人工智能实验四

18340146 计算机科学与技术 宋渝杰

任务：

实现神经网络完成共享单车使用量预测任务

标签内容参考 readme，包含时间、天气等信息

必须实现三层神经网络（输入层，隐藏层，输出层）

必须在给出的优化建议中任意选择一项实现

自己划分验证集（报告里说明是怎么分的）调整参数

因此，本次实验的任务为：对数据 train.csv 分为训练集和验证集，对训练集实现神经网络

算法原理：

数据集的分类：因为这次的数据具有明显的时间集中分布特征，因此**不能采取**之前的“前 t 条数据用于训练，其余数据用于验证”，而应该采取**均匀分布**的分类方式，我采用**“每 100 条数据前 t 条用于训练，其余数据用于验证”**的方式

神经网络：一种分布式并行信息处理的算法数学模型，分为输入层、隐层、输出层。输入层有 n 个神经元，与输入的数据特征维度相同，接收数据输入并直接输出；隐层有 q（参数）个神经元，输出层有 d 个神经元，与输出的数据标签维度相同，每个神经元接收前一层的所有神经元传递过来的输入信号，这些信号通过带权重的连接进行传递，神经元接收到的总输入值再加上偏移量通过一个“激活函数”处理以产生神经元的输出，输出层的输出即为神经网络的输出。

值得提出的是，**感知机（PLA）**属于一种无隐层的神经网络

数学符号：下面对本次实验报告用到的数学符号作统一声明（三层神经网络）：

$\mathbf{x} \in R^d$：输入数据，有 d 个特征。本次实验数据中 $d = 12$

q：隐层的神经元数量

$f(x)$：激活函数（sigmoid、tanh、Relu、elu）

γ_h：隐层第 h 个神经元的偏移量（gamma）

θ_j：输出层第 j 个神经元的偏移量（theta）

v_{ih}：输入层第 i 个神经元与隐层第 h 个神经元之间的连接权

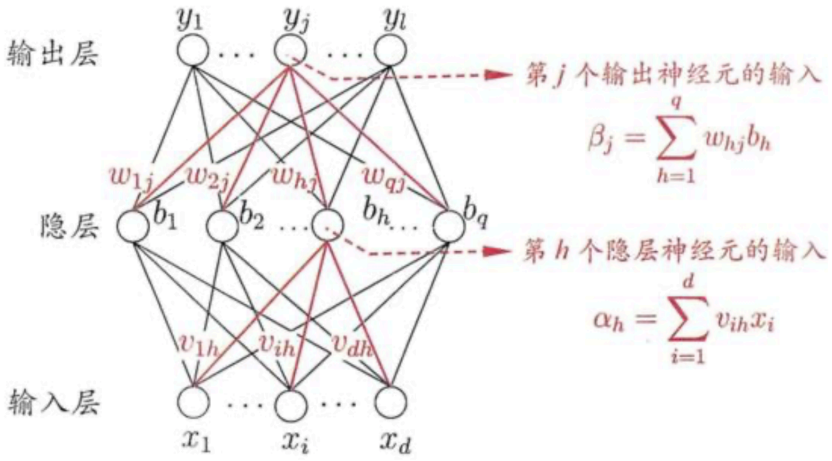
w_{hj}：隐层第 h 个神经元与输出层第 j 个神经元之间的连接权

α_h：隐层第 h 个神经元接收到的输入， $\alpha_h = \sum_{i=1}^{d} v_{ih} x_i + \gamma_h$

b_h：隐层第 h 个神经元的输出，$b_h = f(\alpha_h)$

β_j：输出层第 j 个神经元接收到的输入，$\beta_j = \sum_{h=1}^{q} w_{hj} b_h + \theta_j$

$\mathbf{y} \in R^l$：输出数据，有 l 维标签，$y_j = f(\beta_j)$。本次实验数据中 $l = 1$



反向传播算法（BPNN）：一种训练多层神经网络的学习算法，具体步骤如下：

激活函数：对神经元的输入进行处理得到输出。常见的激活函数有 sigmoid、tanh、Relu、elu

sigmoid：$f(x) = 1/(1+e^{-x}), f'(x) = f(x)(1-f(x))$

tanh：$f(x) = (e^x - e^{-x})/(e^x + e^{-x}) = 2\text{sigmoid}(2x) - 1, f'(x) = 1 - f^2(x)$

Relu：$f(x) = max(x, 0), f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \le 0 \end{cases}$

elu：$f(x) = \begin{cases} x & x > 0 \\ a(e^x - 1) & x \le 0 \end{cases}, f'(x) = \begin{cases} 1 & x > 0 \\ ae^x & x \le 0 \end{cases}$，其中 a 为参数（一般取 0.1）

在这次实验中由于是共享单车使用量预测/回归实验，因此输出层激活函数只能采用 Relu 函数（非负），隐层的激活函数可以采用上述四种中的任意一种。

损失函数：回归问题采用均方误差（MSE）：

单个数据 MSE 为 $E_k = \sum_{j=1}^{l} (\hat{y}_j^k - y_j^k)^2$

所有数据 MSE 为 $E = \frac{1}{m} \sum_{k=1}^{m} E_k$

梯度：对损失函数求偏导，可得其梯度：

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \frac{\partial \hat{y}_j^k}{\partial \beta_j} \frac{\partial \beta_j}{\partial w_{hj}} = -(y_j^k - \hat{y}_j^k) f'(\beta_j) b_h$$

$$\frac{\partial E_k}{\partial \theta_j} = -(y_j^k - \hat{y}_j^k) f'(\beta_j)$$

$$\frac{\partial E_k}{\partial v_{ih}} = \frac{\partial E_k}{\partial b_h} \frac{\partial b_h}{\partial \alpha_h} \frac{\partial \alpha_h}{\partial v_{ih}} = -\sum_{j=1}^{l} w_{hj} (y_j^k - \hat{y}_j^k) f'(\alpha_h) x_i$$

$$\frac{\partial E_k}{\partial \gamma_h} = -\sum_{j=1}^{l} w_{hj} (y_j^k - \hat{y}_j^k) f'(\alpha_h)$$

根据**梯度下降**，一次更新之后的参数为：（注意，3、4 式用到的 w 为原 w，而不是更新之后的 w，之后的公式同理）

$$w_{hj} \leftarrow w_{hj} - \eta \frac{\partial E_k}{\partial w_{hj}} = w_{hj} + \eta(y_j^k - \hat{y}_j^k) f'(\beta_j) b_h$$

$$\theta_j \leftarrow \theta_j - \eta \frac{\partial E_k}{\partial \theta_j} = \theta_j + \eta(y_j^k - \hat{y}_j^k) f'(\beta_j)$$

$$v_{ih} \leftarrow v_{ih} - \eta \frac{\partial E_k}{\partial v_{ih}} = v_{ih} + \eta \sum_{j=1}^{l} w_{hj}(y_j^k - \hat{y}_j^k) f'(\alpha_h) x_i$$

$$\gamma_h \leftarrow \gamma_h - \eta \frac{\partial E_k}{\partial \gamma_h} = \gamma_h + \eta \sum_{j=1}^{l} w_{hj}(y_j^k - \hat{y}_j^k) f'(\alpha_h)$$

由于本次实验 $l = 1$，上述公式可以化简为：

$$w_h \leftarrow w_h + \eta(y^k - \hat{y}^k) f'(\beta) b_h$$
$$\theta \leftarrow \theta + \eta(y^k - \hat{y}^k) f'(\beta)$$
$$v_{ih} \leftarrow v_{ih} + \eta w_h(y^k - \hat{y}^k) f'(\alpha_h) x_i$$
$$\gamma_h \leftarrow \gamma_h + \eta w_h(y^k - \hat{y}^k) f'(\alpha_h)$$

这就是**随机梯度下降（SGD）**的更新公式

题目（PPT）使用的是**批梯度下降（BGD）**，只需稍微修改更新公式如下：

$$w_h \leftarrow w_h + \eta \sum_{x \in X} (y_x^k - \hat{y}_x^k) f'(\beta_x) b_h / len(X)$$
$$\theta \leftarrow \theta + \eta \sum_{x \in X} (y_x^k - \hat{y}_x^k) f'(\beta_x) / len(X)$$
$$v_{ih} \leftarrow v_{ih} + \eta w_h \sum_{x \in X} (y_x^k - \hat{y}_x^k) f'(\alpha_{x,h}) x_i / len(X)$$
$$\gamma_h \leftarrow \gamma_h + \eta w_h \sum_{x \in X} (y_x^k - \hat{y}_x^k) f'(\alpha_{x,h}) / len(X)$$

其中 $len(X)$ 为数据集 X 大小

在优化方式中有一种称为 **Mini-batch 梯度下降（MGD）**，原理为：把整个数据集分为 n 部分，基于每一个小部分对参数做一次批梯度下降，因此更新公式如下：

$$w_h \leftarrow w_h + \eta \sum_{x \in miniX} (y_x^k - \hat{y}_x^k) f'(\beta_x) b_h / len(miniX)$$
$$\theta \leftarrow \theta + \eta \sum_{x \in miniX} (y_x^k - \hat{y}_x^k) f'(\beta_x) / len(miniX)$$
$$v_{ih} \leftarrow v_{ih} + \eta w_h \sum_{x \in miniX} (y_x^k - \hat{y}_x^k) f'(\alpha_{x,h}) x_i / len(miniX)$$
$$\gamma_h \leftarrow \gamma_h + \eta w_h \sum_{x \in miniX} (y_x^k - \hat{y}_x^k) f'(\alpha_{x,h}) / len(miniX)$$

其中每遍历完一个小数据集后立即做一次参数更新

不断重复更新步骤，直至达到最大迭代次数，则停止迭代，此时的各个参数即为最终的模型参数。

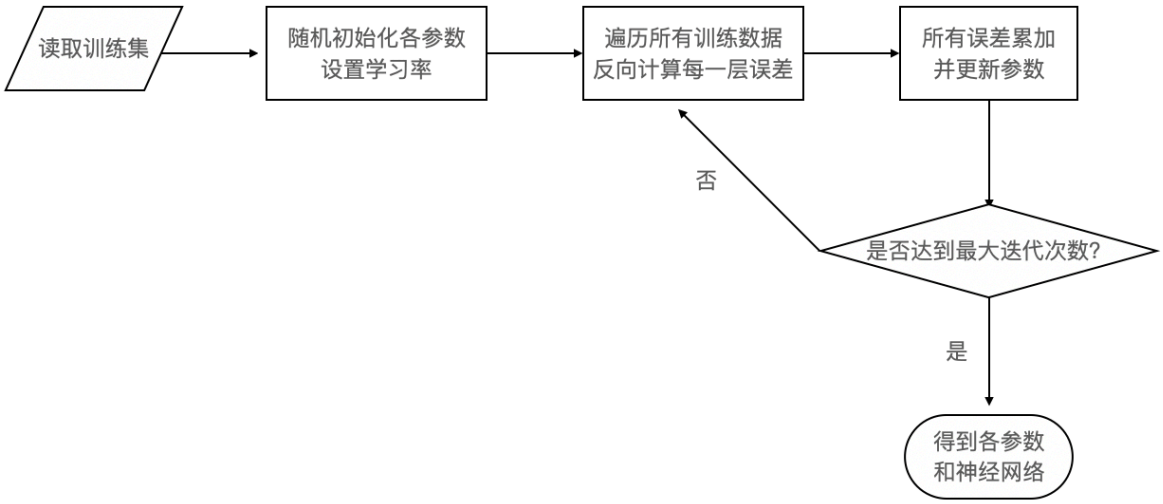
训练：总结上述训练过程，主要分为以下三个步骤：

1. 初始化参数 $w_h, \theta, v_{ih}, \gamma_h$（一般为任意随机值），设置学习率 η
2. 根据**某种梯度下降方法（SGD，BGD，MGD）**，计算梯度并更新参数
3. 重复步骤 2，当最大迭代次数时，停止迭代，得到最终参数

验证：对于验证集的每一条数据，让其正向通过神经网络得到预测的回归值，与其真实值进行对比，最终计算其损失函数（MSE）。

伪代码/流程图：

训练（以 BGD 为例）：



优化：

除了算法原理中描述的 **不同的激活函数** 和 **Mini-batch** 以外，我还采用了两种优化方式：

数据预处理：由于原数据并不是完全逻辑化的（例如 season 中 0～3 分别代表春夏秋冬，但是这种表示方法并不与共享单车使用量呈线性相关，即 season 中并不是数字越大共享单车使用量越大或越小），因此需要改变表达方式，我的预处理方式如下：

1. 统计该特征中每个不同取值对应的数据集中共享单车使用量的平均值
2. 以该平均值作为该特征中对应取值的表示值

   举个例子：下图为 season 中春夏秋冬对应的数据集中共享单车使用量的平均值（即春季的平均值为 72，其余同理），因此春季时 season 的取值修改为 72，其余同理

```
72.16307541625856
157.65592374035407
187.34375 2240
152.82895970009372
```

3. 通过上述方式预处理所有需要处理的特征（season, month, hour, holiday, weekday, workingday, weathersit）

早停（Early Stopping）：由于神经网络强大的表示性，会出现训练过程中训练误差不断减小而验证误差不断增大的情况，因此采取该优化方式进行神经网络泛化能力的提升，原理为：**当某次迭代训练之后进行一次验证，计算验证损失函数值，如果损失值比最好一次训练损失值大 0.5%，或连续十次迭代训练损失值未得到优化，则停止训练**

具体的优化结果会在下文进行展示

代码展示：

下面仅展示关键代码，全部代码请移步 lab4.py 文件

读取测试集和验证集：

```
def read(f,n,t): # 读取训练集、验证集
    data,vali = [],[]
    season = [72,157,187,152] # 数据预处理
    month = [55,74,87,131,182,199,189,186,177,166,142,118]
    hour =
[43,26,18,10,5,14,57,157,263,164,130,155,189,190,182,188,234,350,322,236,173,13
4,104,69]
    holiday = [144,125]
    weekday = [143,145,147,137,142,147,142]
    workingday = [141,144]
    weathersit = [155,135,86,36]
    f.readline() # 去除无用行
    for l in range(n):
        line = [x for x in f.readline().strip('\n').split(',')]
        line = line[2:] # 去除无用数据
        for i in range(len(line)-1): line[i] = float(line[i])
        line[0] = season[int(line[0])-1]/100 # 数据预处理
        line[2] = month[int(line[2])-1]/100
        line[3] = hour[int(line[3])]/100
        line[4] = holiday[int(line[4])]/100
        line[5] = weekday[int(line[5])]/100
        line[6] = workingday[int(line[6])]/100
        line[7] = weathersit[int(line[7])-1]/100
        line[-1] = int(line[-1])
        if l%100 < t: data.append(line) # 训练集
        else: vali.append(line) # 验证集
    return data,vali
```

随机初始化参数：

```
def init(): # 随机初始化参数
    v = [[uniform(0,1) for j in range(q)] for i in range(12)]
    gamma = [uniform(0,1) for i in range(q)]
    w = [uniform(0,1) for i in range(q)]
    theta = uniform(0,1)
    return v,gamma,w,theta
```

神经网络矩阵计算：

```
def mul1(x,v,gamma): # 输入层->隐层
    alpha = []
    for i in range(q):
        alpha.append(sum([x[j]*v[j][i] for j in range(len(x)-1)])+gamma[i])
    return alpha

def mul2(b,w,theta): # 隐层->输出层
    beta = theta
    for i in range(q): beta += w[i]*b[i]
    return beta
```

四种激活函数及其导数：

```
def sig(alpha): # f(x) = 1/(1+e^-x)
    return 1/(1+Exp(-alpha))

def dsig(alpha): # f'(x) = f(x)*(1-f(x))
    return sig(alpha)*(1-sig(alpha))

def tanh(alpha): # f(x) = (e^x-e^-x)/(e^x+e^-x) = 2sigmoid(2x)-1
    return 2*sig(2*alpha)-1

def dtanh(alpha): # f'(x) = 1-f(x)^2
    return 1-tanh(alpha)**2

def elu(alpha): # f(x) = x if x > 0 else a(e^x-1)
    return alpha if alpha > 0 else 0.1*(Exp(alpha)-1)

def delu(alpha): # f'(x) = 1 if x > 0 else a*e^x
    return 1 if alpha > 0 else 0.1*(Exp(alpha))

def Relu(beta): # f(x) = max(x,0)
    return beta if beta > 0 else 0

def dRelu(beta): # f'(x) = 1 if x > 0 else 0
    return 1 if beta > 0 else 0
```

三种梯度下降方法：

```
def BGD(data,eta,v,gamma,w,theta): # 批梯度下降
    w1 = [0 for i in range(q)]
    v1 = [[0 for i in range(q)] for j in range(12)]
    gamma1,theta1,E = [0 for i in range(q)],0,0
    for x in data:
        alpha,b = mul1(x,v,gamma),[] # 隐层输入和输出
        for i in range(q): b.append(elu(alpha[i])) # 激活函数：elu
        beta = mul2(b,w,theta) # 输出层输入
        y = round(Relu(beta)) # 激活函数：Relu，并转换成整数值
        g = dRelu(beta)*(x[-1]-y) # 输出层梯度
        e = []
        for i in range(q): e.append(delu(alpha[i])*w[i]*g) # 隐层梯度
        for i in range(q): w1[i] += g*b[i] # 累加各更新量
        theta1 += g
        for i in range(len(x)-1):
            for j in range(q):
                v1[i][j] += e[j]*x[i]
        for i in range(q): gamma1[i] += e[i]
    for i in range(q): w[i] += eta*w1[i]/len(data) # 批梯度下降
    theta += eta*theta1/len(data)
    for i in range(len(x)-1):
        for j in range(q):
            v[i][j] += eta*v1[i][j]/len(data)
    for i in range(q): gamma[i] += eta*gamma1[i]/len(data)
    return v,gamma,w,theta # 返回参数

def MGD(data,eta,v,gamma,w,theta): # Mini-batch梯度下降
    m,c = 100,0 # Mini-batch大小
    w1 = [0 for i in range(q)]
    v1 = [[0 for i in range(q)] for j in range(12)]
    gamma1,theta1,E = [0 for i in range(q)],0,0
    for x in data:
        alpha,b = mul1(x,v,gamma),[] # 隐层输入和输出
        for i in range(q): b.append(elu(alpha[i])) # 激活函数：elu
        beta = mul2(b,w,theta) # 输出层输入
        y = round(Relu(beta)) # 激活函数：Relu，并转换成整数值
        g = dRelu(beta)*(x[-1]-y) # 输出层梯度
        e = []
        for i in range(q): e.append(delu(alpha[i])*w[i]*g) # 隐层梯度
        for i in range(q): w1[i] += g*b[i] # 累加各更新量
        theta1 += g
        for i in range(len(x)-1):
            for j in range(q):
                v1[i][j] += e[j]*x[i]
        for i in range(q): gamma1[i] += e[i]
        c += 1
        if c == m: # 达到一次Mini-batch数据集大小
            for i in range(q): w[i] += eta*w1[i]/c # 梯度下降
```

```
            theta += eta*theta1/c
            for i in range(len(x)-1):
                for j in range(q):
                    v[i][j] += eta*v1[i][j]/c
            for i in range(q): gamma[i] += eta*gamma1[i]/c
            w1 = [0 for i in range(q)] # 重置各更新量为0
            v1 = [[0 for i in range(q)] for j in range(12)]
            gamma1,theta1 = [0 for i in range(q)],0
            c = 0
    if c != 0: # 对剩下的数据也进行一次梯度下降
        for i in range(q): w[i] += eta*w1[i]/c
        theta += eta*theta1/c
        for i in range(len(x)-1):
            for j in range(q):
                v[i][j] += eta*v1[i][j]/c
        for i in range(q): gamma[i] += eta*gamma1[i]/c
    return v,gamma,w,theta

def SGD(data,eta,v,gamma,w,theta): # 随机梯度下降
    E = 0
    for x in data:
        alpha,b = mul1(x,v,gamma),[] # 隐层输入和输出
        for i in range(q): b.append(elu(alpha[i])) # 激活函数：elu
        beta = mul2(b,w,theta) # 输出层输入
        y = round(Relu(beta)) # 激活函数：Relu，并转换成整数值
        g = dRelu(beta)*(x[-1]-y) # 输出层梯度
        e = []
        for i in range(q): e.append(delu(alpha[i])*w[i]*g) # 隐层梯度
        for i in range(q): w[i] += eta*g*b[i] # 梯度下降
        theta += eta*g
        for i in range(len(x)-1):
            for j in range(q):
                v[i][j] += eta*e[j]*x[i]
        for i in range(q): gamma[i] += eta*e[i]
    return v,gamma,w,theta
```

验证（求损失函数值）：

```
def validation(vali,v,w,gamma,theta): # 对验证集(也可对训练集)求损失函数
    E = 0
    for x in vali:
        alpha,b = mul1(x,v,gamma),[] # 隐层输入和输出
        for i in range(q): b.append(elu(alpha[i])) # 激活函数：elu
        beta = mul2(b,w,theta) # 输出层输入
        y = max(round(beta),0) # 考虑实际情况转负值为0
        E += (x[-1]-y)**2 # 计算均方误差(MSE)
    return E/len(vali)
```

BPNN 算法：

```
def BPNN(data,vali,eta,count,f): # 误差反向传播算法
    v,gamma,w,theta = init() # 随机初始化参数
    bestv,bestgamma,bestw,besttheta,bestindex = v[:],gamma[:],w[:],theta,0
    minEv = 1e6
    for c in range(count):
        v,gamma,w,theta = f(data,eta,v,gamma,w,theta) # 更新一次参数
        Ev = validation(vali,v,gamma,w,theta) # 计算验证集损失函数
        if c%10==9: print("time: ",c+1," Ev: ",Ev)
        if Ev <= minEv:
            bestv,bestgamma,bestw,besttheta,bestindex =
deepcopy(v),gamma[:],w[:],theta,c
            minEv = Ev
        elif Ev > 1.005*minEv or c-bestindex > 10:
            print("time: ",c+1," Ev: ",minEv)
            break # 早停算法
    return bestv,bestgamma,bestw,besttheta
```

实验结果以及分析：

由于采取了 **早停** 策略，因此迭代训练的次数已不再影响结果；在预测试中发现当隐层神经元数量 $q=5$ 时收敛比较稳定且速度较快，而当 q 继续增大时收敛结果及稳定性基本不变而时长呈线性增长，因此隐层神经元数量取 $q=5$。因此最终仅剩参数 t,η 需要调参，以及激活函数、梯度下降方案的选择

激活函数的选择：梯度下降方案先选择 BGD，对四种激活函数进行调参并计算最优损失值，结果如下：

激活函数	η	收敛次数	损失函数值
sigmoid	0.01	59	18168
tanh	0.01	52	18242
elu	**0.001**	**81**	**5733**
Relu	**0.001**	**85**	**5701**

（由于参数初始化为随机数，每运行一次代码都会有略微不同的结果）

总体来说，elu 和 Relu 效果近似，且均远远优于 sigmoid 和 tanh，debug 过程发现，sigmoid 和 tanh 函数对任意验证数据的预测回归值均相同，且回归值为接近训练集标签平均值的一个数，即**神经网络收敛到平均值**。然而这是一个极小值，并不是最小值，也不是我们想要的效果

因此本次实验采用 **elu** 作为隐层的激活函数

梯度下降方案的选择：激活函数选择 elu，对三种梯度下降方案进行调参并计算最优损失值，结果如下：

梯度下降	η	收敛次数	损失函数值
BGD	0.001	92	5638
MGD	**0.0001**	**59**	**5316**
SGD	0.000002	64	5391

其中 MGD 的 Mini-batch size = 100（实际测试中取 50～300 结果基本相同且均优于取过小或过大的值）

可以看出 MGD 的收敛速度和验证集损失函数值均优于其它梯度下降方法，因此本次实验采用 **MGD** 作为梯度下降方法

数据集占比：激活函数选择 elu，梯度下降方案选择 MGD，对应的学习率 $\eta = 0.0001$，调整参数 t，结果如下：

t	数据集占比	收敛次数	损失函数值
60	60%	58	5389
65	65%	58	5431
70	70%	59	5316
75	**75%**	**59**	**5257**
80	80%	57	5279

```
= RESTART: /Users/songyujie/Desktop/
time:  10  Ev:  7173.279069767442
time:  20  Ev:  6069.964534883721
time:  30  Ev:  5535.449418604651
time:  40  Ev:  5312.437209302326
time:  50  Ev:  5257.341860465116
time:  59  Ev:  5257.001744186046
>>>
```

因此本次试验参数 $t = 75$

数据预处理的优化：激活函数选择 elu，梯度下降方案选择 MGD，参数 $\eta = 0.0001, t = 75$，对比是否加入数据预处理的结果如下：

预处理	收敛次数	损失函数值
是	**59**	**5257**
否	57	14499

早停优化：迭代次数设为 100，对比是否加入早停算法的结果如下：

早停	最优训练次数	实际迭代次数	训练损失函数值	验证损失函数值
是	**49**	**59**	**5386**	**5257**
否	47	100	5345	5291

可以看出验证结果差不多的基础下，早停算法可以节省许多不必要的迭代时间

因此，本次试验验证**损失函数值的最终结果为 5257**（由于随机初始化参数的不同，可能结果会有略微不同，基本稳定在 5200～5300 内）

思考题：

- 尝试说明下其他激活函数的优缺点

  Sigmoid：

  优点：Sigmoid 函数的输出映射在 (0,1) 之间，单调连续，输出范围有限，优化稳定；求导容易

  缺点：幂运算，计算成本高；容易出现梯度弥散（反向传播时，很容易就会出现梯度消失的情况，从而无法完成深层网络的训练）；不是以 0 为中心，会导致收敛速度下降

  tanh：

  优点：输出以 0 为中心，导数范围变大，梯度消失问题有所缓解。

  缺点：幂运算，计算成本高；仍未解决梯度消失问题

  Relu：

  优点：梯度不饱和，收敛速度快；相对于 Sigmoid/tanh 激活函数，极大的改善了梯度消失的问题；不需要进行指数运算，因此运算速度快、复杂度低

  缺点：对参数初始化和学习率非常敏感，存在神经元死亡现象：只要输入是负值，那么输出就为 0，这可能导致后面的网络输入都是 0，造成网络大面积坏死；ReLU 的输出均值也大于 0，偏移现象和神经元死亡会共同影响网络的收敛性；

- 有什么方法可以实现传递过程中不激活所有节点

  使用 Relu 激活函数，只要输入是负值，那么输出就为 0，这可能导致后面的网络输入都是 0，即后面的网络都可能不被激活

- 梯度消失和梯度爆炸是什么？可以怎么解决?

  是什么：对激活函数进行求导，如果此部分大于 1，那么层数增多的时候，最终的求出的梯度更新将以指数形式增加，即发生**梯度爆炸**，如果此部分小于 1，那么随着层数增多，求出的梯度更新信息将会以指数形式衰减，即发生了**梯度消失**。

  解决：使用 Relu、elu 等激活函数：让激活函数的导数为 1；