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	**1511**	专业（方向）	**软件工程（移动信息）**
学号	**15352237**	姓名	**刘顺宇**

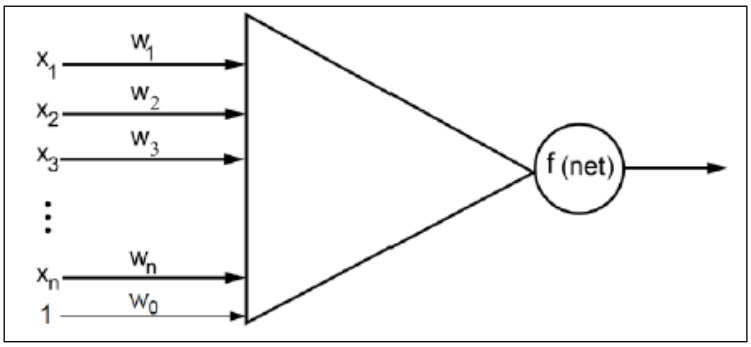
一、 实验题目

Back Propagation Neural Network

二、 实验内容

1. 算法原理

1.1. 简述：神经网络主要由许多的神经元组组成，而每个神经元的结构都如下所示，我们我们神经元都有一个输入信号，然后我们将输入信号乘以权重，再经过设置好的激活函数与阈值函数便可以得到我们当前神经元的输出，而当前神经元的输出可以作为别的神经元的输入信号，这样便构成了一个神经网络。神经网络由输入层，隐藏层和输出层构成，我们通过一开始的输入经过神经网络之后便可以得到最终的输出。而我们要得到一个效果较好的神经网络便需要通过前向传播反向传播更新权重向量进行训练。



1.2. 激活函数：

1.2.1. 此次实验的隐藏层即使用 sigmoid 函数作为激活函数。

$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$$

1.2.2. 此次实验中的输出层激活函数为$f(x) = x$，即为线性关系，输出值等于输入值。

1.3. 数据预处理：

1.3.1. 离散变量的处理：对离散属性进行编码表示，从一个输入节点变为多个输入节点，用 0 和 1 代表每个离散型数据的每个取值的输入，例如我们的 season 离散型数据有四种取值$\{1,2,3,4\}$，这时候我们就需要将这一个输入节点转化为四个输入节点，每个输入节点的取值只有 0 和 1 代表是与否，例如当 season 为 2 时编码则为$\{0,1,0,0\}$，只有这样离散化处理数据才能准确的进行神经网络的构造。

1.3.2. 归一化：属于输入属性的连续性取值，可将其属性取值进行归一化处理，把属性从有量纲的数转化为无量纲的数。

1.4. 算法流程：

1.4.1. 前向传播输入：

给定隐藏层或输入层的单元j，单位j的净输入I_j为

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

其中，w_{ij}是从上一层单元i到单元j的连接权重；O_i是上一层单元i的输出；θ_j为单元的阈值。

给定隐藏层的输入I_j，则单位j的输出O_j的公式为：

$$O_j = \frac{1}{1+e^{-I_j}}$$

给定输出层的输入I_k，则单位k的输出O_k的公式为：

$$O_k = I_k$$

1.4.2. 反向传播误差：

对于输出层中的单元k，节点误差计算公式为

$$E = \frac{1}{2}e^2 = \frac{1}{2}(T-O)^2$$

需要通过最小化节点误差来更新隐藏层到输出层的权重w_{jk}，首先计算在w_{jk}上的偏导数，经过推导可得

$$Err_k = (T_k - O_k)(-1)O_j$$

其中：T_k为输出节点的真实值，O_k为输出节点的预测值，O_j为隐藏层节点的输出值。

对于隐藏层中的单元j，通过E对输入层到隐藏层的权重w_{ij}求偏导数，经过推导可得

$$Err_j = -(T_k - O_k)w_{jk}O_j(1-O_j)x_i$$

隐藏层到输出层权重的更新公式为

$$w_{jk} = w_{jk} - \mu Err_k$$

输入层到隐藏层权重的更新公式为

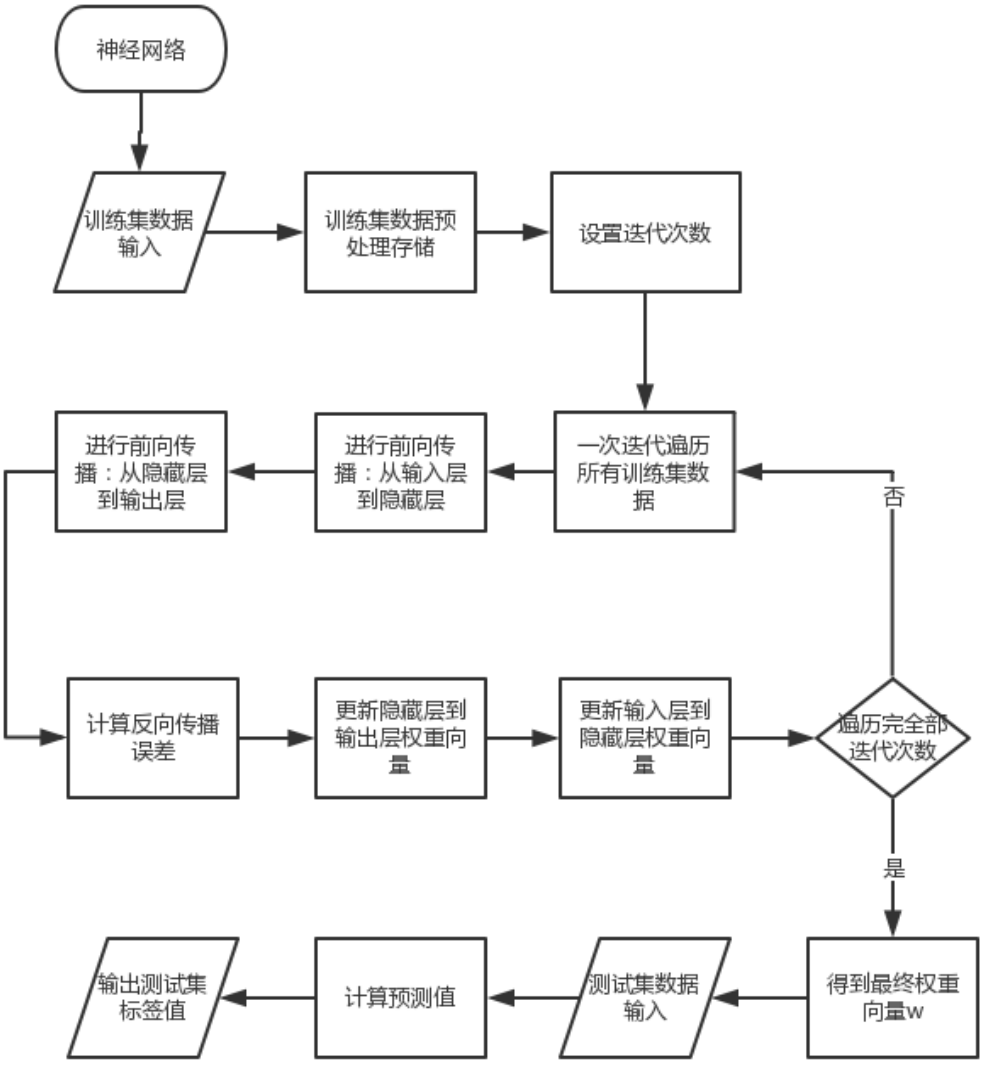
$$w_{ij} = w_{ij} - \mu Err_j$$

其中：μ为学习率

2. 伪代码流程图

2.1. 普通逻辑回归算法代码流程图（具体代码思想在关键代码解析中讲诉）

首先读取训练集进行处理，然后进行前向传播得到输出值计算误差然后再进行反向传播更新权重向量，最后通过权重向量来对测试集数据进行预测。



3. 关键代码截图（这里只解释最关键的建树过程，其他具体代码可看 cpp 文件，有详尽注释）

3.1. 随机初始化权重向量

我们首先需要随机初始化权重向量，不然迭代次数过多时结果可能会收敛到均值，而且随机初始化权重向量也有利于我们跳出局部最优解得到全局最优解。而且需要注意的是由于我们对输入层节点和隐藏层节点都做了向量扩展，所以我们需要初始化权重向量ω_0作为激活函数的阈值。

```
100   //初始化权重向量
101   void initialize_w() {
102       for (int j = 0; j <= hidden_set_num; j++) {
103           w_h2o[j] = (rand() % 100) / 10000.0;
104           if (j != 0) {
105               for (int i = 0; i <= input_set_num; i++) {
106                   w_i2h[i][j] = (rand() % 100) / 10000.0;
107                   //(rand() % 100) / 10000.0
108               }
109           }
110       }
111   }
```

3.2. 前向传播：从输入层到隐藏层

我们首先计算当前数据集从输入层到隐藏层的前向传播的隐藏层输出，在这里使用的基本的激活函数为 sigmod 函数，在这里我们对输入层节点做了向量扩展，所以权重向量中的ω_0作为激活函数的阈值。

```
113   //前向传播：从输入层到隐藏层
114   void forward_pass_i2h(double data[]) {
115       hidden_set[0] = 1.0;
116       for (int j = 1; j <= hidden_set_num; j++) {
117           hidden_set[j] = 0.0;
118           for (int i = 0; i <= input_set_num; i++) {
119               hidden_set[j] += data[i] * w_i2h[i][j];
120           }
121           //激活函数Relu
122           //if (hidden_set[j] > 0) hidden_set[j] = hidden_set[j];
123           //else hidden_set[j] = hidden_set[j] * 0.5;
124           //激活函数sigmod
125           hidden_set[j] = 1 / (1 + exp(-hidden_set[j]));
126       }
127   }
```

3.3. 前向传播：从隐藏层到输出层

接着我们计算数据集从隐藏层到输出层的前向传播的输出层输出，在这里使用的基本的激活函数为 f(x) = x 函数，在这里我们对隐藏层节点做了向量扩展，所以权重向量中的ω_0 作为激活函数的阈值。

```
129   //前向传播：从隐藏层到输出层
130   double forward_pass_h2o() {
131       double output = 0;
132       for (int j = 0; j <= hidden_set_num; j++) {
133           output += hidden_set[j] * w_h2o[j];
134       }
135       return output;
136   }
```

3.4. 反向传播：更新权重向量

在我们得到前向传播的输出时，便可以通过对误差函数$E = \frac{1}{2}e^2 = \frac{1}{2}(T-O)^2$求从输入层到隐藏层的权重向量和从隐藏层到输出层的权重向量分别求偏导便可以得到梯度下降的误差梯度，然后我们使用梯度下降法设置合理的步长来更新权重向量。

```
138   //反向传播：更新权重向量
139   void backward_pass(double output, Item set, int iter) {
140       //动态步长
141       //double alpha1 = (double)iter / 8000.0;
142       //double alpha2 = (double)iter / 8000.0;
143       //if (iter < 800) {
144       //  alpha1 = 0.1;
145       //  alpha2 = 0.1;
146       //}
147       //静态步长
148       double alpha1 = 0.01;
149       double alpha2 = 0.01;
150
151       //计算误差梯度更新从输入层到隐藏层的权重向量
152       for (int j = 1; j <= hidden_set_num; j++) {
153           //if (hidden_set[j] > 0) tem_grad2 = tem_grad1 * w_h2o[j];
154           //else tem_grad2 = tem_grad2 * 0.5 * w_h2o[j];
155           for (int i = 0; i <= input_set_num; i++) {
156               double hidden_errors_grad = (output - (double)set.value)
157                   * hidden_set[j] * (1.0 - hidden_set[j]) * w_h2o[j] * set.data[i];
158               w_i2h[i][j] = w_i2h[i][j] - hidden_errors_grad * alpha1  / (double)Train_row;
159           }
160       }
161       //计算误差梯度更新从从隐藏层到输出层的权重向量
162       for (int j = 0; j <= hidden_set_num; j++) {
163           double output_errors_grad = (output - (double)set.value) * hidden_set[j];
164           w_h2o[j] = w_h2o[j] - output_errors_grad * alpha2 / (double)Train_row;
165       }
166   }
```

4. 创新点&优化（为了避免重复，创新点算法没有在关键代码截图中提及）

4.1. 交叉验证法

由于这次实验只给了训练集没有验证集，所以自己去查看了交叉验证法的相应资料并加以实现，基本思路便是先将数据集随机打乱，然后将数据集分为 K 份（在这次实验中分了 9 份），然后交叉验证每次取 K-1 份数据作训练集，1 份作验证集，循环遍历 K 次，得到的 K 个准确率再取平均便得到了最终的准确率。由于代码过于简单但是过长所以在此处不再贴代码。

4.2. 动态步长

初始学习率较大，当梯度下降到接近最优值时，将学习率降低。在这里我们通过设计一条公式使得步长随着迭代次数的增加而增加便实现了动态步长。

```
140     //动态步长
141     double alpha1 = (double)iter / 8000.0;
142     double alpha2 = (double)iter / 8000.0;
143     if (iter < 800) {
144         alpha1 = 0.1;
145         alpha2 = 0.1;
146     }
```

4.3. Mini-batch 随机批梯度下降法

原本的更新权重向量公式使用批梯度下降，每次更新权重向量都考虑所有样本，而随机批梯度下降每次更新参数都只考虑一部分样本。而我们选取样本的办法采用了随机选取，使用了随机数随机从训练集数据中选取一部分数据计算梯度并更新权重向量。

```
259         //Mini-batch随机批处理神经网络
260         else if (method == 2) {
261             int Train_rand_num = 1000;//定义批梯度下降训练集个数
262             int exit[10000] = { 0, };
263             for (int i = 0; i < Train_rand_num;) {
264                 int rand_i = rand() % Train_row;//随机抽取训练集数据
265                 if (exit[rand_i] == 0) {
266                     exit[rand_i] = 1;
267                     forward_pass_i2h(Train_set[rand_i].data);//前向传播：从输入层到隐藏层
268                     double output = forward_pass_h2o();//前向传播：从隐藏层到输出层
269                     msn_train += pow(((double)Train_set[rand_i].value - output), 2);//计算msn
270                     backward_pass(output, Train_set[rand_i], iter);//反向传播：更新权重向量
271                     i++;
272                 }
273             }
274             msn_train = msn_train / (double)Train_rand_num;
275         }
```

4.4. 数据预处理（通过 excel 表格进行操作）

由于这次的数据既有离散型数据又有连续型数据，我们对离散型数据进行编码表示，用 0 和 1 代表每个离散型数据的每个取值的输入，例如我们的 season 离散型数据有四种取值$\{1,2,3,4\}$，这时候我们就需要将这一个输入节点转化为四个输入节点，每个输入节点的取值只有 0 和 1 代表是与否，例如当 season 为 2 时编码则为$\{0,1,0,0\}$，只有这样离散化处理数据才能准确的进行神经网络的构造。

而对于连续型数据，由于每个数据的量纲不同，所以我们需要对连续型数据进行归一化处理将有量纲的表达式，经过变换后转化为无量纲的表达式，在这里我们采用的归一化公式如下所示。

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

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）

1.1. 输入层有三个节点 1,2 3，值为$x_1 = 1, x_2 = 0, x_3 = 1$。输出层一个节点，标签为 1。

1.2. 隐藏层有两个节点 4,5，节点 4 的阈值为-0.4，节点 5 的阈值为 0.2。输入层到隐藏层的权值初始化如下所示：

$$\begin{pmatrix} w_{14} = 0.2 & w_{15} = -0.3 \\ w_{24} = 0.4 & w_{25} = 0.1 \\ w_{34} = -0.5 & w_{35} = 0.2 \end{pmatrix}$$

1.3. 输出层只有一个节点 6，节点的阈值为 0.1。隐藏层到输出层的权重初始化如下所示：

$$(w_{46} = -0.3 \quad w_{56} = -0.2)$$

1.4. 输入节点进行 1 拓展后为$x_0 = 1, x_1 = 1, x_2 = 0, x_3 = 1$。将隐藏层节点 4,5 的阈值转换为$w_{04}$和$w_{05}$，输入层到隐藏层的 w 对 x 加权求和后得到隐藏层节点 4,5 的输入为

$$x_{4\_in} = 0.2 + 0 - 0.5 - 0.4 = -0.7$$
$$x_{5\_in} = -0.3 + 0 + 0.2 + 0.2 = 0.1$$

1.5. 隐藏层的激活函数为$1/(1 + e^{-x_in})$，因此节点 4,5 的输出为

$$x_{4\_out} = 1/(1 + e^{0.7}) = 0.332$$
$$x_{5\_out} = 1/(1 + e^{-0.1}) = 0.525$$

1.6. 隐藏层的输出进行 1 拓展后变为$(1 \quad 0.332 \quad 0.525)$，将输出节点 6 的阈值转为w，即得到隐藏层到输出层的权值为$(0.1 \quad -0.3 \quad -0.2)$。对隐藏层的输出加权求和可得到输出层的输入为

$$x_{6\_in} = -0.3 * 0.332 - 0.2 * 0.525 + 0.1 = -0.105$$

输出层和隐藏层的激活函数一致，因此

$$x_{6\_out} = 1/(1 + e^{0.105}) = 0.474$$

1.7. 反向传播误差，利用公式$Err_k = O_k(1 - O_k)(T_k - O_k)$，计算输出层的节点误差如下：

$$Err_6 = 0.474 * (1 - 0.474)(1 - 0.474) = 0.1311$$

利用公式$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$计算隐藏层节点误差如下：

$$Err_5 = 0.525 * (1 - 0.525) * 0.1311 * (-0.2) = -0.0065$$
$$Err_4 = 0.332 * (1 - 0.332) * 0.1311 * (-0.3) = -0.0087$$

1.8. 利用权重更新公式$w_{jk} = w_{jk} + 学习率 * Err_k * O_j$，其中学习率取 0.9 更新每一维的 w 如下所示

$$w_{46} = -0.3 + 0.9 * 0.1311 * 0.332 = -0.261$$
$$w_{56} = -0.2 + 0.9 * 0.1311 * 0.525 = -0.138$$
$$w_{14} = 0.2 + 0.9 * (-0.0087) * 1 = 0.192$$
$$w_{15} = -0.3 + 0.9 * (-0.0065) * 1 = -0.306$$
$$w_{24} = 0.4 + 0.9 * (-0.0087) * 0 = 0.4$$
$$w_{25} = 0.1 + 0.9 * (-0.0065) * 0 = 0.1$$
$$w_{34} = -0.5 + 0.9 * (-0.0087) * 1 = -0.508$$
$$w_{35} = 0.2 + 0.9 * (-0.0065) * 1 = 0.194$$

因为 1 拓展的输出衡为 1，因此阈值的更新公式为$\theta_k = \theta_k + 学习率 * Err_k$，更新隐藏层和输出层的阈值如下所示：

$$\theta_6 = 0.1 + 0.9 * 0.1311 = 0.218$$
$$\theta_5 = 0.2 + 0.9 * (-0.0065) = 0.194$$
$$\theta_4 = -0.4 + 0.9 * (-0.0087) = -0.408$$

1.9. 运行程序计算输出结果，经过对比分析可得与上述计算结果一致

```
隐藏层节点输出值为：0.331812 0.524979

输出层节点输出值为：0.473774

更新后的隐藏层到输出层的权值为：-0.260849 -0.138058

更新后的隐藏层节点的阈值为：0.21799

更新后的输入层到隐藏层的权值为：
0.19217 -0.30585
0.4 0.1
-0.50783 0.19415

更新后的隐藏层节点的阈值为：0.19415 -0.40783
```

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）

2.1. 最优结果（交叉验证法）

训练模型	准确率
Mini-batch	3272.44

2.2. 优化结果分析（交叉验证法）

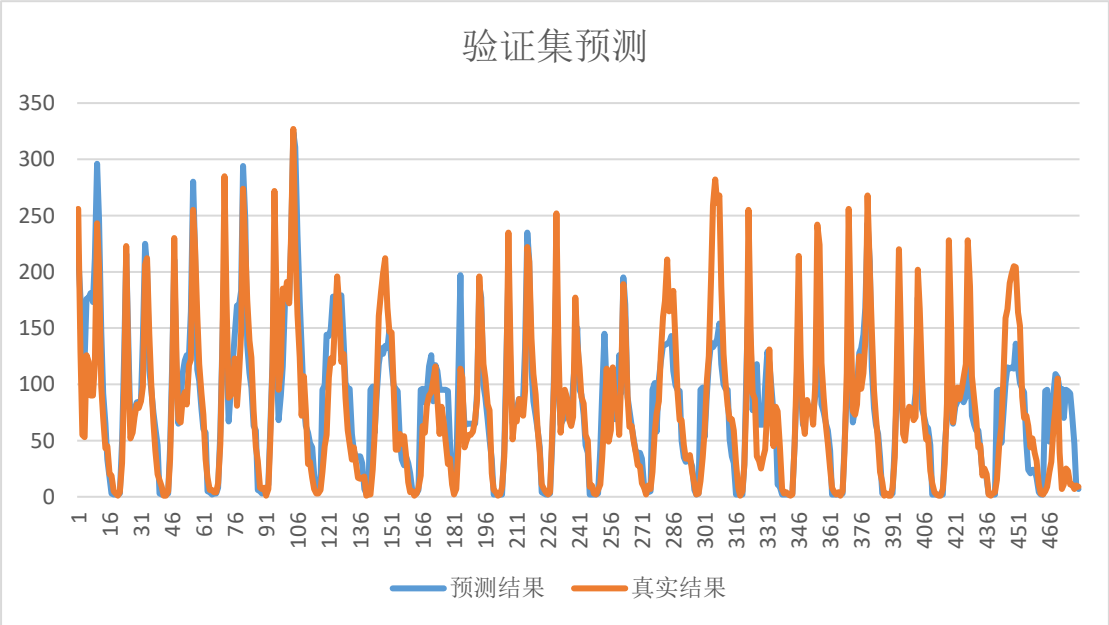
我们可以看到本次实验优化前后的 msn 及速度有较明显的提高，**初步分析是此次数据集本身也是比较易分的，而且数据集数量比较庞大**，在使用神经网络模型时，当我们调整学习率及隐藏层节点个数时，对结果的影响比较巨大，msn 的变化范围及变化速度相差十分大，但是最后只要有合适的学习率基本都能下降到一个较小的 msn，所以我们可以看到其实神经网络模型就是为了通过合理的下降梯度寻找一个最佳的权重向量 w，**而不同的优化都只是为了更加快速更加方便的下降梯度得到最优的权重向量**，而调参也是如此，所以我们神经网络模型受参数的影响是巨大，参数调整好了也是可以得到一个最佳解。

注：由于这次实验数据集被随机打乱，且随机梯度下降也是概率事件，所以下面展示数据具有一定随机性。

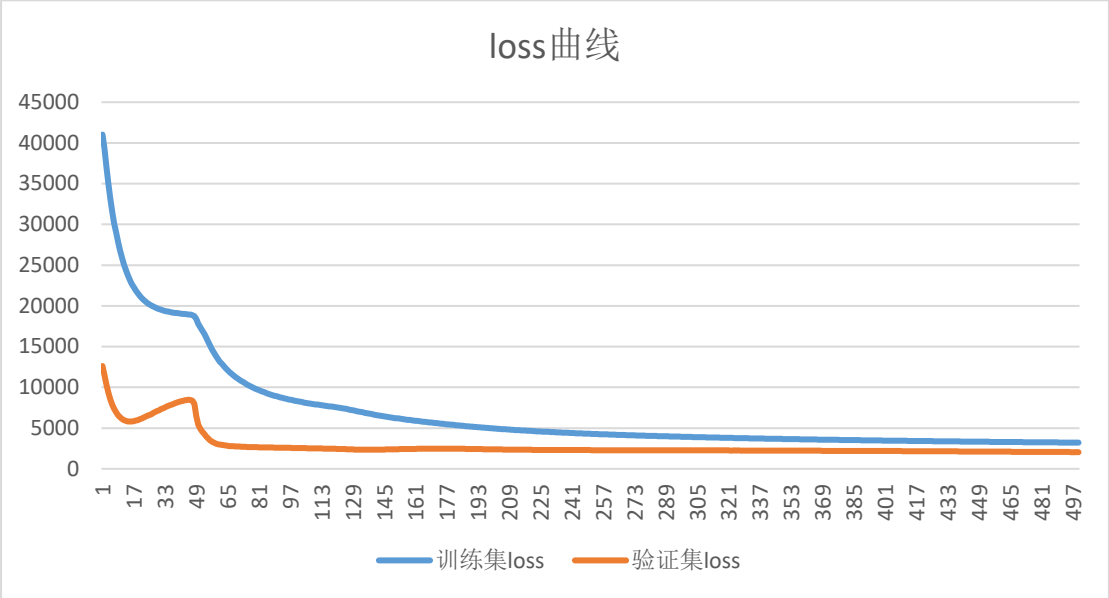
2.2.1. 测试优化模型

训练模型	MSN
普通神经网络	4002.45
Mini-batch	3272.44

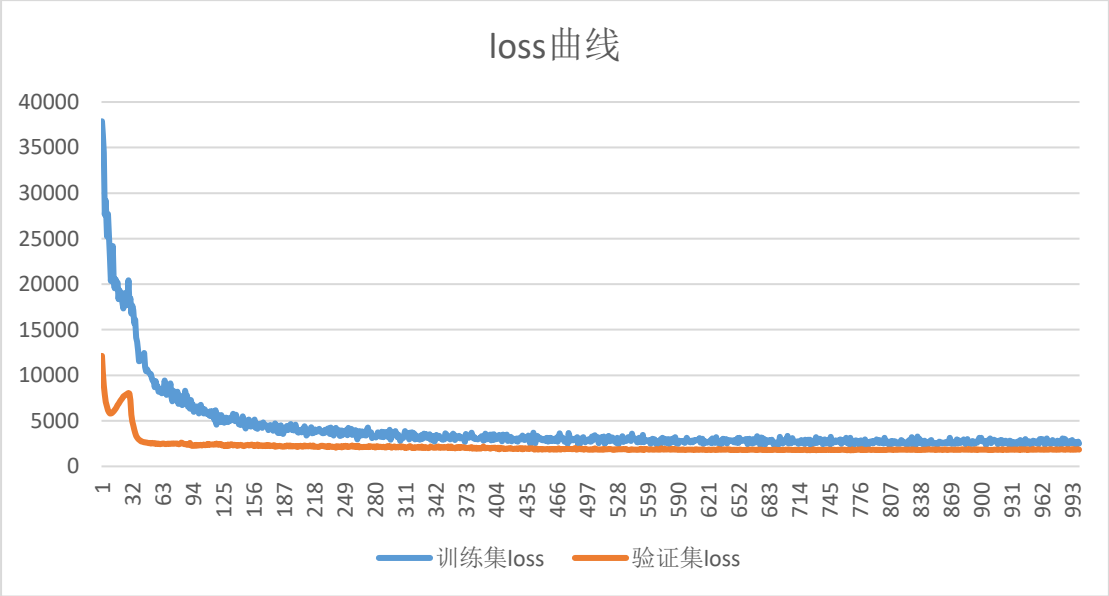
2.2.2. 验证集预测结果与真实结果比较曲线，由下图可得神经网络模型预测结果较拟合。



2.2.3. 普通神经网络训练集与验证集 loss 变化曲线，由图观察可得两条曲线除了在一开始的迭代由略微的起伏，在后面的迭代中呈平缓下降趋势。



2.2.4. Mini-batch 神经网络训练集与验证集 loss 变化曲线，由图观察可得两条曲线总体来说是下降的，但是在下降过程中有明显的震荡，总体呈现震荡收敛的趋势。



四、 思考题

1. 尝试说明下其他激活函数的优缺点。

1.1. sigmiod 函数

这次实验隐藏层用的是 sigmiod 激活函数，形式如下所示：

$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$$

此激活函数可以将输入的数值压缩到 0~1 的范围内，输入值越大，输出值越接近 1，输入值越小，输出值越接近 0。sigmiod 的优点是具有连续可导性，而且很清晰的表达了激活的含义，图像也接近与理想阶跃函数的形式。但其缺点是只有在 0 附近有比较好的激活性，即存在梯度，但是在输入值过大或过小的地方 sigmiod 函数为水平线，即饱和时梯度消失。

1.2. ReLu 函数和 Leaky ReLu 函数

ReLu 激活函数表达形式如下所示：

$$y = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

ReLu 激活函数相对于 sigmiod，因为其在$x \geq 0$时为线性关系，且不会出现饱和线性，即梯度为 0 的情况，且计算形式比较简单。经过实验尝试发现此函数可加快梯度下降的速率。但其也存在缺点，在x =0 处这个函数是连续不可导的，而且因为在$x < 0$的时候梯度为 0，因此如果有梯度为负的数经过该神经元节点，则会被置 0，导致在今后的迭代中此神经元节点不再被任何数据激活，发生数据丢失的情况。

Leaky ReLu 激活函数可以解决上述说的问题，其表达形式如下：

$$y = \begin{cases} x & x \geq 0 \\ ax & x < 0 \end{cases}$$

与 ReLu 不同，Leaky ReLu 在$x < 0$的时候值不再为 0，而是一个斜率较小的数，修正了数据分布，这样在有负梯度输入的时候，信息便不会丢失。

1.3. Tanh 函数

Tanh 函数即双曲正切函数，表达形式如下：

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

此激活函数关于坐标 0 点对称，把输入值压缩在-1~1 之间，和 sigmiod 函数具有类型性，且连续可导。当输入大于 0 时，输出为非线性上升，当输入小于 0 时，输出为非线性下降，相比于 sigmiod 函数，Tanh 函数延长了梯度到达 0 的输入 x 范围，即延迟了饱和期，且 Tanh 的输出基本是 0 均值的。但是 Tanh 的缺点是仍然存在梯度饱和的问题，

2. 有什么方法可以实现传递过程中不激活所有节点？

2.1. 如果希望再传递过程中不激活某些节点，可以将这些节点的激活函数暂时设为 f(x)=0，这样就可以不激活当前的神经元，且不变动输入层和输出层的节点。

2.2. 通过查阅资料了解到不激活所有节点主要应用在 dropout 技术，我们在训练时会随机的临时的删除一般的神经元（不激活某些节点），然后保持输出层和输出层不做变动，这种做法可以大大的改善神经网络耗时长和过拟合的缺点。

3. 梯度消失和梯度爆炸是什么？可以怎么解决？

3.1. 梯度消失：首先我们需要理解神经网络更新权重的办法，它首先根据前向传播得到一个预测输出，然后计算得到预测输出与真实输出的误差 E，接着我们便通过梯度下降法的思想来进行反向传播依次更新每一层隐藏层的权重向量，而这个反向传播是一层层进行传播的，而随着反向传播的层数的加深，我们对误差求偏导数的链式求导的乘积数也会逐渐加深，而如果我们使用的激活函数是类似 sigmoid 函数的 0-1 的激活函数，我们的乘积结果便会越来越小，例如对 sigmoid 函数求导便是$O_j(1-O_j)$，这个两个 0-1 之间的数进行相乘便会得到一个更小的数，这样就会导致随着反向传播的层数加深导致梯度逐渐消失。

总的来说，梯度消失就是由于神经网络的层数过多，反向传播中求偏导数的链式乘积项过多导致在传播的过程中使得某一层的传播梯度过低，电脑精度不够导致判断为 0，这时候就导致无法继续传播更新之后的隐藏层的权重向量，便造成了梯度消失的后果。

3.2. 梯度爆炸：一种原因是由于我们的学习率步长设置的过长，导致我们在进行反向传播时，权重向量不收敛反而发散，这是由于权重向量是一个凸函数的，如果步长过大则会导致权重向量反而随着梯度往增加的方向不断震荡，从而发散出去。

另一种原因便是由于我们的权重向量的初始化设置不合理或者使用了某些激活函数，就像梯度下降时分析的那样，由于随着神经网络反向传播层数的加深，我们的反向传播中求偏导数的链式乘积项过多，导致累乘梯度的结果迅速增长，这个时候我们会发现在进行反向传播时前面层的权重更新变化会比后面层变化的更快，导致我们的权重向量越来越大，发散出去，便导致了梯度爆炸的情况。

3.3. 如果要解决梯度消失或梯度爆炸的问题可以考虑换不同的激活函数进行传播，比如使用 ReLU 函数来取代 sigmoid 函数，这样在链式求导乘积项过多的时候便可以避免随着乘积项的增多导致乘积结果越小的情况，而同样的用 ReLU 函数也可能导致发生梯度爆炸的情况，所以我们必须考虑选取不同的合适的激活函数进行神经网络模型的构造。

无论是梯度消失还是梯度爆炸，从本质上来说都是不稳定的梯度传递的问题，都是由于链式求导而造成的后果，所以我们要消除这种不稳定还有一种办法便是考虑对不同的层采用不同的学习率，通过调整合适的学习率或者动态学习率，使得我们的梯度传播更加稳定可靠，从而避免了梯度消失或者梯度爆炸的问题。