中山大学数据科学与计算机学院
移动信息工程专业-人工智能
本科生实验报告

（2017-2018 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级	**m2**	专业（方向）	**软件工程（移动工程信息）**
学号	**15352446**	姓名	**钟展辉**

一、 实验题目

Back Propagation Neural Network(反向传递神经网络)

二、 实验内容

1. 算法原理

BP（Back Propagation）神经网络概述：人类神经系统的计算模型：每当接收到外部刺激时，信息会在人脑的多个神经元传递，每个神经元都对信息进行自己的加工，最后人脑接收信息响应刺激。神经网络算法用同样的方式，在输入和输出之间，加入了非常多的“节点”和许多层次，每个节点会对前一个节点传来的数据，按照自己拥有的一个权重系数进行加工，最终得到输出后计算误差，再反向传递误差信息，调整某些相关节点的权重。通过不断地用最终结果去返回调试，这个神经网络给正确结果赋予的概率会越来越高，反之给错误结果的概率会越来越低。

实验要求实现三层神经网络（输入层，隐藏层，输出层），其中输入层节点是每一个样本的所有特征的值，因此输入层节点数目是确定的，输出层节点即最后的预测结果，在本次实验中是一个连续型变量，即输出层节点只有一个。唯一节点数目不确定的就是隐藏层了，隐含层节点个数的多少对神经网络的性能是有影响的，有一个经验公式可以用以确定隐含层节点数目，hidenum=sqrt(m+n)+a，其中 hidenum 为隐含层节点数目，m 为输入层节点数目，n 为输出层节点数目，a 为 1~10 之间的调节常数。训练集样本前两个特征值意义不大因此省略，故而中 m=13，n=1，最后取 hidenum=6，其中包含一个始终值为 1 的节点。

构造神经网络模型分为两个过程：工作信号正向传递子过程以及误差信号反向传递子过程。

正向传递：

正向传递的第一个的步骤是生成隐藏层。假设输入层节点值数组为 X(Xi 表示该样本第 i 个特征的数值)，隐藏层节点数组为 H，输入层与隐藏层之间的权值数组为 W(Wij 表示输出层节点 Xi 与隐藏层节点 Hj 之间的权值系数)。则隐藏层的输入为：

$$IN=\sum_{i=0}^{m-1}W_{ij}x_i$$

其中 IN 表示第 j 个隐藏层节点的输入，由于可以把阈值像 PLA 实验那样融合进 X、 W 向量中，就能进一步简化计算，因此公式省略阈值 b。而要确定隐藏层节点的值，还需要使用激活函数（必须是可导函数），即 $h_j = f(IN)$,因为线性模型的表达能力不够，故而使用激活函数以加入非线性因素，一般使用 sigmoid 函数为激活函数，至此就确定了隐藏层节点的数值。

正向传递的第二个步骤是生成输出层节点，假设输出层节点值为 y，隐藏层与输出层之间的权值数组为 w(wi 表示第 i 个隐藏层节点与输出层节点之间的权值系数)，则输出层的输入为：In=w0*h0+w1*h1+...+wn*hn；实验要求此处的激活函数用线性函数 y=x，因此输出层节点值直接取其输入。至此完成了正向传递过程。

反向传递：

反向传递首先计算预测结果与真实标签的误差值，使用函数：$E = \frac{1}{2}(y - \hat{y})^2$

反向传递的目的就是修正权值，通过沿着相对误差平方和的最快速下降方向，连续调整网络的权值，使得误差函数值达到最小。根据梯度下降法，权值向量的修正正比于当前位置上误差函数 E 的梯度。

如果要更新隐藏层与输出层之间的权值系数向量 w，wi=wi+△wi，则△wi 为步长*误差函数 E 的梯度，即：

$$\Delta W_i = -\eta \frac{\partial E}{\partial W_i} = -\eta \frac{\partial}{\partial W_i}\left[\frac{1}{2}(y-\hat{y})^2\right]$$

$$= -\eta (y-\hat{y}) \frac{\partial}{\partial W_i}(y-\hat{y})$$

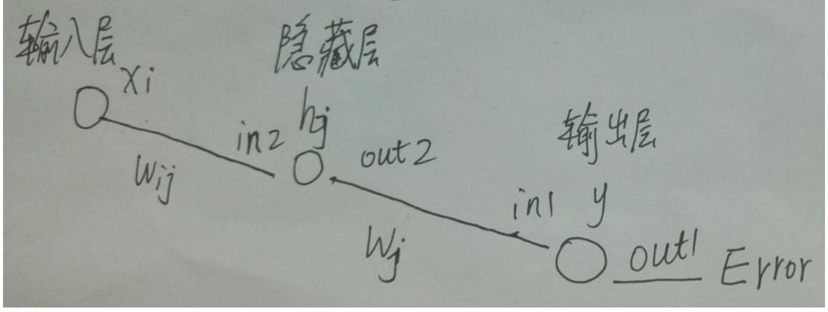
$$= \eta (y-\hat{y}) \frac{\partial}{\partial W_i}(W_0 \cdot h_0 + W_1 \cdot h_1 + \cdots W_i \cdot h_i)$$

$$= \eta (y-\hat{y}) \cdot h_i$$

因为这里采用的激活函数是 y=x 这样简单的线性函数，因此计算过程十分简单。

如果要更新输入层与隐藏层之间的权值系数向量 W，Wij=Wij+△Wij，则△Wij 为步长*误差函数 E 的梯度，即：

$$\Delta W_{ij} = \eta \frac{\partial E}{\partial W_{ij}}$$



上图中，节点的 in 是指之前节点传过来的数值，这个数值经过节点内处理后得到的结果作为节点值和 out 输出给下一个节点。

更新 Wij 的求导过程更复杂：

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial out1} \cdot \frac{\partial out1}{\partial in1} \cdot \frac{\partial in1}{\partial out2} \cdot \frac{\partial out2}{\partial in2} \cdot \frac{\partial in2}{\partial W_{ij}}$$

① $E = \frac{1}{2}(label - out1)^2 \quad \therefore \frac{\partial E}{\partial out1} = -(y - \hat{y})$ y即真实标签 $\hat{y}$为预测标签

② $out1 = in1 \quad \therefore \frac{\partial out1}{\partial in1} = 1$

③ $in1 = W_0 \cdot h_0 + W_1 \cdot h_1 + W_2 \cdot h_2 + \dots W_n \cdot h_n \quad \therefore \frac{\partial in1}{\partial out2} = W_j$

④ $out2 = sigmoid(in2) \quad \therefore \frac{\partial out2}{\partial in2} = \underbrace{sigmoid(in2)}_{即 h_j} \cdot \underbrace{[1 - sigmoid(in2)]}_{即 1-h_j}$

⑤ $h_j = W_{0j} \cdot X_0 + W_{1j} X_1 + \dots W_{nj} X_n \quad \therefore \frac{\partial in2}{\partial W_{ij}} = X_i$

因此 $\frac{\partial E}{\partial W_{ij}} = -(y - \hat{y}) \cdot W_j \cdot h_j \cdot (1 - h_j) \cdot X_i$

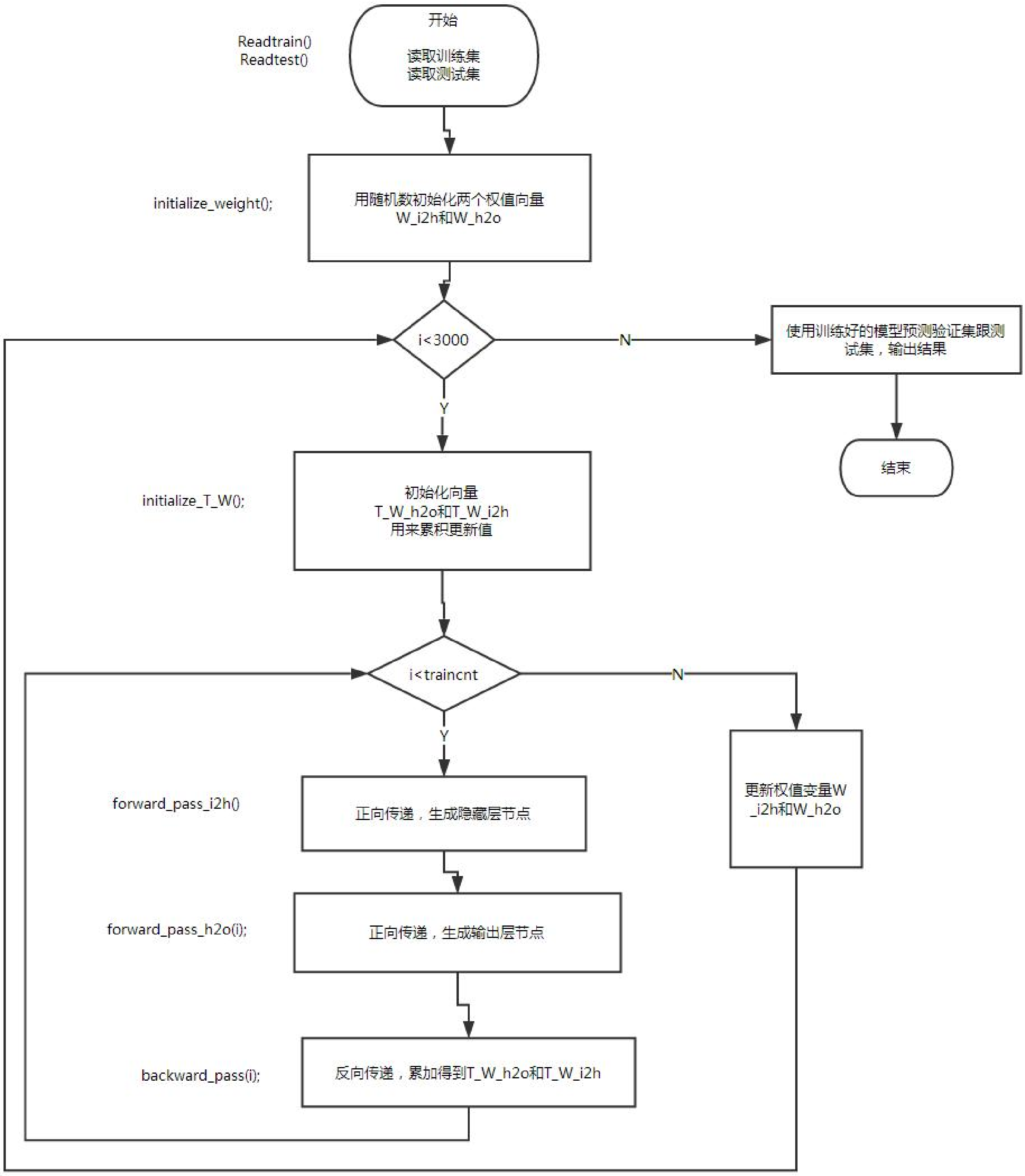
$$\Delta W_{ij} = -\eta \cdot (y - \hat{y}) \cdot W_j \cdot h_j \cdot (1 - h_j) \cdot x_i$$

以此来更新 Wij，则反向传递完毕。

执行以上过程，迭代 n 次，则能训练出符合要求的权值向量 Wij(输入层与隐藏层之间的权值向量)和 wj（隐藏层与输出层之间的权值向量）。当使用该神经网络模型预测测试集样本时，使用这两个向量，依次生成隐藏层和输出层，输出层结果即为预测结果。

至此 BP 神经网络的原理讲完。

1、伪代码


Readtrain()
Readtest()
开始
读取训练集
读取测试集
initialize_weight();
用随机数初始化两个权值向量
W_i2h和W_h2o
i<3000
N
使用训练好的模型预测验证集跟测试集，输出结果
结束
Y
initialize_T_W();
初始化向量
T_W_h2o和T_W_i2h
用来累积更新值
i<traincnt
N
更新权值变量W_i2h和W_h2o
Y
forward_pass_i2h()
正向传递，生成隐藏层节点
forward_pass_h2o(i);
正向传递，生成输出层节点
backward_pass(i);
反向传递，累加得到T_W_h2o和T_W_i2h

2. 关键代码截图（带注释）

```
int main()
{
    Readtrain();//读取训练集，且其中训练集每三个样本取前两个作为训练集，第三个作为验证集
    Readtest();//读取测试集
    cout<<"traincnt="<<traincnt<<" valicnt="<<valicnt<<" testcnt="<<testcnt<<endl;
    initialize_weight();//初始化两个权值向量
    for(int k=0;k<3000;k++)//训练3000次
    {
        MSE=0;
        initialize_T_W();//初始化△Wij和△Wi为全零向量
        for(int i=0;i<traincnt;i++)
        {
            if(k==2999) flag=1;//到训练3000次的最后一次时flag=1，把预测结果存储进数组
            else flag=0;
            for(int j=0;j<Length;j++) x[j]=train[i][j];//取出当前遍历到的样本的特征值放进x数组
            forward_pass_i2h(); //正向传递，从输入层到隐藏层
            forward_pass_h2o(i);//正向传递，从隐藏层到输出层
            backward_pass(i);//反向传递，累计△Wij和△Wi
        }
        Update_Weight();//每遍历完一次训练集更新一次权值向量
        cout<<"MSE="<<MSE<<endl;
    }
```

```
void initialize_weight()
{
    srand(time(0));
    //初始化输入层到隐藏层之间权值向量W_i2h
    for(int i=0;i<Length;i++)
        for(int j=0;j<hidenum;j++)
            W_i2h[i][j]=rand()*1.0/RAND_MAX*2;
    //初始化隐藏层到输出层之间权值向量W_h2o
    for(int i=0;i<hidenum;i++) W_h2o[i]=rand()*1.0/RAND_MAX*2;
    //readpreviousW();
}

void initialize_T_W()
{
    //初始化T_W_i2h，即△Wij
    for(int i=0;i<Length;i++)
        for(int j=0;j<hidenum;j++)
            T_W_i2h[i][j]=0;
    //初始化T_W_h2o，即△Wi
    for(int i=0;i<hidenum;i++) T_W_h2o[i]=0;
}

void forward_pass_i2h()
{
    h[0]=1;//为阈值所准备
    for(int i=1;i<hidenum;i++)
    {
        double in=0;//首先计算输入到隐藏层节点的值
        for(int j=0;j<Length;j++) in+=W_i2h[j][i]*x[j];
        h[i]=1/(1+exp(-1*in));//以sigmoid函数为激活函数，确定隐藏层节点数值
    }
}
```

```
void forward_pass_h2o(int index)
{
    double in=0;//首先计算输入到输出层节点的值
    for(int i=0;i<hidenum;i++) in+=W_h2o[i]*h[i];
    y=in;//以线性函数f(x)=x为激活函数，确定输出层节点数值
    if(flag==1){//预测结果存储进数组trainpredict
        trainpredict[tpcnt++]=y;
    }
    //累加均方误差
    MSE+=(y-trainlabel[index])*(y-trainlabel[index])/traincnt;
}
```

```
void backward_pass(int index)
{
    delta_out=trainlabel[index]-y;
    for(int i=0;i<hidenum;i++)
        delta_hide[i]=delta_out*W_h2o[i]*h[i]*(1-h[i]);
    //累加权值向量W_h2o的更新值
    for(int i=0;i<hidenum;i++) T_W_h2o[i]+=delta_out*h[i];
    //累加权值向量W_i2h的更新值
    for(int j=0;j<hidenum;j++)
        for(int i=0;i<Length;i++)
            T_W_i2h[i][j]+=delta_hide[j]*x[i];
}
```

```
void Update_Weight()
{
    double eta=0.001;//步长
    //更新W_i2h
    for(int j=0;j<hidenum;j++)
        for(int i=0;i<Length;i++)
            W_i2h[i][j]+=eta*T_W_i2h[i][j]/traincnt;
    //更新W_h2o
    for(int i=0;i<hidenum;i++) W_h2o[i]+=eta*T_W_h2o[i]/traincnt;
}
```

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

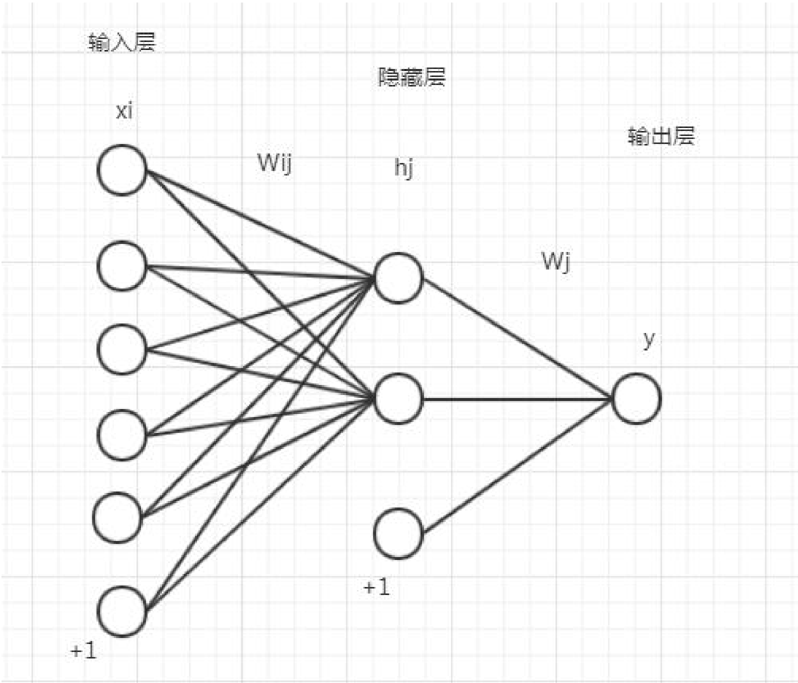
小数据集：

```
small_dataset_train
1,2011/1/1,1,1,0.4,0.2,0.1,1
2,2011/1/1,2,3,0.7,0.4,0.2,2
3,2011/1/1,1,1,0.4,0.2,0.1,1
```

除去不读取的内容，可以简化为：

```
1,1,0.4,0.2,0.1,1
2,3,0.7,0.4,0.2,2
1,1,0.4,0.2,0.1,1
```

共三条样本，每条样本加上阈值项共 6 个特征值。因此输入层节点有 6 个，隐藏层节点取 3 个，输出层节点一个。如下图：



第一个样本开始训练：

初始化两个权值向量为：

```
W_i2h
0.1 0.2 0.3
0.4 0.5 0.6
0.7 0.8 0.9
1 1.1 1.2
1.3 1.4 1.5
1.6 1.7 1.8

W_h2o
1.9 2 2.1
```

正向传递步骤一，生成隐藏层节点(总共 3 个节点，其中第一个节点用作阈值项)：

h[0]=1;(阈值项,不接受输出层传递的信息，因此 W_i2h 向量的第一列其实没有用到)

h[1]:

首先计算权值和输入层节点值乘积和:

in=W_i2h[0][1]*x[0] + W_i2h[1][1]*x[1] + W_i2h[2][1]*x[2] + W_i2h[3][1]*x[3] + W_i2h[4][1]*x[4] + W_i2h[5][1]*x[5] = 0.2*1 + 0.5*1 + 0.8*1 + 1.1*0.4 + 1.4*0.2 + 1.7*0.1 =2.39

然后应用 sigmoid 激活函数：

$$h[1]=\frac{1}{1+e^{-in}}=0.916062$$

同理：

h[2]:

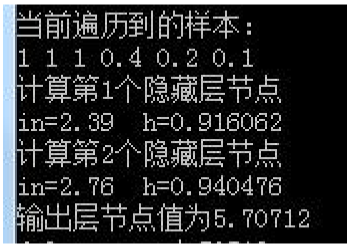
in=W_i2h[0][2]*x[0] + W_i2h[1][2]*x[1] + W_i2h[2][2]*x[2] + W_i2h[3][2]*x[3] + W_i2h[4][2]*x[4] + W_i2h[5][2]*x[5] = 0.3*1 + 0.6*1 + 0.9*1 + 1.2*0.4 + 1.5*0.2 + 1.8*0.1 =2.76

$$h[2]=\frac{1}{1+e^{-in}}=0.940476$$

正向传递步骤二，生成输出层节点：

y= W_h2o[0]*h[0] + W_h2o[1]*h[1] + W_h2o[2]*h[2] =5.7

以上正向传递计算结果与代码运行结果一致：



反向传递：

对权值向量 Wj 求梯度

$$\delta=(label-y)=1-5.7=-4.7$$

$$\Delta W_0=\delta\cdot h[0]=-4.7*1=-4.7$$

$$\Delta W_1=\delta\cdot h[1]=-4.7*0.916=-4.31$$

$$\Delta W_2=\delta\cdot h[2]=-4.7*0.940=-4.42$$

对权值向量 Wij 求梯度

$$\delta_0=\delta\cdot W\_h2o[0]\cdot h[0]\cdot(1-h[0])=-4.7*1.9*1*(1-1)=0$$

$$\delta_1=\delta\cdot W\_h2o[1]\cdot h[1]\cdot(1-h[1])=-4.7*2.0*0.916*(1-0.916)=-0.724$$

$$\delta_2=\delta\cdot W\_h2o[2]\cdot h[2]\cdot(1-h[2])=-4.7*2.1*0.94*(1-0.94)=-0.553$$

$$\Delta W_{00}=\delta_0\cdot x[0]=0*1=0$$

......

$$\Delta W_{01}=\delta_1\cdot x[0]=-0.724*1=-0.724$$

同理。。。

以上计算结果与代码运行结果一致：

```
delta_out=-4.70712
delta_hide:-0 -0.723887 -0.553372
T_W_h2o:
-4.70712 -4.31201 -4.42693
T_W_ioh:
0 0 0 0 0 0
-0.723887 -0.723887 -0.723887 -0.289555 -0.144777 -0.0723887
-0.553372 -0.553372 -0.553372 -0.221349 -0.110674 -0.0553372
```

当第一次遍历完所有三个训练集样本后更新权值向量，第二次遍历时，预测的结果分别为：

```
输出层节点值为4.44053
```
```
输出层节点值为4.68966
```
```
输出层节点值为4.44053
```
```
MSE=10.3029
```

这与真实标签的 1　2　1 相差甚远。均方误差 MSE 也很大。

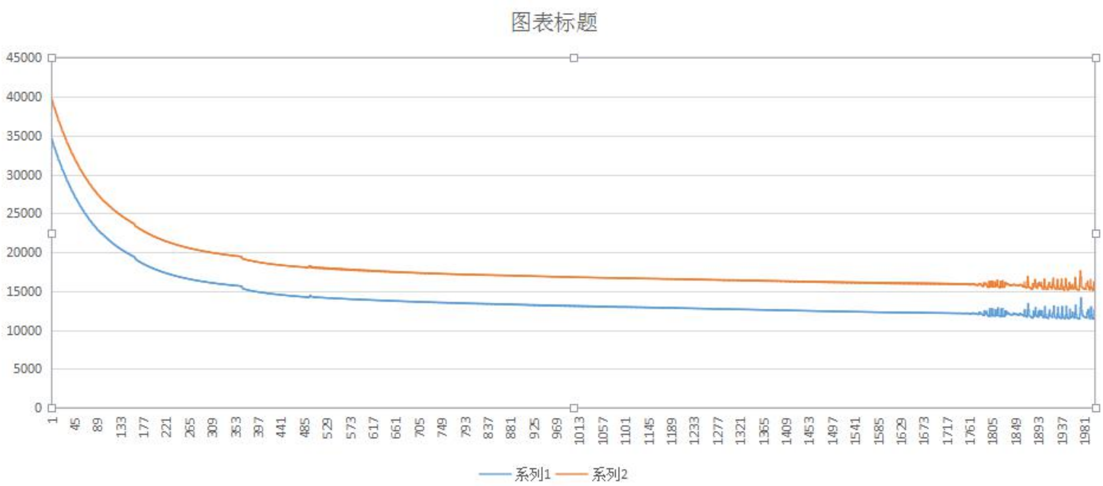
当遍历训练集 1000 遍后，预测的结果为：

```
输出层节点值为1.00844
```
```
输出层节点值为1.98556
```
```
输出层节点值为1.00844
```
```
MSE=0.000117003
```

已经非常接近真实标签值了，而且 MSE 也下降到非常小。

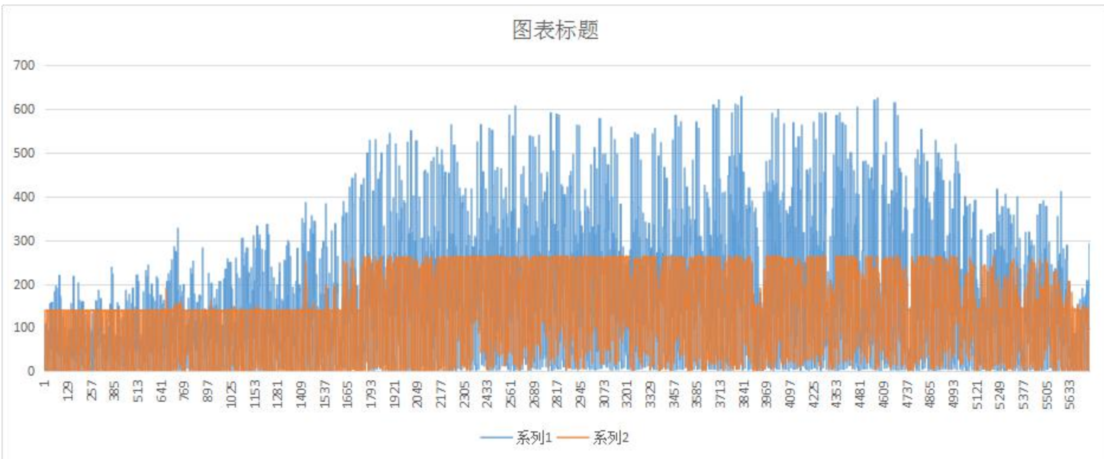
2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

下图使用 loss function，即 MSE（均方误差）作为评测指标，横坐标为训练次数，纵坐标为 MSE 值，蓝线为训练集，橙线为验证集。

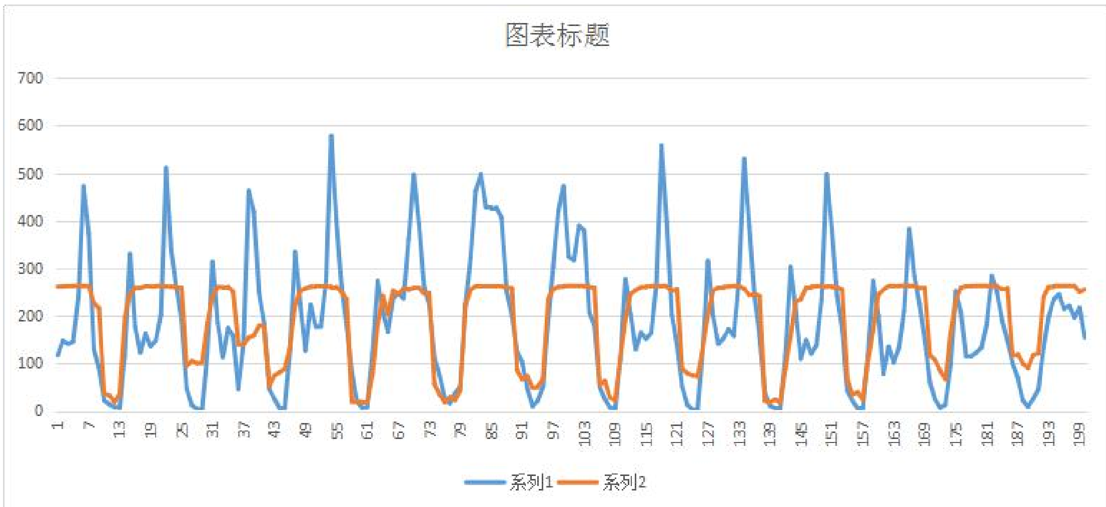


可见在不断训练模型的同时，MSE 在下降，说明模型正在学习优化，而验证集 MSE 始终低于训练集 MSE，也是合理的。

下图是训练集的真实值和预测结果对比图，蓝线是真实值，橙线是预测值。

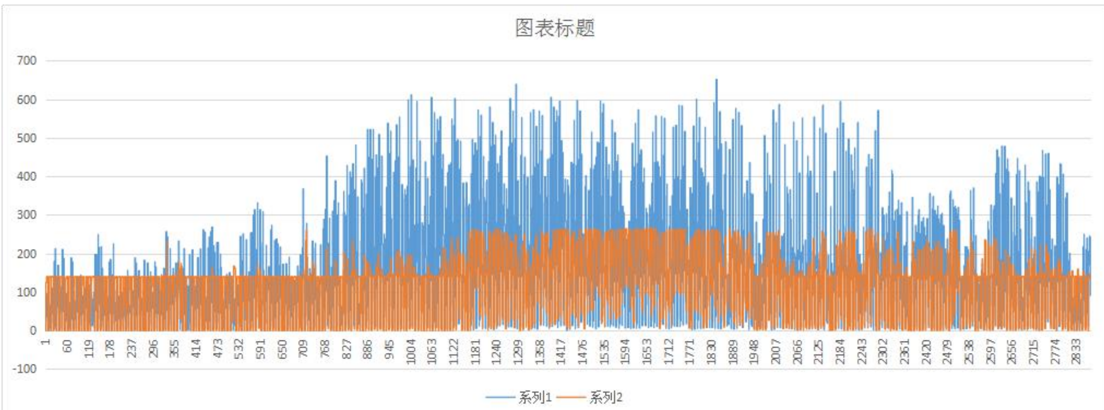


上图横坐标跨度过大，难以看出拟合效果，从中截取从 3000 到 3200 这一段的值分析，即下图：

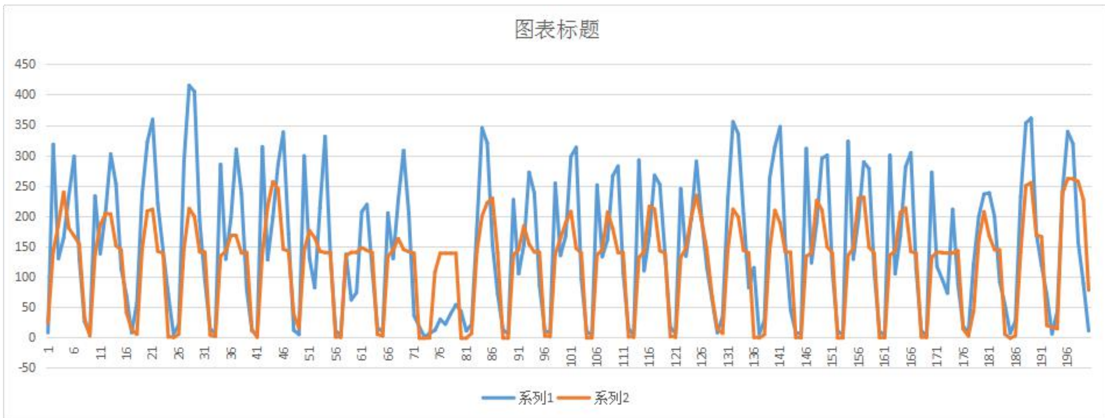


可见，真实值集合中，除了一些数值很大的极端值外，预测结果基本符合真实值，较好的拟合了真实值曲线的上升下降的趋势。

下图是验证集的真实值和预测结果对比图，蓝线是真实值，橙线是预测值。



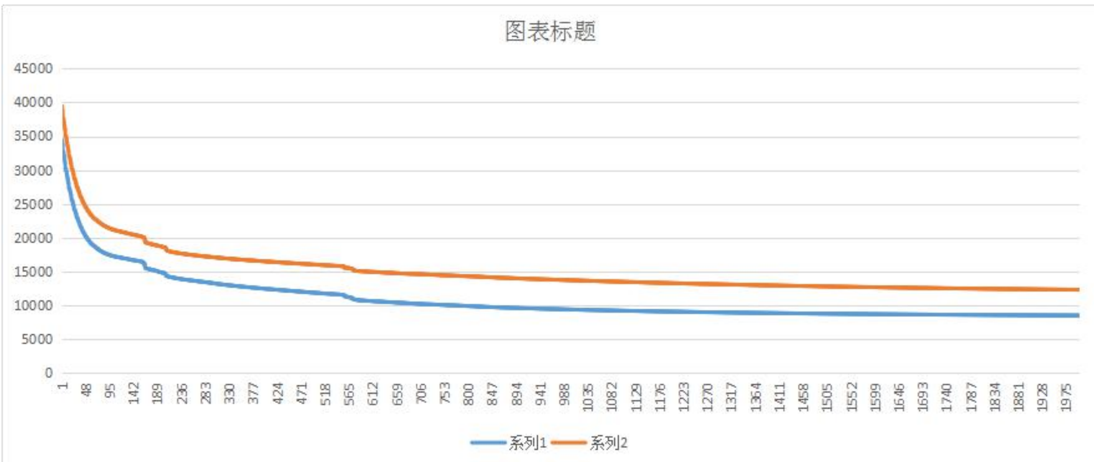
截取从 1000 到 1200 这一段的值分析，即下图：



预测结果较好的拟合了真实值。

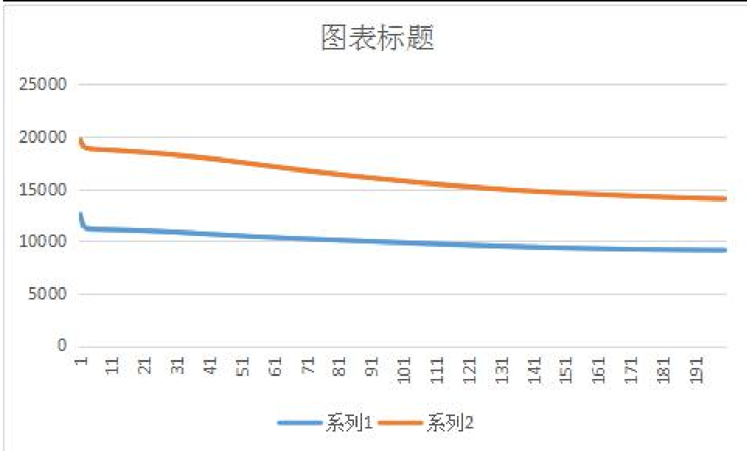
3、创新点&优化

（1）使用随机梯度下降方法。原本的做法是每一次迭代都遍历一次整个训练集，把每个样本的更新值累计起来，遍历完毕后才更新权值变量，但是这样耗费时间长，学习效率较低，需要跑很长时间才能到达 MSE=8000；而如果使用随机梯度下降方法，每次遍历一个样本就更新权值变量，能在更短时间内到达 MSE=8000。



（2）尝试使用其他激活函数，比如 Relu，即 y=max(0,x);

下图为迭代过程中 MSE 变化曲线，蓝色为训练集，橙色为验证集。



上图中，第一步优化就直接把 MSE 降低到 10000 多，而且只迭代 200 次就已经收敛。可见其优点是前面部分的梯度下降速度快，能在很短的时间内降低均方误差，但是递归到后面开始放慢，且过早收敛。

四、 思考题

1、尝试说明下其他激活函数的优缺点。

（1）首先 Sigmoid 是常用的非线性的激活函数，也是这次实验使用到的激活函数，它能够把输入的连续实值“压缩”到 0 和 1 之间。但 sigmoid 也有缺点，就是当输入非常大或者非常小的时候，这些神经元的梯度是接近于 0 的，模型优化跟学习将十分缓慢。

（2）Tanh 函数， $\tanh = 2sigmoid(2x) - 1$，优点是 sigmoid 的进阶版，0 均值，能够压缩数据到-1 到 1 之间；缺点是输出不是 0 均值的，这会导致后层的神经元的输入是非 0 均值的信号，这会对梯度产生影响，假设后层神经元的输入都为正，那么对 w 求局部梯度则都为正，这样在反向传播的过程中 w 要么都往正方向更新，要么都往负方向更新，使得收敛缓慢。

（3）ReLU 函数： $f(x) = \max(0, x)$。

优点：①因为是线性，而且梯度不会饱和，所以收敛速度会比 Sigmoid/tanh 快很多；②相比于 Sigmoid/tanh 需要计算指数等，计算复杂度高，ReLU 只需要一个阈值就可以得到激活值；
缺点：训练的时候很脆弱，有可能导致神经元坏死。由于 ReLU 在 x<0 时梯度为 0，这样就导致负的梯度在这个 ReLU 被置零，而且这个神经元有可能再也不会被任何数据激活，无法继续学习，准确率就得不到改进。

2、有什么方法可以实现传递过程中不激活所有节点？

正常传递过程中，当输入层的数值传递到隐藏层时，会通过激活函数将隐藏层节点激活。不激活所有节点：为隐藏层节点设置一个阈值，当判断到输入过大或过小时，不使用激活函数而是直接将隐藏层节点值设为 0。这样做能避免极端值影响传递过程。

3、梯度消失和梯度爆炸是什么？可以怎么解决？

随着神经网络层数的增加，会体现出深度神经网络中的梯度不稳定性，出现梯度消失或者梯度爆炸的问题。
深度神经网络训练的时候，采用的反向传播方式，为了更新权值向量，需要使用链式求导，计算每层梯度的时候会涉及一些连乘操作，因此如果网络过深，那么如果连乘的因子大部分小于 1，最后乘积可能趋于 0；另一方面，如果连乘的因子大部分大于 1，最后乘积可能趋于无穷。这就是所谓梯度消失与梯度爆炸。

解决方法比如有：
一种方式是设置梯度剪切阈值，一旦梯度超过改值，直接置为该值，能有效防止梯度爆炸。
另一种方式是使用 ReLU,maxout 等替代 sigmoid，比如使用 ReL 函数时：gradient = 0 (if x < 0), gradient = 1 (x > 0)。不会产生梯度消失问题。

|--------------- 如有优化，重复 1，2 步的展示，分析优化后结果 -------------|

PS：可以自己设计报告模板，但是内容必须包括上述的几个部分，不需要写实验感想