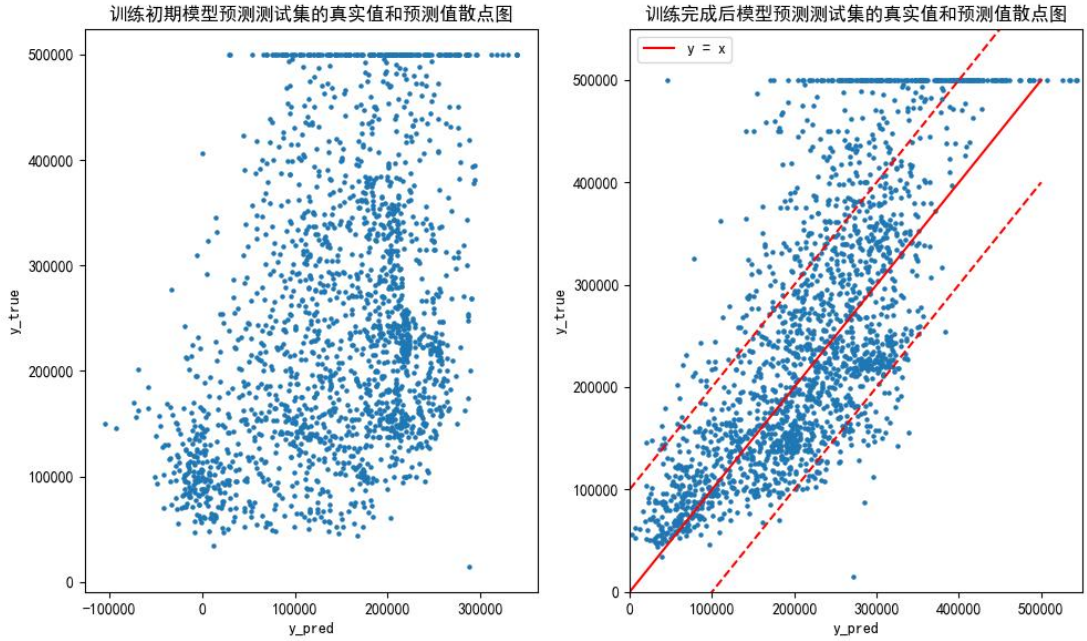
损失值 $\mathcal{L}oss$ 曲线





模型预测值和真实值的散点图的前后对比图

红色直线表示预测值等于真实值，散点越靠近红色直线说明预测的模型的效果越好



2. 评测指标展示及分析

2.1 不同参数、激活函数下的实验结果表

在这部分中，我们分别以实验中最优的结果为基准(加粗部分)，分别调整隐藏层神经元个数、激活函数、学习率来寻找这些因素对 MLP 网络训练的影响

隐藏层数	激活函数	隐藏层神经元个数	学习率	训练次数	测试集 Loss 值
1	*ReLU*	**4**	0.1	1879	0.028060635
1	*ReLU*	**64**	**0.1**	792	0.021790256
1	***ReLU***	**16**	**0.1**	**916**	**0.019004523**
1	*ReLU*	16	**0.01**	1158	0.027387542
1	*ReLU*	16	1	729	0.021361896
1	***Sigmoid***	16	0.1	1927	0.026611272
2	*ReLU*	64×16	0.1	692	0.020639404
2	*Sigmoid*	64×16	0.1	2442	0.022961511

表 1 实验结果表

通过上表，以及下方的对比图可知

- 在本次实验中 *ReLU* 激活函数的效果会优于 *Sigmoid* 激活函数，可能是因为 *ReLU* 函数在训练的时候会让一部分的神经元的输出为0，可以形成网络的稀疏性，且 *Sigmoid* 函数的训练时间较长，训练次数都会明显大于同情况的 *ReLU* 函数
- 相同条件下，适中的学习率对模型的训练效果更加好，过小的学习率可能无法达到最优解，过高的学习率有时往往会有过拟合的风险，这一点将会在后面的对比图详细说明
- 相同条件下，适当的隐藏层神经元数量对模型的训练效果更加好，数量过多，加大训练成本，同时有着过拟合的风险，数量太少，对样本的训练学习程度不够，效果不好，同样这一点将会在后面的对比图详细说明

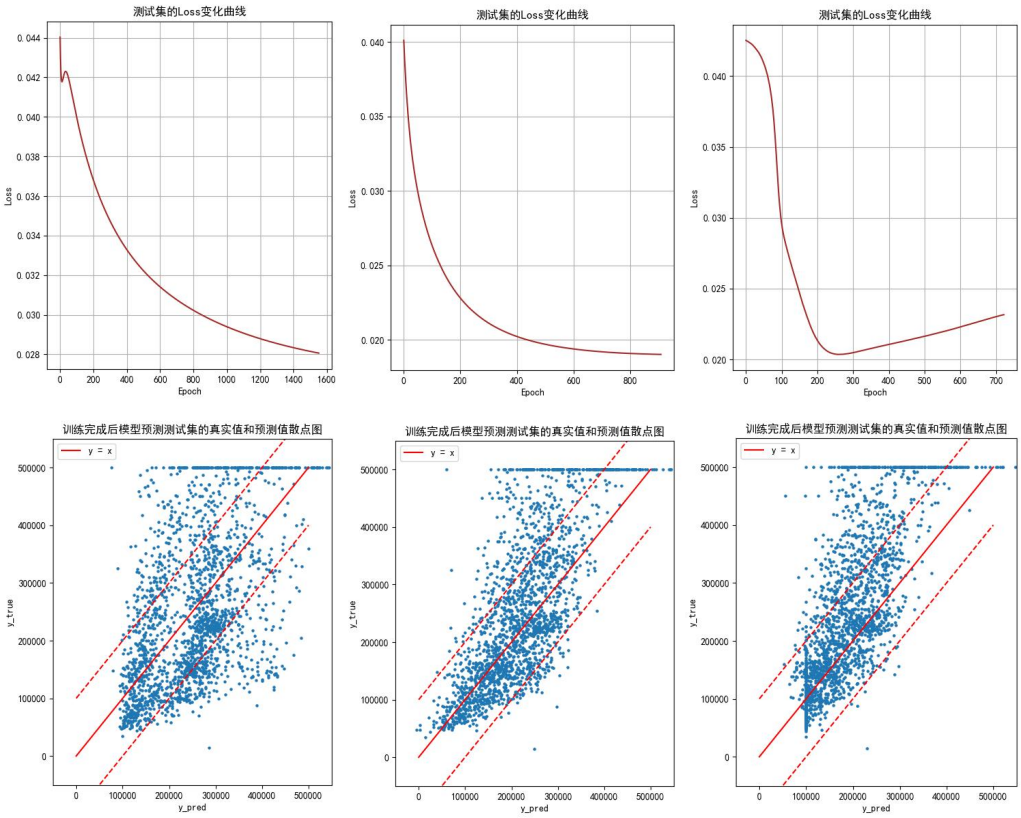
2.2 不同学习率的训练效果对比

下图从**左至右分别是学习率为 0.01、0.1、1 的情况**下测试集的 $Loss$ 曲线和训练前后的散点对比图

可以很明显的观察到学习率为 0.1 时的更加的拟合我们的红色直线

当学习率为 0.01 时模型训练的效果不佳，虽然 $Loss$ 在不断的下降，但是并**没有很接近我们的最小值**；

而当学习率为 1 时虽然 $Loss$ 下降的效果不错，但是我们可以很明显的看到后期 $Loss$ 在上升，这有可能是因为我们**越过的最小值点，导致我们的效果不佳**



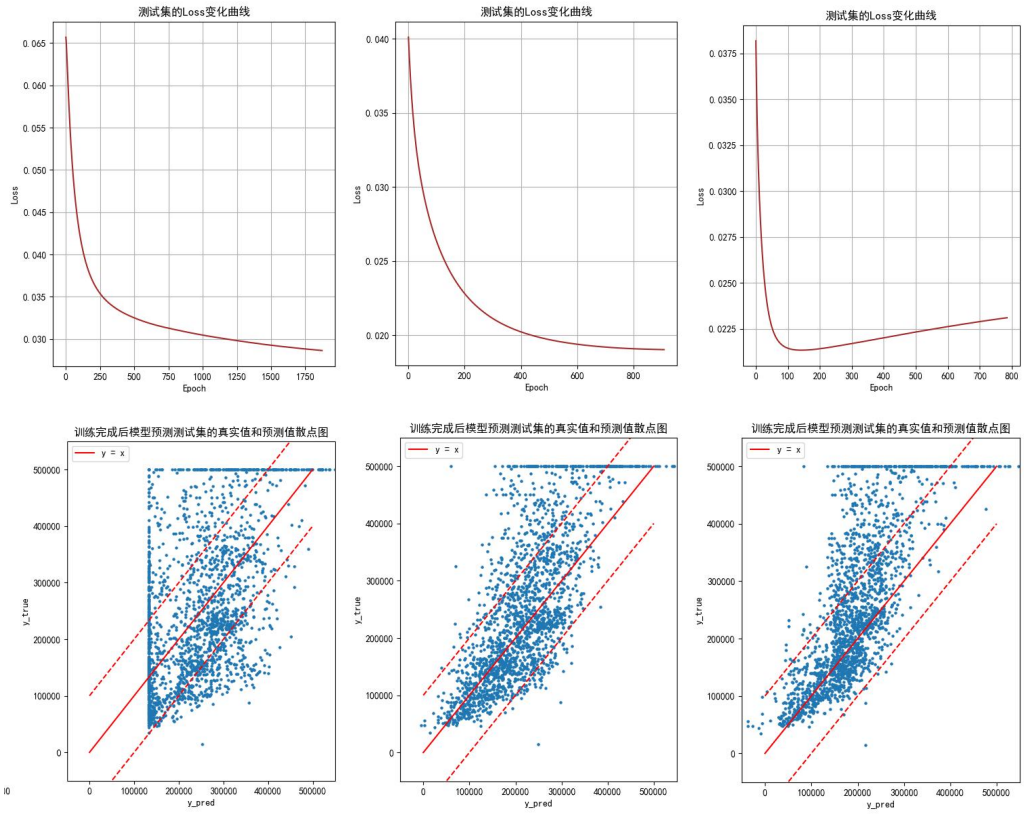
2.3 不同隐藏神经元数量的训练效果对比

下图从**左至右分别是隐藏层神经元为 4、16、64 的情况**下测试集的 $\mathcal{L}oss$ 曲线和训练前后的散点对比图

可以观察到隐藏层神经元为 16 时，模型的效果最佳

当隐藏层神经元**数量过少**时，**对规律的学习效果不佳，并能做出良好的预测**

当隐藏层神经元**数量过多**时，后期 $\mathcal{L}oss$ 曲线出现了增长的现象，这时因为模型的训练中**出现了过拟合的情况**，模型的训练过度的契合训练集，导致我们的**测试集对模型进行测试的时候 $\mathcal{L}oss$ 曲线在后期反而增加了**

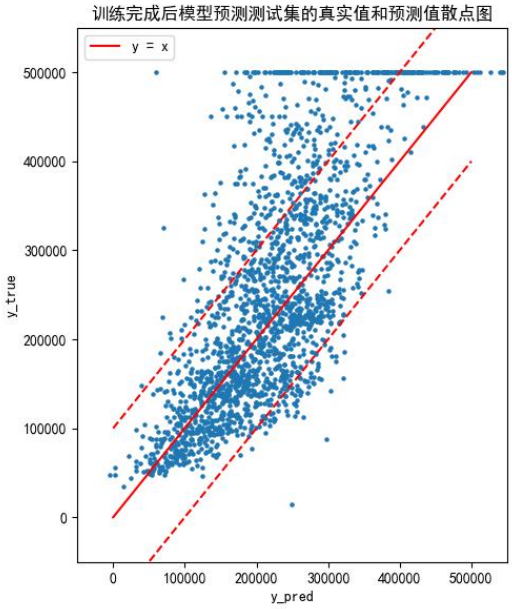


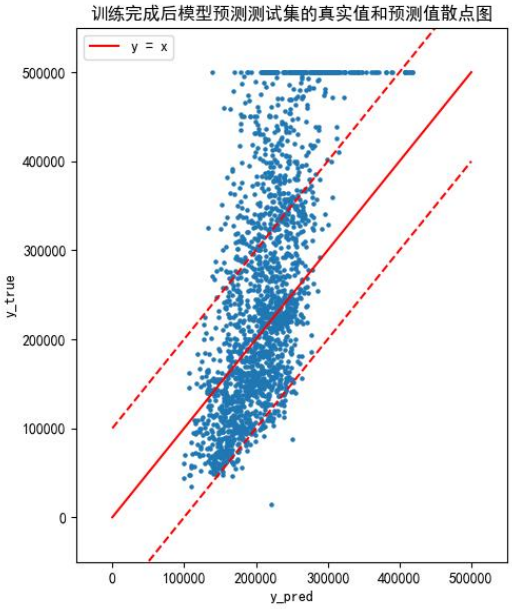
2.4 不同激活函数的训练效果对比

左边是使用 $ReLU$ 激活函数的效果，右边是使用 $Sigmoid$ 激活函数的效果

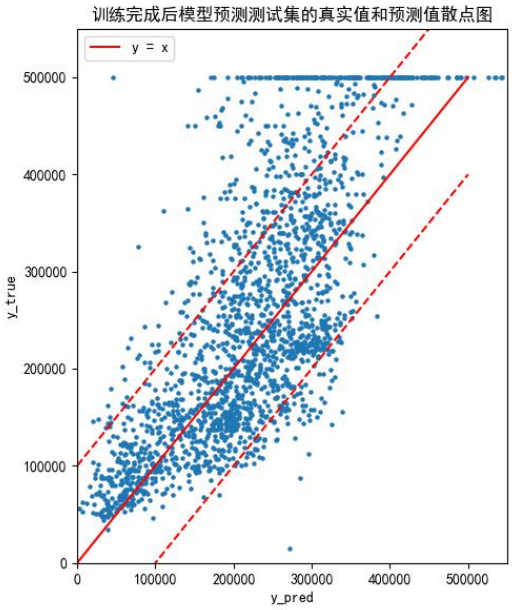
可以明显的发现，**左边的散点图相较于右边的散点图更加的均匀分布在红色直线的两侧**，这表明 $ReLU$ 函数的在本次实验中的效果是更加好的

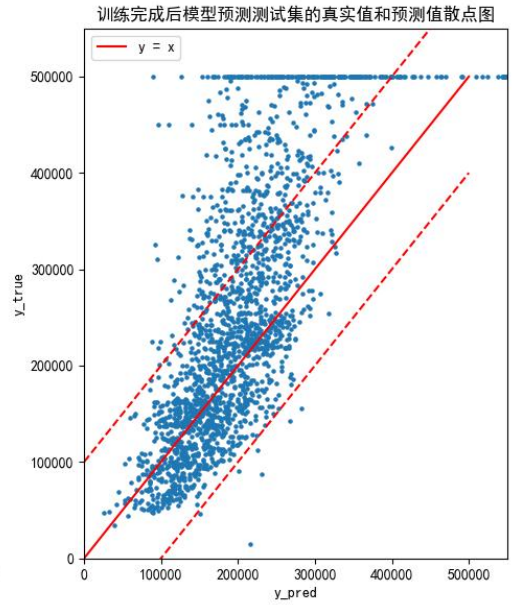
单层 MLP：





双层 MLP：

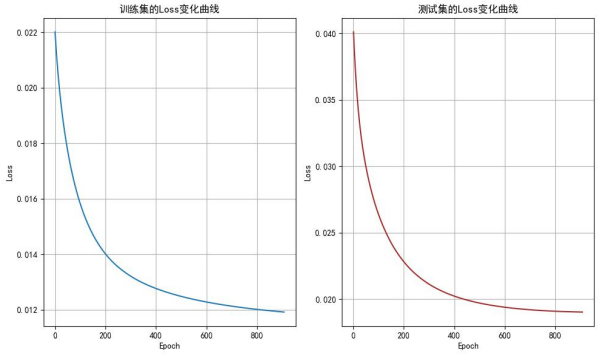




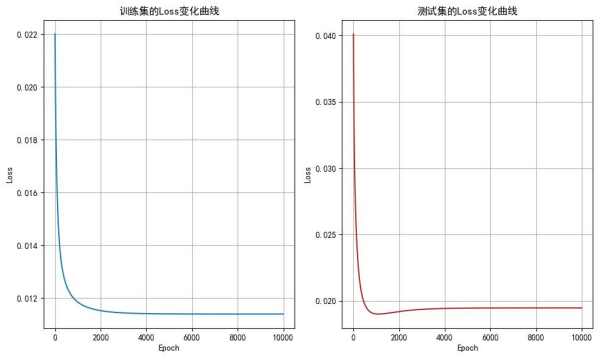
2.5 消融实验验证早停机制和学习率衰减机制

在本次实验的最后，为了验证本次实验所提出的早停止机制和学习率衰减机制的有效性，对提出的**早停机制和学习率衰减机制的方法进行消融实验**

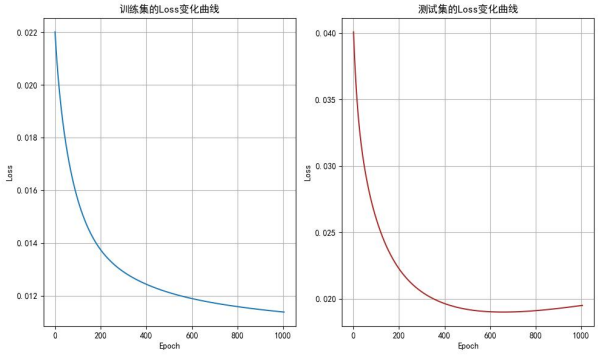
本次实验的算法：



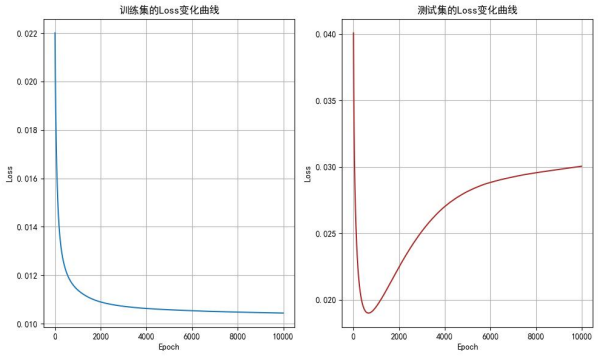
无早停止机制：



无学习率衰减机制：



无早停机制+无学习率衰减机制：



从上面的结果可以总结出：虽然四种方法在对训练集进行训练的时候 Loss 值都在下降，但是当我们使用测试集进行测试的时候，**后面三种缺少机制的方法的测试集 Loss 值相对于第一种效果都不好**，在后期都会出现上升的趋势，尤其是最后一种无早停机制和无学习率衰减机制，Loss 值上升的非常的明显，存在明显的过拟合和超过极小值的特征.

四种方法中，本次实验所使用的方法测试集曲线并没有很明显的上升趋势，说明**本次实验提出的方法在训练的时候效果最优，这证明了我们提出的两种机制的有效性**。